VDM modelling: Password Manager

MODDLING OF CRITICAL SYSTEMS

Name	Student Number
Anton Vigen Smolarz	201807327

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AARHUS UNIVERSITY

DECEMBER 5, 2022

Contents

1 Requirements

Bellow is given the the requirements for the system. Definitions for the requirements;

- The user or users er the actors interacting with the Password Manager. (Customers)
- The database maintainer is the actors owning and maintaining the database and password manager. (Owners)
- In Req. 6 the system is the user's system, on which the password manager client is running.

1.1 General

Req. 1 The Password Manager should be able to have multiple users.

Req. 2 Each User should have a password database.

Req. 3 Each password in the password database should hold the login information for an account, and some match information to where that account is used.

1.2 Access

Req. 4 For the user to be able to login the user must provide a correct password to the given username.

Req. 5 A device should be authorized for each user before the user can login through that device.

Req. 6 If the user has been inactive on the system for 5 minutes while logged in on the Password Manager, the system should log the user out.

1.3 Security

Req. 7 Encryption:

1. The user's password database should be encrypted while the user is logged out.
2. The user's password database should be decrypted while the user is logged in.

Req. 8 Only the logged in user should be able to access their database unencrypted. The database maintainer should not be able to decrypt it from the backend.

2 Appedix

Appendix 1: Coverage for “AccessManager.vdmpp”

```
class AccessManager is subclass of TimeObject
    instance variables
        user: User;
        database: Database := new Database();
        acceptedDevices: set of Device_t := {};
        encKey: [EncryptionKey] := nil;
        lastActiveTime: Seconds := 0;
        curTime: Seconds := 0;
        accessState: LockState := <Unlocked>;
        static timeout: Seconds := 60 * 5;
operations

    public AccessManager: String * String * Device_t ==> AccessManager
    AccessManager(userName, password, dev) == (user := new User(userName, password);
        Time`GetInstance().Register(self);
        encKey := user.GetEncryptionKey(new Password(password));
        AuthorizeDevice(dev);
        (dcl foo: bool; foo := Logout()));

    public Login: String * Device ==> bool
    Login(password, dev) == (
        if accessState = <Locked> then
        (if dev in set acceptedDevices then
        let pass = new Password(password) in
        (
            if user.IsCorrectPassword(pass) then
                let ek = user.GetEncryptionKey(pass) in
                (database.Unlock(ek);
                encKey := ek;
                accessState := <Unlocked> ;
                return true);
            io.print("User: ");io.print(GetUserName());io.print(", Login Failed: Wrong passoword.\

        ) else (io.print("User: ");io.print(GetUserName());io.print(", Login Failed: Device not au
        else (io.print("User: ");io.print(GetUserName());io.print(", Login Failed: User already lo
        return false
    )
    post accessState = <Unlocked> => database.GetLockState() = <Unlocked>;

    public Logout: () ==> bool
    Logout() ==
            (if accessState = <Unlocked> then
```

```
                (database.Lock(encKey);encKey:=nil; accessState := <Locked>;
                return true);
            io.print("User: ");io.print(GetUserName());io.print(", Logout Failed: User already log
            return false)
        post accessState = <Locked> and database.GetLockState() = <Locked>;
    public TimeTicked: Seconds ==> ()
    TimeTicked(currentTime) == (
        if encKey <> nil and lastActiveTime + timeout <= currentTime then
            (dcl foo: bool; foo := Logout();
                io.print("User: ");io.print(GetUserName());io.print(", Logged out! User was inacti
        curTime := currentTime
    )
    post currentTime = curTime and lastActiveTime <= curTime;

    public DoActivity: () ==> ()
    DoActivity() == lastActiveTime := curTime
    post lastActiveTime = curTime;

    public AuthorizeDevice: Device_t ==> ()
        AuthorizeDevice(dev) == acceptedDevices := acceptedDevices union {dev};

    public pure GetUserName: () ==> String
    GetUserName() == return user.GetUserName();

    public pure GetDatabase: () ==> [Database]
    GetDatabase() ==  (if database.GetLockState() = <Unlocked> then
    return database;
    return nil);

    public pure GetLoginState: () ==> LockState
    GetLoginState() == return accessState;

end AccessManager
```

Function or operation	Line	Coverage	Calls
AccessManager	12	100%	121
Login	19	100%	72
Logout	39	100%	171
TimeTicked	47	100%	171
DoActivity	56	100%	18
AuthorizeDevice	60	100%	124
GetUserName	63	100%	276
GetDatabase	66	100%	150
GetLoginState	71	100%	32
AccessManager.vdmpp		100%	1135

Table 2.1: Coverage statistics for `AccessManager.vdmpp`

Appendix 2: Coverage for "Database.vdmpp"

```
class Database is subclass of GLOBAL
instance variables
    private passwords: seq of Password := [];
    private eqState: LockState := <Unlocked>;

    inv forall password in seq passwords & password.GetMatchString() <> nil;
operations

    public Unlock: EncryptionKey ==> ()
        Unlock(encKey) ==
        ( for all i in set inds passwords do
            passwords(i).Decrypt(encKey);
        let isUnlocked = {password.GetEncryptionState() | password in seq passwords} in
        if(card isUnlocked = 1 and <PlainText> in set isUnlocked or len passwords = 0) then
            eqState  := <Unlocked>)
        pre eqState = <Locked> and forall password in seq passwords & password.GetEncryptionState
        post eqState = <Unlocked> and forall password in seq passwords & password.GetEncryptionSta

    public Lock: EncryptionKey_t ==> ()
        Lock(encKey) ==
        (for all i in set inds passwords do
            passwords(i).Encrypt(encKey);
            eqState := <Locked>)
        pre eqState = <Unlocked> and forall password in seq passwords & password.GetEncryptionStat
        post eqState = <Locked> and forall password in seq passwords & password.GetEncryptionState

    public pure GetLockState: () ==> LockState
        GetLockState() ==
        return eqState;

    public AddPassword: Password ==> ()
        AddPassword(password) == if password.GetMatchString() <> nil then passwords :=
            passwords ^ [password]
        else
            io.print("Password needs to have a match string to be added to the database.\n")
        pre eqState = <Unlocked> and password.GetEncryptionState() = <PlainText>
        post password.GetMatchString() <> nil => len passwords = len passwords~ + 1;

    public GetAllPasswords: () ==> seq of Password
        GetAllPasswords() == return passwords;

    public FindPasswords: String ==> seq of Password
        FindPasswords(match) ==
            (dcl ret : seq of Password := [];
                for all i in set inds passwords do
                    if passwords(i).GetMatchString() = match then
                        ret := ret ^ [passwords(i)];
                return ret;
            )
        pre eqState = <Unlocked>;

    public RemovePassword: nat ==> ()
        RemovePassword(idx) == if idx <= len passwords then
            passwords := [passwords(k) | k in set inds passwords & k <> idx]
        pre eqState = <Unlocked>;
end Database
```

Function or operation	Line	Coverage	Calls
Unlock	8	100%	60
Lock	18	100%	169
GetLockState	26	100%	391
AddPassword	30	100%	203
GetAllPasswords	38	100%	17
FindPasswords	41	100%	10
RemovePassword	51	100%	4
Database.vdmpp		100%	854

Table 2.2: Coverage statistics for `Database.vdmpp`

Appendix 3: Coverage for "Device.vdmpp"

```
class Device is subclass of GLOBAL
    instance variables
        deviceId: token;
        static idCounter: int := 0;
    operations
    public Device: () ==> Device
        Device() == (deviceId := mk_token(deviceId);
        idCounter := idCounter + 1);

    public pure GetToken: () ==> token
        GetToken() == return deviceId;
end Device
```

Function or operation	Line	Coverage	Calls
Device	5	100%	54
GetToken	9	100%	36
Device.vdmpp		100%	90

Table 2.3: Coverage statistics for `Device.vdmpp`

Appendix 4: Coverage for "EncryptionKey.vdmpp"

```
class EncryptionKey
instance variables
    public keyToken : token;
operations
    public EncryptionKey: int * Password ==> EncryptionKey
    EncryptionKey(keyId, passwd) ==
        (keyToken := mk_token({keyId, passwd.GetPasswordToken()}));

    public pure GetToken: () ==> token
    GetToken() == return keyToken;
end EncryptionKey
```

Function or operation	Line	Coverage	Calls
EncryptionKey	4	100%	361
GetToken	8	100%	408
EncryptionKey.vdmpp		100%	769

Table 2.4: Coverage statistics for `EncryptionKey.vdmpp`

Appendix 5: Coverage for "Environment.vdmpp"

```
class Environment is subclass of TEST
types
    private Actions = <Login> | <Logout> | <FindPass> | <AddPass> | <RMPass> | <AuthDevice> | <New
    private UserToCreate = (String * String * seq of (String * String * String));
    private ActionsToDo = ([Seconds] * [Actions] * (LoginArgs | LogoutArgs | FindPassArgs | AddPas

    private LoginArgs = (String * nat);

    private LogoutArgs = [nat];

    private FindPassArgs = String;

    private AddPassArgs = (String * String * String);

    private RMPassArgs = nat;

    private AuthDeviceArgs = nat;

    private NewDeviceArgs = [nat];

    private FindUserArgs = String;

    private inline =(seq of UserToCreate * seq of ActionsToDo);

instance variables
    users : UserDatabase := new UserDatabase();
    actions : seq of ActionsToDo;
    currentUser : [AccessManager] := nil;
    devices : seq of Device_t := [new Device()];
    time : Time := Time`GetInstance();
operations
    public Environment :String==>Environment
    Environment(fname) ==
        def mk_(-,mk_(allUsers, allActions))=io.freadval[inline](fname)
        in
            (actions := allActions;
             for all i in set inds allUsers do
                (dcl acMa : AccessManager,
                    foo: bool;
                    foo := users.AddUser(allUsers(i).#1, allUsers(i).#2, devices(1));
                    acMa := users.FindUser(allUsers(i).#1);
                    foo := acMa.Login(allUsers(i).#2, devices(1));
                    for all k in set inds allUsers(i).#3 do
                        acMa.GetDatabase().AddPassword(new Password(allUsers(i).#3(k).#1,allUsers
                    foo := acMa.Logout();
                ));

    public Run :()==>()
        Run() ==
            while len actions > 0 do
                (dcl
                    -- Preamble --
                    action : ActionsToDo := hd actions;
                    def mk_(timeTick, actionType, args) = action in(
                    io.print("\n-----------------\n");
                    -- Handle action --
                    if timeTick <> nil and timeTick <> 0 then
```

```
                    (time.TickTime(timeTick);
                    io.print("Time passed: ");
                    if(timeTick > 60) then
                        (io.print(timeTick div 60);io.print(" minutes "));
                    if((timeTick mod 60) <> 0) then
                        (io.print(timeTick mod 60);io.print(" seconds"));
                        io.print("\n"););

                cases actionType:
                <Login> -> HandleLogin(args),
                <Logout> -> HandleLogout(args),
                <FindPass> -> HandleFindPass(args),
                <AddPass> -> HandleAddPass(args),
                <RMPass> -> HandleRMPass(args),
                <AuthDevice> -> HandleAuthDevice(args),
                <NewDevice> -> HandleNewDevice(args),
                <FindUser> -> HandleFindUser(args)
                end;
                -- Postamble --
                actions := tl actions;
                io.print("-----------------\n");
            ));


HandleLogin: LoginArgs ==> ()
HandleLogin(args) ==
    def mk_(password, devIdx) = args in
        if currentUser.Login(password, devices(devIdx)) then
            io.print("Login success!\n")
        else io.print("Login failed!\n");

HandleLogout: LogoutArgs ==> ()
HandleLogout(-) ==
    if currentUser.Logout() then
        io.print("Logout success!\n")
    else io.print("Logout failed!\n");

HandleFindPass: FindPassArgs ==> ()
    HandleFindPass(match) == (dcl database : [Database] := currentUser.GetDatabase(),
                                matchedPasses : seq of Password;
        if database <> nil then
            (matchedPasses := database.FindPasswords(match);
            if len matchedPasses > 0 then
                (io.print(len matchedPasses);io.print(" passwords was found matching \"");io.p
                for all i in set inds matchedPasses do
                    matchedPasses(i).PrintPassword())
            else (io.print("No passwords was found matching \"");io.print(match);io.print("\".
        else io.print("Password could not be found. Database is locked.\n"););

HandleAddPass: AddPassArgs ==> ()
HandleAddPass(args) ==
    (dcl database : [Database] := currentUser.GetDatabase(),
    pass : Password;
     if database <> nil then
        def mk_(userName, password, match) = args in
            ( pass := new Password(userName, password, match);
                database.AddPassword(pass);
                io.print("Following password was added:\n");
                pass.PrintPassword())
     else io.print("Password could not be added. Database is locked.\n"));
```

```
    HandleRMPass: RMPassArgs ==> ()
    HandleRMPass(idx) ==
        (dcl database : [Database] := currentUser.GetDatabase(),
        pass : Password;
         if database <> nil then
                (pass := database.GetAllPasswords()(idx);
                 database.RemovePassword(idx);
                 io.print("Following password was removed:\n");
                 pass.PrintPassword())
         else io.print("Password could not be added. Database is locked.\n"));

    HandleNewDevice: NewDeviceArgs ==> ()
    HandleNewDevice(-) ==
        ( devices := devices ^ [new Device()];
          io.print("Device # ");io.print(len devices);io.print(" was added.\n");
        );

    HandleAuthDevice: AuthDeviceArgs ==> ()
        HandleAuthDevice(idx) == (currentUser.AuthorizeDevice(devices(idx));
         io.print("Device # ");io.print(idx);io.print(" was authorized to user: \n");io.print(curr
        );

    HandleFindUser: FindUserArgs ==> ()
    HandleFindUser(userName) == (currentUser := users.FindUser(userName);
        if currentUser <> nil then
            (io.print("User: ");io.print(userName);io.print(" was found!\n"))
           else
            (io.print("Could not find the user ");io.print(userName);io.print(".\n"))
        );


end Environment
```

Function or operation	Line	Coverage	Calls
Environment	31	100%	1
Run	47	100%	15
HandleLogin	81	100%	3
HandleLogout	88	67%	1
HandleFindPass	94	82%	3
HandleAddPass	106	71%	1
HandleRMPass	118	100%	2
HandleNewDevice	129	100%	1
HandleAuthDevice	135	100%	1
HandleFindUser	140	66%	1
Environment.vdmpp		90%	29

Table 2.5: Coverage statistics for `Environment.vdmpp`

Appendix 6: Coverage for "GLOBAL.vdmpp"

```
class GLOBAL
types
    public String = seq of char;

    public Seconds = nat;

    public Device_t = Device
        eq a = b == a.GetToken() = b.GetToken();

    public Password_t = Password
        eq a = b == Password`PasswordCompaire(a,b);

    public EncryptionKey_t = EncryptionKey
        eq a = b == a.GetToken() = b.GetToken();

    public OptEncryptionKey = [EncryptionKey_t]
        eq a = b == if a = nil or b = nil then a = nil and b = nil else (a.GetToken() = b.GetToken

    public LockState = <Locked> | <Unlocked>;
instance variables
    public static io: IO := new IO();

end GLOBAL
```

Function or operation	Line	Coverage	Calls
GLOBAL.vdmpp		100%	0

Table 2.6: Coverage statistics for `GLOBAL.vdmpp`

Appendix 7: Coverage for “Password.vdmpp”

```
class Password is subclass of GLOBAL
types
    public State = <Encrypted> | <PlainText>;
instance variables
    password: String;
    enqState: State;
    enqKey: OptEncryptionKey;
    userName: [String] := nil;
    matchData: [String] := nil;

inv enqState = <Encrypted> => enqKey <> nil;

operations

    public Password: String ==> Password
    Password(keyPhrase) ==
        (password := keyPhrase;
         enqState := <PlainText>;
         enqKey := nil;
        );

    public Password: String * String ==> Password
    Password(keyPhrase, match) ==
        (password := keyPhrase;
         enqState := <PlainText>;
         enqKey := nil;
         matchData := match;
        );

    public Password: String * String * String ==> Password
    Password(user, keyPhrase, match) ==
        (password := keyPhrase;
         enqState := <PlainText>;
         enqKey := nil;
         matchData := match;
         userName := user;
        );


    public pure GetEncryptionState: () ==> State
    GetEncryptionState() == return enqState;

    public static pure PasswordCompaire: Password * Password ==> bool
        PasswordCompaire(cmp1, cmp2) ==
            return cmp1.password = cmp2.password and cmp1.enqState = cmp2.enqState and cmp1.enqKey

    public GetPasswordToken: () ==> token
        GetPasswordToken() ==
            return mk_token(password);

    public Encrypt: EncryptionKey_t ==> ()
    Encrypt(key) ==
    atomic (enqState := <Encrypted>;
     enqKey := key)
    pre enqState = <PlainText>
    post enqState = <Encrypted>;
```

```
    public Decrypt: EncryptionKey_t ==> ()
    Decrypt(key) ==
    if key = enqKey
    then
        atomic (enqState := <PlainText>;
         enqKey := nil)
    pre enqState = <Encrypted>
    post key = enqKey~ => enqState = <PlainText>;

    public GetUserName: () ==> [String]
    GetUserName() == return userName;

    public pure GetMatchString: () ==> [String]
        GetMatchString() == return matchData;

    public PrintPassword: () ==> ()
    PrintPassword() == (
        if matchData <> nil then
        (io.print("Password to: ");io.print(matchData);io.print("\n"));
        if userName <> nil then
        (io.print("User Name: ");io.print(userName);io.print("\n"));
        io.print("Password: ");
        if enqState = <Encrypted> then
            io.print("********")
        else io.print(password);
        io.print("\n");
    );
end Password
```

Function or operation	Line	Coverage	Calls
Password	14	100%	577
GetEncryptionState	39	100%	900
PasswordCompaire	42	100%	344
GetPasswordToken	46	100%	361
Encrypt	50	100%	194
Decrypt	57	100%	102
GetUserName	66	100%	6
GetMatchString	69	100%	454
PrintPassword	72	92%	6
Password.vdmpp		98%	2944

Table 2.7: Coverage statistics for `Password.vdmpp`

Appendix 8: Coverage for "TEST.vdmpp"

```
class TEST is subclass of GLOBAL
    instance variables
    protected static BoolToNat: map bool to int := {true |-> 1, false |-> 0};
operations
    protected assert: seq1 of nat * seq1 of nat ==> bool
    assert(data, expected) == return data = expected
    post RESULT = true;
end TEST
```

Function or operation	Line	Coverage	Calls
assert	4	100%	219
TEST.vdmpp		100%	219

Table 2.8: Coverage statistics for `TEST.vdmpp`

Appendix 9: Coverage for "TestAccessManager.vdmpp"

```
class TestAccessManager is subclass of TEST
instance variables
    public passwordText: String := "UserPassword";
    public device: Device_t := new Device();
    public accessManager: AccessManager;
    time: Time := Time`GetInstance();
operations
    public TestAccessManager: () ==> TestAccessManager
    TestAccessManager() == (accessManager := new AccessManager("TestUser", passwordText, device);
            (dcl foo: bool; foo := accessManager.Login(passwordText, device));
            accessManager.GetDatabase().AddPassword(new Password("GamerTag", "YouShallNotPassword1
            accessManager.GetDatabase().AddPassword(new Password("BuisnessAccount", "buisness123",
            accessManager.GetDatabase().AddPassword(new Password("testmail@mail.com", "test123", '
            accessManager.GetDatabase().AddPassword(new Password("PersonalAccount", "birthday", "h
            (dcl foo: bool; foo := accessManager.Logout()););

    public testTimeout: seq of Seconds * [nat] ==> ()
    testTimeout(timeSeq, timeoutIdx) ==
        for all i in set inds timeSeq do
        (time.TickTime(timeSeq(i));
         if timeoutIdx <> nil and i >= timeoutIdx then
            (dcl foo: bool; foo := assert([BoolToNat(accessManager.GetLoginState() = <Locked>)],
            else
            (dcl foo: bool; foo := assert([BoolToNat(accessManager.GetLoginState() = <Unlocked>)],
            );

    public testDoActivity: seq of Seconds * [nat] ==> ()
    testDoActivity(timeSeq, timeoutIdx) ==
        for all i in set inds timeSeq do
        (time.TickTime(timeSeq(i));
         accessManager.DoActivity();
         if timeoutIdx <> nil and i >= timeoutIdx then
            (dcl foo: bool; foo := assert([BoolToNat(accessManager.GetLoginState() = <Locked>)],
            else
            (dcl foo: bool; foo := assert([BoolToNat(accessManager.GetLoginState() = <Unlocked>)],
            );
traces
    TestLogin:
        let test in set {mk_(passwordText, true),
                         mk_("OtherPassword", false),
                         mk_("ThirdPassword", false)} in
        assert([BoolToNat(accessManager.Login(test.#1, device))],[BoolToNat(test.#2)])

    TestLoginTwice:
        (assert([BoolToNat(accessManager.Login(passwordText, device))],[BoolToNat(true)]);
        assert([BoolToNat(accessManager.Login(passwordText, device))],[BoolToNat(false)]));

    TestLogout:
        (assert([BoolToNat(accessManager.Logout())],[BoolToNat(false)]);
         assert([BoolToNat(accessManager.Login(passwordText, device))],[BoolToNat(true)]);
         assert([BoolToNat(accessManager.Logout())],[BoolToNat(true)]);
        )

    TestAuthorizedDevices:
        let dev in set {
            mk_(device, true),
            mk_(new Device(), false)
```

```
        } in
         (assert([BoolToNat(accessManager.Login(passwordText, dev.#1))],[BoolToNat(dev.#2)]);
          accessManager.Logout();
          accessManager.AuthorizeDevice(dev.#1);
          assert([BoolToNat(accessManager.Login(passwordText, dev.#1))],[BoolToNat(true)]);
          accessManager.Logout();
          assert([BoolToNat(accessManager.Login(passwordText, device))],[BoolToNat(true)]);)

    TestLogoutOnTimeout:
        let intervals in set {
            mk_([10,20,270], 3),
            mk_([310, 301,1], 0),
            mk_([43, 257, 1, 41], 2),
            mk_([1, 2, 3, 4], nil)
        } in
            (accessManager.Login(passwordText, device);
             testTimeout(intervals.#1, intervals.#2))

    TestDoActivity:
        let intervals in set {
            mk_([10,20,270], nil),
            mk_([310, 301,1], 0),
            mk_([43, 257, 1, 41], nil),
            mk_([1, 2, 3, 4], nil),
            mk_([12, 23, 300, 4], 3)
        } in
            (accessManager.Login(passwordText, device);
             testDoActivity(intervals.#1, intervals.#2))

    TestGetUsername:
        assert([BoolToNat(accessManager.GetUserName() = "TestUser")], [BoolToNat(true)])

    TestGetDatabase:
        (assert([BoolToNat(accessManager.GetDatabase() = nil)], [BoolToNat(true)]);
         accessManager.Login(passwordText, device);
         assert([BoolToNat(accessManager.GetDatabase() <> nil)], [BoolToNat(true)])
        );
end TestAccessManager
```

Function or operation	Line	Coverage	Calls
TestAccessManager	7	100%	34
testTimeout	16	100%	4
testDoActivity	26	100%	5
TestAccessManager.vdmpp		100%	43

Table 2.9: Coverage statistics for `TestAccessManager.vdmpp`

Appendix 10: Coverage for "TestDatabase.vdmpp"

```
class TestDatabase is subclass of TEST
instance variables
    database: Database := new Database();
operations
    TestDatabase: () ==> TestDatabase
    TestDatabase() == (
            database.AddPassword(new Password("GamerTag", "YouShallNotPassword1", "https://store.s
            database.AddPassword(new Password("BuisnessAccount", "buisness123", "https://bank.com"
            database.AddPassword(new Password("testmail@mail.com", "test123", "https://mail.com"))
            database.AddPassword(new Password("PersonalAccount", "birthday", "https://bank.com"))
        )
traces
    TestLockUnlock:
        (
            assert([BoolToNat(database.GetLockState() = <Unlocked>)], [BoolToNat(true)]);
            database.Lock(new EncryptionKey(1, new Password("SomePassword")));
            assert([BoolToNat(database.GetLockState() = <Locked>)], [BoolToNat(true)]);
            database.Unlock(new EncryptionKey(1, new Password("SomePassword")));
            assert([BoolToNat(database.GetLockState() = <Unlocked>)], [BoolToNat(true)]);
        )

    TestGetDatabaseAndAddPassword:
        (
            assert([len database.GetAllPasswords()], [4]);
            database.AddPassword(new Password("PersonalAccount", "birthday", "https://bank.com"));
            assert([len database.GetAllPasswords()], [5]);
            database.AddPassword(new Password("SomeAccount", "password", "app:string:to:match"));
            assert([len database.GetAllPasswords()], [6]);
            database.AddPassword(new Password("passWithoutMatchString"));
            assert([len database.GetAllPasswords()], [6])
        )

    TestGetDatabaseAndRemovePassword:
        (
            assert([len database.GetAllPasswords()], [4]);
            database.RemovePassword(3);
            assert([len database.GetAllPasswords()], [3]);
            database.RemovePassword(4);
            assert([len database.GetAllPasswords()], [3]);
            database.RemovePassword(3);
            assert([len database.GetAllPasswords()], [2])
        )

    TestFindPassowrds:
        let data in set {
            mk_("https://mail.com", 1),
            mk_("https://bank.com", 2),
            mk_("https://store.steampowered.com", 1),
            mk_("https://pageNotFound.com", 0)
        } in
            assert([BoolToNat((len database.FindPasswords(data.#1)) = data.#2)], [BoolToNat(true)]
end TestDatabase
```

Function or operation	Line	Coverage	Calls
TestDatabase	4	100%	15
TestDatabase.vdmpp		100%	15

Table 2.10: Coverage statistics for `TestDatabase.vdmpp`

Appendix 11: Coverage for "TestPassword.vdmpp"

```
class TestPassword is subclass of TEST
instance variables
password: Password := new Password("testUser", "test123", "https://testurl.com")
traces
TestPasswordCreation:
    let pass in set {
        mk_(new Password("test123"), true, true),
        mk_(new Password("test123", "https://testurl.com"), false, true),
        mk_(new Password("testUser", "test123", "https://testurl.com"), false, false)
    } in
        assert([BoolToNat(pass.#1.GetMatchString() = nil),
                BoolToNat(pass.#1.GetUserName() = nil)],
                [BoolToNat(pass.#2), BoolToNat(pass.#3)]);
TestEncryptDecrypt:
    let keys in set {
        mk_(new EncryptionKey(1, new Password("test")), new EncryptionKey(1, new Password("test"))
        mk_(new EncryptionKey(1, new Password("test")), new EncryptionKey(1, new Password("other")
    } in
        (
         assert([BoolToNat(password.GetEncryptionState() = <PlainText>)], [BoolToNat(true)]);
         password.Encrypt(keys.#1);
         assert([BoolToNat(password.GetEncryptionState() = <Encrypted>)], [BoolToNat(true)]);
         password.Decrypt(keys.#2);
         assert([BoolToNat(password.GetEncryptionState() = <PlainText>)], [BoolToNat(keys.#3)]));
end TestPassword
```

Function or operation	Line	Coverage	Calls
TestPassword.vdmpp		100%	0

Table 2.11: Coverage statistics for `TestPassword.vdmpp`

Appendix 12: Coverage for "TestTime.vdmpp"

```
class TestTimeObject is subclass of TimeObject
-- Helper class for time tests --
instance variables
    currentTime: Seconds;
    expireTime: Seconds;
    timeExired: bool := false;

operations
    public TestTimeObject: Time * Seconds ==> TestTimeObject
        TestTimeObject(time, seconds) == (
            currentTime := time.GetCurrentTime();
            expireTime := currentTime + seconds;
            time.Register(self));

    public TimeTicked: Seconds ==> ()
    TimeTicked(curTime) ==
        (currentTime := curTime;
         if curTime >= expireTime then
            timeExired := true;);

    public GetExpired: () ==> bool
        GetExpired() == return timeExired;

end TestTimeObject

class TestTime is subclass of TEST
instance variables
    time: Time := Time`GetInstance();
traces
    TestTickTime:
        let secs1 in set {20,30,40,50,12,1,302} in
                let secs2 in set {20,30,40,50,12,1,302} in
                    -- Tick once --
                    (time.TickTime(secs1);
                    assert([time.GetCurrentTime()], [secs1]);
                    -- Tick twice --
                    ( time.TickTime(secs2);
                    assert([time.GetCurrentTime()], [secs1+secs2])));
    TestTimeObject:
        let testObjs in set {mk_(new TestTimeObject(time, 100), 40, 40, false, false),
                             mk_(new TestTimeObject(time, 100), 40, 80, false, true),
                             mk_(new TestTimeObject(time, 100), 100, 80, true, true),
                             mk_(new TestTimeObject(time, 42), 40, 2, false, true),
                             mk_(new TestTimeObject(time, 42), 44, 80, true, true),
                             mk_(new TestTimeObject(time, 42), 1, 40, false, false)} in
             (
               -- Register is normaly done in the constructer of TestTimeObject
               -- But it seams like the set for the test is created once, and that
               -- time is reset for each test therefore it is done here manually.
               time.Register(testObjs.#1);
               time.TickTime(testObjs.#2);
               assert([BoolToNat(testObjs.#1.GetExpired())], [BoolToNat(testObjs.#4)]);
               time.TickTime(testObjs.#3);
               assert([BoolToNat(testObjs.#1.GetExpired())], [BoolToNat(testObjs.#5)]);
             );
end TestTime
```

Function or operation	Line	Coverage	Calls
TestTimeObject	8	100%	12
TimeTicked	14	100%	12
GetExpired	20	100%	24
TestTime.vdmpp		100%	48

Table 2.12: Coverage statistics for `TestTime.vdmpp`

Appendix 13: Coverage for "TestUser.vdmpp"

```
class TestUser is subclass of TEST
types
    password_test = Password * bool;
instance variables
    userPassword: String := "Test123";
    user1: User := new User("TestUser1", userPassword);
    user2: User := new User("TestUser2", userPassword);
traces
    TestIsPasswordCorrect:
        let pswd in set {mk_(new Password("wrong"), false),
                         mk_(new Password("otherPass"), false),
                         mk_(new Password("TestUser1"), false),
                         mk_(new Password("Test123"), true)} --be
          --st user.IsCorrectPassword(pswd) in
              --user.GetEncryptionKey(pswd)
          in (assert([BoolToNat(user1.IsCorrectPassword(pswd.#1))], [BoolToNat(pswd.#2)]);
              assert([BoolToNat(user2.IsCorrectPassword(pswd.#1))], [BoolToNat(pswd.#2)]););
    TestGetUserName:
        let usr in set {mk_("TestUser2", false, true),
                         mk_("TestUser1", true, false),
                         mk_("Test123", false, false),
                         mk_("Test321", false, false)}
          in (assert([BoolToNat(user1.GetUserName() = usr.#1)],
                     [BoolToNat(usr.#2)]);
              assert([BoolToNat(user2.GetUserName() = usr.#1)],
                     [BoolToNat(usr.#3)]););
    TestGetEncryptionKey:
                let usr in set{user1, user2} in
                    usr.GetEncryptionKey(new Password(userPassword));


    TestChangePassword:
                let newPass in set {"1", "2", "3", "4"} in
                   ( assert([BoolToNat(user1.IsCorrectPassword(new Password(userPassword))),
                             BoolToNat(user1.IsCorrectPassword(new Password(newPass)))],
                             [BoolToNat(true),BoolToNat(false)]);
                    (user1.ChangePassword(newPass));
                    assert([BoolToNat(user1.IsCorrectPassword(new Password(userPassword))),
                             BoolToNat(user1.IsCorrectPassword(new Password(newPass)))],
                             [BoolToNat(false),BoolToNat(true)]););
end TestUser
```

Function or operation	Line	Coverage	Calls
TestUser.vdmpp		100%	0

Table 2.13: Coverage statistics for `TestUser.vdmpp`

Appendix 14: Coverage for "TestUserDatabase.vdmpp"

```
class TestUserDatabase is subclass of TEST
instance variables
    database: UserDatabase := new UserDatabase();
    device: Device_t := new Device();
operations
    TestUserDatabase: () ==> TestUserDatabase
    TestUserDatabase() ==
        (dcl foo: bool; foo := database.AddUser("User1", "password1", device);
        foo := database.AddUser("User2", "password2", device);
        foo := database.AddUser("User3", "password3", device);
        foo := database.AddUser("User4", "password4", device);
        foo := database.AddUser("User5", "password5", device));

traces

    TestFindUser:
        let toFind in set {
            mk_("User1", true),
            mk_("User2", true),
            mk_("User3", true),
            mk_("User4", true),
            mk_("User5", true),
            mk_("User6", false),
            mk_("User7", false)
        } in(assert([BoolToNat(database.FindUser(toFind.#1) <> nil)],[BoolToNat(toFind.#2)]))

    TestCreateUser:
        let toCreate in set {
            mk_("User6", "password6", true),
            mk_("User7", "password7", true),
            mk_("User8", "password8", true),
            mk_("User1", "password1", false),
            mk_("User2", "password2", false)
        } in
            assert([BoolToNat(database.AddUser(toCreate.#1,toCreate.#2,device))],[BoolToNat(toCrea

end TestUserDatabase
```

Function or operation	Line	Coverage	Calls
TestUserDatabase	5	100%	16
TestUserDatabase.vdmpp		100%	16

Table 2.14: Coverage statistics for `TestUserDatabase.vdmpp`

Appendix 15: Coverage for "Time.vdmpp"

```
class Time is subclass of GLOBAL
    instance variables
        currentTime: Seconds := 0;
        updateObjects: seq of TimeObject := [];
        static timeInstance : [Time] := nil;
    operations

        private Time: () ==> Time
            Time() == return self;

        static public GetInstance: () ==> Time
            GetInstance() == (if timeInstance = nil then
                timeInstance := new Time();
            return timeInstance);

        public TickTime: Seconds ==> ()
        TickTime(amout) == (currentTime := currentTime + amout;
            for all i in set inds updateObjects do
                updateObjects(i).TimeTicked(currentTime);
        );

        public Register: TimeObject ==> ()
        Register(obj) == updateObjects := updateObjects ^ [obj];

        public GetCurrentTime: () ==> Seconds
        GetCurrentTime() == return currentTime;
end Time
```

Function or operation	Line	Coverage	Calls
Time	7	100%	110
GetInstance	10	100%	215
TickTime	15	100%	145
Register	21	100%	139
GetCurrentTime	24	100%	208
Time.vdmpp		100%	817

Table 2.15: Coverage statistics for `Time.vdmpp`

Appendix 16: Coverage for "TimeObject.vdmpp"

```
class TimeObject is subclass of GLOBAL
operations
    public TimeTicked: Seconds ==> ()
    TimeTicked(currentTime) ==
        is not yet specified
end TimeObject
```

Function or operation	Line	Coverage	Calls
TimeObject.vdmpp		100%	0

Table 2.16: Coverage statistics for `TimeObject.vdmpp`

Appendix 17: Coverage for "User.vdmpp"

```
class User is subclass of GLOBAL
instance variables
    private userName: String;
    private password: Password_t;
    private userId: int;
    private internalEncryptionKey: EncryptionKey;
    private static idCounter: int := 0;
    private static allUsers: set of User := {};

inv forall user1 in set allUsers &
    forall user2 in set allUsers \ {user1} &
    user1.internalEncryptionKey <> user2.internalEncryptionKey and
    user1.userName <> user2.userName ;

operations
    public User: String * String ==> User
    User(userNme, passwd) ==
    (userName:=userNme;
    password:=new Password(passwd);
    userId := idCounter;
    internalEncryptionKey := new EncryptionKey(idCounter, new Password(passwd));
    idCounter := idCounter + 1;
    allUsers := allUsers union {self};
    )
    pre forall user in set allUsers & userNme <> user.userName
    post userId <> idCounter and allUsers~ psubset allUsers;

    public GetEncryptionKey: Password_t ==> EncryptionKey_t
        GetEncryptionKey(passwd) ==
        return new EncryptionKey(userId, passwd)
        pre password = passwd;

    public pure IsCorrectPassword: Password_t ==> bool
         IsCorrectPassword(pass) ==
            (if pass = password then return true; return false)
        post RESULT = false or pass = password;

    public pure GetUserName: () ==> String
        GetUserName() == return userName;

    public ChangePassword: String ==> ()
        ChangePassword(passwd) == (password:=new Password(passwd);
        internalEncryptionKey :=  GetEncryptionKey(new Password(passwd)));

end User
```

Function or operation	Line	Coverage	Calls
User	15	100%	165
GetEncryptionKey	27	100%	186
IsCorrectPassword	32	100%	32
GetUserName	37	100%	292
ChangePassword	40	100%	4
User.vdmpp		100%	679

Table 2.17: Coverage statistics for `User.vdmpp`

Appendix 18: Coverage for "UserDatabase.vdmpp"

```
class UserDatabase is subclass of GLOBAL
instance variables
    users: seq of AccessManager := [];
operations
    public FindUser: String ==> [AccessManager]
    FindUser(userName) ==
        (for all i in set inds users do
            if users(i).GetUserName() = userName then
                return users(i);
            io.print("Could not find user!\n");
            return nil;
        );

    public AddUser: String * String * Device_t ==> bool
    AddUser(userName, password, device) ==
        (for all i in set inds users do
            if users(i).GetUserName() = userName then
                (io.print("Could not add user, username already exits!\n"); return false);
        users := users ^ [new AccessManager(userName, password, device)];
        return true);

end UserDatabase
```

Function or operation	Line	Coverage	Calls
FindUser	4	100%	16
AddUser	13	100%	91
UserDatabase.vdmpp		100%	107

Table 2.18: Coverage statistics for `UserDatabase.vdmpp`

Appendix 19: Coverage for "World.vdmpp"

```
class World
instance variables
    public static env: [Environment]:= nil;
operations
    public World :()==>World
    World() ==
    env := new Environment("scenario.txt");

    public Run :()==>()
    Run() ==
    (env.Run(););
end World
```

Function or operation	Line	Coverage	Calls
World	4	100%	1
Run	8	100%	2
World.vdmpp		100%	3

Table 2.19: Coverage statistics for `World.vdmpp`