Project 7

Bharath Karumudi

August 31, 2019

Abstract

This project is to demonstrate the capabilities of implementing constructing and deconstructing HOL Terms using the tools and techniques - LaTeX, AcuTeX, emacs and ML.

Each chapter documents the given problems with a structure of:

1. Problem Statement
2. Relevant Code
3. Execution Transcripts
4. Explanation of results

Acknowledgments: Professor Marvine Hamner and Professor Shiu-Kai Chin who taught the Certified Security By Design.

Contents

Chapter 1

Executive Summary

All requirements for this project are statisfied specifically, and by using HOL proved the below theorems:

Chapter 2

Exercise 15.6.1

2.1 Problem Statement

In this we need to prove:

$\vdash \forall$ *keyAlice k text*. (deciphS *keyAlice* (Es *k* (SOME *text*)) = SOME "This is from Alice") $\iff$ (*k* = *keyAlice*) $\wedge$ (*text* = "This is from Alice")

$\vdash \forall$ *P message*. (deciphP (pubK *P*) *enMsg* = SOME *message*) $\iff$ (*enMsg* = Ea (privK *P*) (SOME *message*))

2.2 Proof of exercise15_6_1a_thm

2.2.1 Relevant Code

```
val exercise15_6_1a_thm =
TAC_PROOF(
([],
``! key enMsg message. (deciphS key enMsg = SOME message) <=> (enMsg = Es key (SOME message
ASM_REWRITE_TAC [deciphS_one_one]);

val _ = save_thm("exercise15_6_1a_thm",exercise15_6_1a_thm)
```

2.2.2 Execution Transcripts

1

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
> > > >

val exercise15_6_1a_thm =
    []
|- !(key :symKey) (enMsg :'a symMsg) (message :'a).
     (deciphS key enMsg = SOME message) <=>
     (enMsg = Es key (SOME message)):
   thm
>
```

2.2.3 Explanation of Results

The above results shows that the requirements are satisfied.

2.3 Proof of exercise15_6_1b_thm

2.3.1 Relevant Code

```
val exercise15_6_1b_thm =
TAC_PROOF(
([],
``! keyAlice k text.
(deciphS keyAlice (Es k (SOME text)) =
SOME "This is from Alice") <=>
(k = keyAlice) /\ (text = "This is from Alice")``),
ASM_REWRITE_TAC [deciphS_one_one] THEN
PROVE_TAC[]
);

val _ = save_thm("exercise15_6_1b_thm",exercise15_6_1b_thm)
```

2.3.2 Execution Transcripts

1

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
> > > >

> # # # # # # # # # Meson search level: ......................
val exercise15_6_1b_thm =
    []
|- !(keyAlice :symKey) (k :symKey) (text :string).
     (deciphS keyAlice (Es k (SOME text)) =
      SOME "This is from Alice") <=>
     (k = keyAlice) /\ (text = "This is from Alice"):
   thm
>
```

2.3.3 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 3

Exercise 15.6.2

3.1 Problem Statement

In this we need to prove:

$\vdash \forall key\ text.$ (`deciphP` (`pubK` *Alice*) (`Ea` *key* (`SOME` *text*)) = `SOME` "This is from Alice") $\iff$ (*key* = `privK` *Alice*) $\wedge$ (*text* = "This is from Alice")

$\vdash \forall key\ text.$ (`deciphP` (`pubK` *Alice*) (`Ea` *key* (`SOME` *text*)) = `SOME` "This is from Alice") $\iff$ (*key* = `privK` *Alice*) $\wedge$ (*text* = "This is from Alice")

3.2 Proof of exercise15_6_2a_thm

3.2.1 Relevant Code

```
val exercise15_6_2a_thm =
TAC_PROOF(
([],
``! P message.
(deciphP (pubK P) enMsg = SOME message)<=>
(enMsg = Ea (privK P) (SOME message))``),
PROVE_TAC[deciphP_one_one]
);

val _ = save_thm("exercise15_6_2a_thm",exercise15_6_2a_thm)
```

3.2.2 Execution Transcripts

1

```
------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------
> # # # # # # # <<HOL message: inventing new type variable names: 'a, 'b>>
Meson search level: ..........
val exercise15_6_2a_thm =
    []
|- !(P :'a) (message :'b).
     (deciphP (pubK P) (enMsg :('b, 'a) asymMsg) = SOME message) <=>
     (enMsg = Ea (privK P) (SOME message)):
   thm
>
```

3.2.3 Explanation of Results

The above results shows that the requirements are satisfied.

3.3 Proof of exercise15_6_2b_thm

3.3.1 Relevant Code

```
val exercise15_6_2b_thm =
TAC_PROOF(
([],
``! key text.
(deciphP (pubK Alice) (Ea key (SOME text)) =
SOME "This is from Alice") <=>
(key = privK Alice)      (text = "This is from Alice")``),
ASM_REWRITE_TAC[deciphP_one_one] THEN
ASM_REWRITE_TAC[option_CLAUSES]);

val _ =save_thm("exercise15_6_2b_thm", exercise15_6_2b_thm)
```

3.3.2 Execution Transcripts

1

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
> # # # # # # # # <<HOL message: inventing new type variable names: 'a>>
val exercise15_6_2b_thm =
    []
|- !(key :'a pKey) (text :string).
     (deciphP (pubK (Alice :'a)) (Ea key (SOME text)) =
      SOME "This is from Alice") <=>
     (key = privK Alice) /\ (text = "This is from Alice"):
   thm
>
```

3.3.3 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 4

Exercise 15.6.3

4.1 Problem Statement

In this we need to prove:

$\vdash \forall$ *signature*. `signVerify` (`pubK` *Alice*) *signature* (`SOME` "This is from Alice") $\iff$ (*signature* = `sign` (`privK` *Alice*) (`hash` (`SOME` "This is from Alice")))

4.1.1 Relevant Code

```
val exercise15_6_3_thm =
TAC_PROOF(
([],
``! signature.signVerify (pubK Alice) signature
(SOME "This is from Alice") <=>
(signature =
sign (privK Alice) (hash (SOME "This is from Alice")))``),
ASM_REWRITE_TAC[signVerify_one_one]);

val _ =save_thm("exercise15_6_3_thm",exercise15_6_3_thm)
```

4.1.2 Execution Transcripts

1

```
------------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
------------------------------------------------------------------------
> # # # # # # # <<HOL message: inventing new type variable names: 'a>>
val exercise15_6_3_thm =
    []
|- !(signature :(string digest, 'a) asymMsg).
     signVerify (pubK (Alice :'a)) signature
       (SOME "This is from Alice") <=>
     (signature = sign (privK Alice) (hash (SOME "This is from Alice"))):
   thm
>
```

4.1.3 Explanation of Results

The above results shows that the requirements are satisfied.

Chapter 5

Appendix A: cipherScript

The following code is from the file cipherScript.sml

```
(*---------------------------------------------------------*)
(* Cipher operations                                       *)
(* Created 3 May 2014: Shiu-Kai Chin                       *)
(* Replaced datatype contents with HOL built-in optionTheory *)
(*---------------------------------------------------------*)

(* Interactive mode
app load ["isainfRules","TypeBase","optionTheory"]

(* Disable Pretty-Printing *)
set_trace "Unicode" 0;
*)

structure cipherScript = struct

open HolKernel boolLib Parse bossLib
open TypeBase isainfRules optionTheory

(***********
* create a new theory
***********)
val _ = new_theory "cipher";

(************************
* THE DEFINITIONS START HERE
************************)

(*---------------------------------------------*)
(* Symmetric Encryption/Decryption             *)
(*---------------------------------------------*)

(*---------------------------------------------*)
(* Creating symmetric (secret) keys and        *)
(* encrypted messages with symmetric keys.     *)
(*---------------------------------------------*)

val _ = Datatype `symKey = sym num`;
val _ = Datatype `symMsg = Es symKey ('message option)`;

val symKey_one_one = TypeBase.one_one_of ``:symKey``
val _ = save_thm("symKey_one_one",symKey_one_one)
```

```
val symMsg_one_one = TypeBase.one_one_of ``:'message symMsg``
val _ = save_thm("symMsg_one_one",symMsg_one_one)

(*********************************************)
(* Deciphering with symmetric keys           *)
(* Define with pattern matching. If the key  *)
(* matches then we can retrieve the plain text. *)
(* No definition is offered for the result if  *)
(* the key in the message doesn't match the key *)
(* that is supplied.                         *)
(*********************************************)
val deciphS_def =
    Define
    `(deciphS (k1:symKey) (Es k2 (SOME (x:'message))) =
     if (k1 = k2) then (SOME x) else (NONE:'message option)) /\
     (deciphS (k1:symKey) (Es k2 (NONE:'message option)) = NONE)`;

(*********************************************)
(* Creating asymmetric public and private keys. *)
(* As these keys are created using a common  *)
(* parameter, we will model this parameter with *)
(* the principal with whom the keys are      *)
(* associated.                               *)
(*********************************************)
val _ = Datatype `pKey = pubK 'princ | privK 'princ`;
val _ = Datatype `asymMsg = Ea ('princ pKey) ('message option)`;

val pKey_one_one = TypeBase.one_one_of ``:'princ pKey``
val _ = save_thm("pKey_one_one",pKey_one_one)

val pKey_distinct_clauses = distinct_clauses ``:'princ pKey``
val _ = save_thm("pKey_distinct_clauses",pKey_distinct_clauses)

val asymMsg_one_one = TypeBase.one_one_of ``:('princ,'message) asymMsg``
val _ = save_thm("asymMsg_one_one",asymMsg_one_one)

(*********************************************)
(* Deciphering with asymmetric keys          *)
(* Define with pattern matching. If the      *)
(* corresponding keys match then the text is *)
(* recovered. In all other cases NONE is     *)
(* returned.                                 *)
(*********************************************)
val deciphP_def =
    Define
    `(deciphP (key:'princ pKey) (Ea (privK (P:'princ)) (SOME (x:'message))) =
      if ((key:'princ pKey) = (pubK (P:'princ))) then (SOME (x:'message)) else (NONE:'messa
     /\
     (deciphP (key:'princ pKey) (Ea (pubK (P:'princ)) (SOME (x:'message))) =
      if ((key:'princ pKey) = (privK (P:'princ))) then (SOME (x:'message)) else (NONE:'mes
     /\
     (deciphP (k1:'princ pKey)(Ea (k2:'princ pKey) (NONE:'message option)) = (NONE:'message
```

```
(*********************************************)
(* Message digests are cryptographic hashes of  *)
(* messages.                                    *)
(*********************************************)
val _ = Datatype `digest = hash  ('message option)`;
val digest_one_one = TypeBase.one_one_of ``:'message digest``
val _ = save_thm("digest_one_one",digest_one_one);

(*********************************************)
(* Signatures are digests encrypted by the      *)
(* private key of the sender.                   *)
(*********************************************)
val sign_def =
    Define
    `sign (pubKey:'princ pKey) (dgst:'message digest) = Ea pubKey (SOME dgst)`;

(*********************************************)
(* Integrity checking of messages is checking   *)
(* the hash of the received message equals the  *)
(* signature decrypted with the sender's public *)
(* key.                                         *)
(*********************************************)
val signVerify_def =
    Define
    `signVerify (pubKey:'princ pKey)(signature:('message digest,'princ)asymMsg)(msgContents
    ((SOME (hash msgContents)) = (deciphP pubKey signature))`;

(*********************************************)
(* A theorem to make sure that our integrity    *)
(* checking function works with the way we      *)
(* create digital signatures.                   *)
(*********************************************)
val signVerifyOK =
    save_thm
    ("signVerifyOK",
    TAC_PROOF(
    ([],``!(P:'princ)(msg:'message).signVerify (pubK P) (sign (privK P) (hash (SOME msg)))(
    (REWRITE_TAC [signVerify_def, sign_def, deciphP_def])));

val th1 =
    TAC_PROOF(
    ([], ``!P text.((deciphP (pubK P)(Ea (privK P) (SOME text)) = (SOME text)) /\
    (deciphP (privK P)(Ea (pubK P) (SOME text)) = (SOME text)))``),
    (REPEAT STRIP_TAC THEN
     REWRITE_TAC [deciphP_def]));

val option_distinct =
    save_thm("option_distinct",TypeBase.distinct_of (Type `:'a option`));

val th2a =
TAC_PROOF(
([], ``!k P text.
         (deciphP k (Ea (privK P) (SOME text)) = (SOME text)) ==> (k = pubK P)``),
```

```
   (REPEAT GEN_TAC THEN
    REWRITE_TAC [deciphP_def] THEN
    BOOL_CASES_TAC ``k = (pubK P)`` THEN
    REWRITE_TAC [option_distinct]));

val th2b =
TAC_PROOF(
([],
``!k P text.
    (k = pubK P) ==> (deciphP k (Ea (privK P) (SOME text)) = (SOME text))``),
PROVE_TAC[deciphP_def])

val th2 =
TAC_PROOF(
([], ``!k P text.
          (deciphP k (Ea (privK P) (SOME text)) = (SOME text)) = (k = pubK P)``),
PROVE_TAC[th2a,th2b])

val th3a = TAC_PROOF(
 ([], ``!k P text.
          (deciphP k (Ea (pubK P) (SOME text)) = (SOME text)) ==> (k = privK P)``),
  (REPEAT GEN_TAC THEN
   REWRITE_TAC [deciphP_def] THEN
   BOOL_CASES_TAC ``k = (privK P)`` THEN
   REWRITE_TAC [option_distinct]));

val th3b = TAC_PROOF(
 ([],
``!k P text.
  (k = privK P) ==> (deciphP k (Ea (pubK P) (SOME text)) = (SOME text))``),
PROVE_TAC[deciphP_def])

val th3 = TAC_PROOF(
 ([],
``!k P text.
  (deciphP k (Ea (pubK P) (SOME text)) = (SOME text)) = (k = privK P)``),
PROVE_TAC[th3a,th3b])

val th4 =
GEN_ALL
(REWRITE_RULE[pKey_distinct_clauses]
(ISPECL
 [``pubK (P1:'b)``,``P2:'b``]
 (GENL [``key:'princ pKey``,``P:'princ``](CONJUNCT2 (SPEC_ALL deciphP_def)))))

val th5 =
GEN_ALL
(REWRITE_RULE[pKey_distinct_clauses]
(ISPECL
 [``privK (P1:'b)``,``P2:'b``]
 (GENL [``key:'princ pKey``,``P:'princ``](CONJUNCT1 (SPEC_ALL deciphP_def)))))

val deciphP_clauses =
```

```
    save_thm("deciphP_clauses",LIST_CONJ [th1,th2,th3,th4,th5]);

val th1 =
    TAC_PROOF(
    ([], ``!k text.(deciphS k (Es k (SOME text)) = (SOME text))``),
    (REPEAT STRIP_TAC THEN
     REWRITE_TAC [deciphS_def]));

val th2a =
TAC_PROOF(
 ([], ``!(k1:symKey) (k2:symKey) text.
         (deciphS k1 (Es k2 (SOME text)) = (SOME text)) ==> (k1 = k2)``),
  (REPEAT GEN_TAC THEN
   REWRITE_TAC [deciphS_def] THEN
   BOOL_CASES_TAC ``k1:symKey = k2:symKey`` THEN
   REWRITE_TAC [option_distinct]));

val th2b =
TAC_PROOF(
([], ``!(k1:symKey) (k2:symKey) text.
        (k1 = k2)  ==> (deciphS k1 (Es k2 (SOME text)) = (SOME text))``),
PROVE_TAC[deciphS_def])

val th2 =
TAC_PROOF(
([], ``!(k1:symKey) (k2:symKey) text.
         (deciphS k1 (Es k2 (SOME text)) = (SOME text)) = (k1 = k2)``),
PROVE_TAC[th2a,th2b])

val th3 =
TAC_PROOF(
([],``!(k1:symKey)(k2:symKey) text.
      (deciphS k1 (Es k2 (SOME text)) = NONE) = (k1 <> k2)``),
REPEAT STRIP_TAC THEN
Cases_on`k1 = k2` THEN
EQ_TAC THEN
ASM_REWRITE_TAC[deciphS_def,NOT_SOME_NONE])

val th4 =
TAC_PROOF(
([],``!(k1:symKey)(k2:symKey).
       deciphS k1 (Es k2 NONE) = NONE``),
REWRITE_TAC[deciphS_def])

val deciphS_clauses =
    save_thm("deciphS_clauses",LIST_CONJ [th1,th2,th3,th4]);

val option_one_one = TypeBase.one_one_of``:'a option``
val _ = save_thm("option_one_one",option_one_one)

val option_distinct_clauses = CONJ (distinct_of``:'a option``) (GSYM(distinct_of``:'a optio

val signlemma1 =
```

```
GEN_ALL(TAC_PROOF(
([],
``(sign pubKey1 (hash m1) = sign pubKey2 (hash m2)) ==> ((pubKey1 = pubKey2) /\ (m1 = m2))
REWRITE_TAC[sign_def,pKey_one_one,option_one_one,asymMsg_one_one,digest_one_one]))

val signlemma2 =
GEN_ALL(TAC_PROOF(
([],
``((pubKey1 = pubKey2) /\ (m1 = m2)) ==> (sign pubKey1 (hash m1) = sign pubKey2 (hash m2))
PROVE_TAC[]))

val sign_one_one =
TAC_PROOF(
([],
``!pubKey1 pubKey2 m1 m2.
  (sign pubKey1 (hash m1) = sign pubKey2 (hash m2)) = ((pubKey1 = pubKey2) /\ (m1 = m2))``)
PROVE_TAC[signlemma1,signlemma2])

val _ = save_thm("sign_one_one",sign_one_one)

val lemma1a =
GEN_ALL(TAC_PROOF(
([],``(deciphS k1 (Es k2 (SOME text2)) = SOME text1) ==> ((k1 = k2) /\ (text1 = text2))``)
(REWRITE_TAC [deciphS_def] THEN
COND_CASES_TAC THEN
REWRITE_TAC[option_distinct_clauses,option_one_one] THEN
PROVE_TAC[])))

val lemma1b =
TAC_PROOF(
([],
``((k1 = k2) /\ (text1 = text2)) ==> (deciphS k1 (Es k2 (SOME text2)) = SOME text1)``),
PROVE_TAC[deciphS_clauses])

val lemma1 =
TAC_PROOF(
([],``!k1 k2 text1 text2.(deciphS k1 (Es k2 (SOME text2)) = SOME text1) = ((k1 = k2) /\ (te
PROVE_TAC[lemma1a,lemma1b])

val lemma2 =
TAC_PROOF(
([],``!(enMsg:'message symMsg) text key.(deciphS key enMsg = (SOME (text:'message))) = (enM
Cases_on`enMsg`  THEN
REWRITE_TAC[deciphS_def,symMsg_one_one] THEN
REPEAT GEN_TAC THEN
EQ_TAC THEN
REPEAT(DISCH_THEN (fn th => ASSUME_TAC th THEN ONCE_REWRITE_TAC[th])) THEN
REWRITE_TAC[deciphS_clauses] THEN
UNDISCH_TAC
``deciphS (key :symKey) (Es (s :symKey) (o' :'message option)) =
        SOME (text :'message)`` THEN
Cases_on`o'` THEN
REWRITE_TAC[deciphS_def,option_CLAUSES] THEN
```

```
COND_CASES_TAC THEN
PROVE_TAC[option_CLAUSES])


val deciphS_one_one = CONJ lemma1 lemma2
val _ = save_thm("deciphS_one_one",deciphS_one_one)

val lemma1a =
GEN_ALL(TAC_PROOF(
([],``(deciphP (pubK P1)(Ea (privK P2) (SOME text2)) = SOME text1) ==> ((P1 = P2) /\ (text1
(REWRITE_TAC[deciphP_def] THEN
COND_CASES_TAC THEN
REWRITE_TAC[option_distinct_clauses,option_one_one] THEN
PROVE_TAC[pKey_one_one])))

val lemma1b =
TAC_PROOF(
([],
``!P1 P2 text1 text2.
  ((P1 = P2) /\ (text1 = text2)) ==> (deciphP (pubK P1)(Ea (privK P2) (SOME text2)) = SOME
PROVE_TAC[deciphP_def])

val lemma1 =
TAC_PROOF(
([],
``!P1 P2 text1 text2.
  (deciphP (pubK P1)(Ea (privK P2) (SOME text2)) = SOME text1) = ((P1 = P2) /\ (text1 = tex
PROVE_TAC[lemma1a,lemma1b])


val lemma2a =
GEN_ALL(TAC_PROOF(
([],``(deciphP (privK P1)(Ea (pubK P2) (SOME text2)) = SOME text1) ==> ((P1 = P2) /\ (text1
(REWRITE_TAC[deciphP_def] THEN
COND_CASES_TAC THEN
REWRITE_TAC[option_distinct_clauses,option_one_one] THEN
PROVE_TAC[pKey_one_one])))

val lemma2b =
TAC_PROOF(
([],
``!P1 P2 text1 text2.
  ((P1 = P2) /\ (text1 = text2)) ==> (deciphP (privK P1)(Ea (pubK P2) (SOME text2)) = SOME
PROVE_TAC[deciphP_def])

val lemma2 =
TAC_PROOF(
([],
``!P1 P2 text1 text2.
  (deciphP (privK P1)(Ea (pubK P2) (SOME text2)) = SOME text1) = ((P1 = P2) /\ (text1 = tex
PROVE_TAC[lemma2a,lemma2b])
```

```
val lemma3a =
TAC_PROOF(
([],``!(p:'b pKey)(c:'a option).(deciphP(pubK (P:'b))(Ea p c) = SOME (msg:'a)) ==> (p = pri
Cases THEN
Cases THEN
REWRITE_TAC[deciphP_def,pKey_distinct_clauses,deciphP_clauses,option_distinct_clauses,lemma
PROVE_TAC[COND_ID,option_distinct_clauses])

val lemma3b =
TAC_PROOF(
([],
``!(p:'b pKey)(c:'a option).
   ((p = privK P) /\ (c = SOME msg)) ==> (deciphP(pubK (P:'b))(Ea p c) = SOME (msg:'a))``)
PROVE_TAC[deciphP_def])

val lemma3 =
TAC_PROOF(
([],
``!(p:'b pKey)(c:'a option) P msg.
(deciphP(pubK (P:'b))(Ea p c) = SOME (msg:'a)) = (p = privK P) /\ (c = SOME msg)``),
PROVE_TAC[lemma3a,lemma3b])

val lemma4a =
TAC_PROOF(
([],``!(enMsg:('a,'b)asymMsg).(deciphP(pubK (P:'b))enMsg = SOME (msg:'a)) ==> (enMsg = Ea
Cases THEN
REWRITE_TAC[asymMsg_one_one,lemma3])

val lemma4b =
TAC_PROOF(
([],``!(enMsg:('a,'b)asymMsg).(enMsg = Ea (privK P) (SOME msg)) ==> (deciphP(pubK (P:'b))en
PROVE_TAC[deciphP_def])

val lemma4 =
TAC_PROOF(
([],
``!(enMsg:('a,'b)asymMsg) P msg.
   (deciphP(pubK (P:'b))enMsg = SOME (msg:'a)) = (enMsg = Ea (privK P) (SOME msg))``),
PROVE_TAC[lemma4a,lemma4b])

val lemma5a =
TAC_PROOF(
([],``!(p:'b pKey)(c:'a option).(deciphP(privK (P:'b))(Ea p c) = SOME (msg:'a)) ==> (p = pu
Cases THEN
Cases THEN
REWRITE_TAC[deciphP_def,pKey_distinct_clauses,deciphP_clauses,option_distinct_clauses,lemma
PROVE_TAC [COND_ID,option_distinct_clauses])

val lemma5b =
TAC_PROOF(
([],
``!(p:'b pKey)(c:'a option).
  ((p = pubK P) /\ (c = SOME msg)) ==> (deciphP(privK (P:'b))(Ea p c) = SOME (msg:'a))``),
```

```
PROVE_TAC [deciphP_clauses])

val lemma5 =
TAC_PROOF(
([],
``!(p:'b pKey)(c:'a option) P msg.
  (deciphP(privK (P:'b))(Ea p c) = SOME (msg:'a)) = (p = pubK P) /\ (c = SOME msg)``),
PROVE_TAC [lemma5a, lemma5b])

val lemma6a =
TAC_PROOF(
([],``!(enMsg:('a,'b)asymMsg) P msg.(deciphP(privK (P:'b))enMsg = SOME (msg:'a)) ==> (enMs
Cases THEN
REWRITE_TAC[asymMsg_one_one,lemma5])

val lemma6b =
TAC_PROOF(
([],
``!(enMsg:('a,'b)asymMsg) P msg.
   (enMsg = Ea (pubK P) (SOME msg)) ==> (deciphP(privK (P:'b))enMsg = SOME (msg:'a))``),
PROVE_TAC[deciphP_def])

val lemma6 =
TAC_PROOF(
([],
``!(enMsg:('a,'b)asymMsg) P msg.
   (deciphP(privK (P:'b))enMsg = SOME (msg:'a)) = (enMsg = Ea (pubK P) (SOME msg))``),
PROVE_TAC[lemma6a,lemma6b])

val deciphP_one_one = LIST_CONJ [lemma1,lemma2,lemma3,lemma4,lemma5,lemma6]

val _ = save_thm("deciphP_one_one",deciphP_one_one)

val lemma1a =
TAC_PROOF(
([],``!P m1 m2.signVerify (pubK (P:'a)) (Ea (privK P) (SOME (hash (SOME (m1 :'b)))))
      (SOME (m2:'b)) ==> (m1 = m2)`` ),
PROVE_TAC[signVerify_def,deciphP_def,option_one_one,digest_one_one])

val lemma1b =
TAC_PROOF(
([],``!(P:'a) (m1:'b) (m2:'b).
      (m1 = m2) ==>
      signVerify
      (pubK (P:'a)) (Ea (privK P) (SOME (hash (SOME (m1 :'b)))))
      (SOME (m2:'b))`` ),
PROVE_TAC[signVerify_def,deciphP_def])

val lemma1 =
TAC_PROOF
(
 ([],``!P m1 m2.signVerify (pubK (P:'a)) (Ea (privK P) (SOME (hash (SOME (m1 :'b)))))
      (SOME (m2:'b)) = (m1 = m2)`` ),
```

```
PROVE_TAC[lemma1a,lemma1b])

(* Start here *)
val lemma2 =
TAC_PROOF(
([],``!signature P text.signVerify (pubK (P:'princ)) signature (SOME (text:'message)) = (si
let val [_,_,lemma3,_,_,_] = CONJUNCTS deciphP_one_one
in
Cases_on `signature` THEN
REWRITE_TAC[signVerify_def] THEN
REWRITE_TAC[sign_def] THEN
REWRITE_TAC[asymMsg_one_one] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
DISCH_TAC THEN
ASM_REWRITE_TAC[deciphP_clauses] THEN
PAT_ASSUM``x`` (fn th => ASSUME_TAC(SYM th)) THEN
IMP_RES_TAC lemma3 THEN
(* The ASM_REWRITE_TAC appears to go on forever *)
(* PROVE_TAC [] *)
PROVE_TAC[]
end)

val lemma3a =
GEN_ALL(TAC_PROOF(
([],``signVerify (pubK P1) (sign (privK P2) (hash (SOME text2)))(SOME text1) ==> ((P1 = P2)
(REWRITE_TAC[signVerify_def,sign_def] THEN
DISCH_TAC THEN
PAT_ASSUM``a = b``(fn th => ASSUME_TAC (SYM th)) THEN
IMP_RES_TAC deciphP_one_one THEN
PAT_ASSUM``hash a = hash b``(fn th => ASSUME_TAC (REWRITE_RULE[digest_one_one,option_one_on
ASM_REWRITE_TAC[])))

val lemma3b =
GEN_ALL(TAC_PROOF(
([],
``((P1 = P2) /\ (text1 = text2)) ==> signVerify (pubK P1) (sign (privK P2) (hash (SOME text
PROVE_TAC[signVerifyOK]))

val lemma3 =
GEN_ALL(TAC_PROOF(
([],
``signVerify (pubK P1) (sign (privK P2) (hash (SOME text2)))(SOME text1) = ((P1 = P2) /\ (
PROVE_TAC[lemma3a,lemma3b]))

val signVerify_one_one = LIST_CONJ [lemma1,lemma2,lemma3]
val _ = save_thm("signVerify_one_one",signVerify_one_one)

 (* ==== start here ====
==== end here ==== *)

(*****************************)
(* Print and export the theory *)
```

```
(*****************************)
val _ = print_theory "-";

val _ = export_theory();

end;
```

Chapter 6

Appendix B: cryptoExericisesScript

The following code is from the file cryptoExercisesScript.sml

```
(*****************************************************************************)
(* Exercise Chapter 15                                                       *)
(* Author: Bharath Karumudi                                                  *)
(* Date: Aug 27 2019                                                         *)
(*****************************************************************************)

structure cryptoExercisesScript = struct

open HolKernel boolLib Parse bossLib
open TypeBase isainfRules optionTheory cipherTheory stringTheory

val _ = new_theory "cryptoExercises";

(*****************************************************************************)
(* Exercise 15.6.1A                                                          *)
(* `     key enMsg message.
*)
(*  deciphS key enMsg = SOME message)          (enMsg = Es key (SOME message))
*)
(*****************************************************************************)

val exercise15_6_1a_thm =
TAC_PROOF(
([],
``! key enMsg message. (deciphS key enMsg = SOME message) <=> (enMsg = Es key (SOME message
ASM_REWRITE_TAC [deciphS_one_one]);

val _ = save_thm("exercise15_6_1a_thm",exercise15_6_1a_thm)

(*****************************************************************************)
(* Exercise 15.6.1B                                                          *)
(* `     keyAlice k text.
*)
(*    (deciphS keyAlice (Es k (SOME text)) =                                 *)
(*    SOME  This is from Alice  )
*)
(*    (k = keyAlice)     (text =  This is from Alice  )
*)
(*****************************************************************************)

val exercise15_6_1b_thm =
TAC_PROOF(
```

```
([],
``! keyAlice k text.
(deciphS keyAlice (Es k (SOME text)) =
SOME "This is from Alice") <=>
(k = keyAlice) /\ (text = "This is from Alice")``),
ASM_REWRITE_TAC [deciphS_one_one] THEN
PROVE_TAC[]
);

val _ = save_thm("exercise15_6_1b_thm",exercise15_6_1b_thm)

(*****************************************************************************)
(* Exercise 15.6.2A                                                          *)
(*   `     P message.
*)
(*    (deciphP (pubK P) enMsg = SOME message)
*)
(*    (enMsg = Ea (privK P) (SOME message))                                   *)
(*****************************************************************************)

val exercise15_6_2a_thm =
TAC_PROOF(
([],
``! P message.
(deciphP (pubK P) enMsg = SOME message)<=>
(enMsg = Ea (privK P) (SOME message))``),
PROVE_TAC[deciphP_one_one]
);

val _ = save_thm("exercise15_6_2a_thm",exercise15_6_2a_thm)


(*****************************************************************************)
(* Exercise 15.6.2B                                                          *)
(*   `     key text.
*)
(*    (deciphP (pubK Alice) (Ea key (SOME text)) =                           *)
(*    SOME  This is from Alice  )
*)
(*    (key = privK Alice)     (text =  This is from Alice  )
*)
(*****************************************************************************)

val exercise15_6_2b_thm =
TAC_PROOF(
([],
``! key text.
(deciphP (pubK Alice) (Ea key (SOME text)) =
SOME "This is from Alice") <=>
(key = privK Alice)     (text = "This is from Alice")``),
ASM_REWRITE_TAC[deciphP_one_one] THEN
ASM_REWRITE_TAC[option_CLAUSES]);
```

```
val _ =save_thm("exercise15_6_2b_thm", exercise15_6_2b_thm)

(*************************************************************************)
(* Exercise 15.6.3                                                       *)
(*   `     signature.
*)
(*    signVerify (pubK Alice) signature                                  *)
(*    (SOME  This is from Alice )
*)
(*    (signature =                                                       *)
(*    sign (privK Alice) (hash (SOME  This is from Alice )))
*)
(*************************************************************************)

val exercise15_6_3_thm =
TAC_PROOF(
([],
``! signature.signVerify (pubK Alice) signature
(SOME "This is from Alice") <=>
(signature =
sign (privK Alice) (hash (SOME "This is from Alice")))``),
ASM_REWRITE_TAC[signVerify_one_one]);

val _ =save_thm("exercise15_6_3_thm",exercise15_6_3_thm)



(*************************************************************************)
(* Exporting Theory                                                      *)
(*************************************************************************)

val _ = print_theory "-";
val _ = export_theory();

end (* structure *)
```