Project: DRA FRONT END FILTER PROJECT

Title: Phase II Proof Strategy

Ref: DS/FMU/FEF/034 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
R. D. Arthan	WIN01		

Authorisation for Issue:

Name	*Function*	*Signature*	*Date*
R.B. Jones	HAT Manager		

Abstract: This document discusses proof opportunities and proof strategies for phase II of the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0 DOCUMENT CONTROL **2**
0.1 Contents List . . . 2
0.2 Document Cross References . . . 2
0.3 Changes History . . . 3
0.4 Changes Forecast . . . 3

1 GENERAL **4**
1.1 Scope . . . 4
1.2 Introduction . . . 4

2 PRELIMINARIES **5**

3 SYSTEM CONSTRUCTION **5**

4 ARCHITECTURAL MODEL **5**

5 EXECUTION MODEL **8**

6 TABLE COMPUTATIONS **9**

7 CLOSING DOWN **12**

8 THE THEORY fef034 **13**
8.1 Parents . . . 13
8.2 Children . . . 13
8.3 Constants . . . 13
8.4 Definitions . . . 13
8.5 Theorems . . . 14

9 INDEX **16**

0.2 Document Cross References

[1] DS/FMU/FEF/003. *Formal Security Policy.* G.M. Prout, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/007. *Proof Strategy.* G.M. Prout, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/021. *Specification of TSQL.* G.M. Prout, ICL Secure Systems, WIN01.

[4] DS/FMU/FEF/022. *SWORD Front End Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

[5] DS/FMU/FEF/024. *A HOL Specification of the SWORD Output Filter.* G.M. Prout, ICL Secure Systems, WIN01.

[6] DS/FMU/FEF/025. *Representation of an SSQL State as a TSQL State.* G.M. Prout, ICL Secure Systems, WIN01.

[7] DS/FMU/FEF/026. *Critical Requirements on the SWORD Query Transformations.* R.D. Arthan, ICL Secure Systems, WIN01.

[8] DS/FMU/FEF/028. *Specification of Query Transformations in HOL (I).* R.D. Arthan, ICL Secure Systems, WIN01.

[9] DS/FMU/FEF/029. *Specification of Query Transformations in HOL (II).* R.D. Arthan, ICL Secure Systems, WIN01.

[10] DS/FMU/FEF/032. *Table Computations for SWORD.* R.D. Arthan, ICL Secure Systems, WIN01.

[11] *The SWORD Front End Filter Specification.* Simon Wiseman, DRA, 21st January 1993.

[12] *SSQL Transformations.* Simon Wiseman, DRA, 14th January 1993.

0.3 Changes History

Issue 1.1 (7 September 1993) First draft.

Issue 1.5 (10 November 1993) Typos corrected.

Issue *Revision* : *2.1* **(5 June 2016)** Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document identifies various opportunities for proofs in phase II of the Front End Filter project and comments upon their significance. It also contains discussion of and formal definitions relating to the problem of finding the proofs.

1.2 Introduction

The specifications for phase II of the Front End Filter project are somewhat more complicated in structure than those of phase I. This is partly because of the greater complexity of the informal specifications and indeed of the system they model. It is also because the separation of security concerns from other concerns is less straightforward for the Front End implementation than it is for the SSQL semantics.

The three main documents in the phase II specifications in which the issues which are felt to be practically amenable to formal reasoning are separated out in stages are [4, 7, 10]. These are concerned with an *Architectural Model*, a *TSQL Execution Model* and the *Table Computations*, respectively.

The specifications in these documents draw on supporting material from the phase I specifications and from the documents [3, 5, 6, 8, 9] as necessary. Each of [4, 7, 10] identifies critical properties for the stage it is concerned with, and each of these provides an opportunity for formal proof work.

It is the intention of the present document to give a brief overview of the three stages of the discussion and to discuss in more detail the strategy for carrying out partial proofs of selected critical properties. As a result of preliminary correspondence and discussion with DRA, it is intended to concentrate effort on the security properties which relate most directly with the specifications of the table computations produced by the Front End Transformations.

This document itself gives some partial proofs demonstrating how the overall security properties of the system may be reduced to properties of its subsystems and how the material of [7, 10] recasts aspects of the critical properties on one of the subsystems into a more explicit and tractable form. These partial proofs are presented as ProofPower proofs of theorems whose conclusions are, in this document, always the overall security property for the SWORD system. The theorems have assumptions which are best thoughts of as lemmas in a putative security proof which have not yet been proved. Here, for example, the first partial proof shows that the system is secure provided its top-level subsystems possess certain security properties and that the architectural construction of the system from its subsystems has a certain property. Each partial proof step replaces one or more of these assumptions from an earlier theorem by zero or more new assumptions, the aim being that any new assumptions should be more tractable or more plausible, than the assumptions they replace. Here, in the final partial proof, the subsystem security property relating to the *SELECT* query has been replaced by three new assumptions which express the desired security property in a rather more explicit fashion.

2 PRELIMINARIES

The following ProofPower instructions set up the new theory *fef034*, which is used to hold definitions relating to the properties to be proved.

SML

```
open_theory "fef032";
(force_delete_theory "fef034" handle _ => ());
new_theory"fef034";
set_pc"hol";
```

3 SYSTEM CONSTRUCTION

The present document provides a convenient point to pull together the constructions of the other phase II specifications to give the formal definition of the behavioural model of the SWORD system whose security properties we wish to investigate by formal proof:

HOL Constant

FE_SWORD_SYSTEM : (*Query*, *ANSWER*) *BEHAVIOURS*

FE_SWORD_SYSTEM = *FE_SWORD reprState TSQLtf STP outputFilter*

Thus the model is obtained by applying the architectural construction function *FE_SWORD* of [4] to the the representation function and subsystems defined in [4, 6, 3, 5].

The overall security conjecture we are interested in is the assertion that this behavioural model satisfies the security policy defined in [1]. This may be captured in the following definition:

HOL Constant

FE_SWORD_SYSTEM_secure : *BOOL*

FE_SWORD_SYSTEM_secure ⇔ *FE_SWORD_SYSTEM* ∈ *secure*

4 ARCHITECTURAL MODEL

A formal model of the architecture of the Front End Implementation of SWORD is given in [4]. This architectural model defines how certain subsystems with specified interfaces are combined to produce a behavioural model of the SWORD system to which the formal security policy of [1] applies directly. [4] also defines critical properties on the subsystem and states a conjecture that the policy is satisfied if these hold.

The subsystems identified in [4] are as follows:

- The target DBMS, thought of as a processor of TSQL queries.

- An Output Filter, informally specified in [11], whose main role is to purge data from the output of the target DBMS which the client is not cleared to see.
- The SSQL Transformation Processor, informally specified in [12], whose main role is to map SSQL queries into sequences of one or two TSQL queries to be executed on the target DBMS.

The architectural construction is also parameterised by a representation function which is really an artefact of the modelling approach rather than a subsystem in its own right.

The main functional aspect described in [4] is the overall interconnection of the three subsystems. In particular, it defines the way the optional check queries produced by the SSQL Transformation Processor are used to prevent execution of queries which would otherwise cause illegal information flows which the Output Filter is not able to suppress.

[4] identifies five critical properties. Four of these, $subsys\_secureA \ldots subsys\_secureD$ constrain only the SSQL transformation processor and the target DBMS. The fifth $subsys\_secureE$ constrains all three subsystems. What is, in effect, the conjunction of the five properties is given the name $subsys\_secure$; it is conjectured that, if the three subsystems have this combined property, then the SWORD system constructed from them by the architectural model will satisfy the the security policy.

The architectural security conjecture taken from the informal discussion at the end of [4] may be recorded for future reference in a metalanguage variable[1]:

```
SML
|val SWORD_architecture_secure_conj = ⌜
|       ∀repr dbms stp filter•
|                       (dbms, stp, filter) ∈ subsys_secure repr
|               ⇒       FE_SWORD repr dbms stp filter ∈ secure
|⌝;
```

It is expected that the proof of this lemma would be quite similar to that of the unwinding result proved during phase I for the SSQL semantics (see [2]). Indeed, it is likely that the commonality between the two could be exploited to adapt parts of the phase I proofs for re-use here. These common parts would be those concerned with showing that appropriate critical properties on the transition function of the system entail the security of the behavioural model constructed from it. The more novel part of the proof would mainly be concerned with the function $mkTf_f$ of [4] which constructs the transition functions on the system and is largely concerned with looking after the optional check query and the check it makes.

The particular instance of this conjecture with which we are concerned is captured by the following definition.

[1] Because this conjecture is polymorphic, i.e. contains type variables, ProofPower does not allow us to define a boolean constant to capture it in the theory database (essentially because the truth value might depend on the instantiation of the type variables in the conjecture, which would render the definition inconsistent). Consequently, we use a term held in a metalanguage variables to record the conjecture rather than follow the style of [2].

HOL Constant

Architecture_Secure : *BOOL*

Architecture_Secure ⇔
(*TSQLtf*, *STP*, *outputFilter*) ∈ *subsys_secure reprState*
⇒
FE_SWORD_SYSTEM ∈ *secure*

On the basis of the definition of *subsys_secure* the hypothesis here might be expected to reduce to 5 subsidiary security properties as follows:

HOL Constant

Subsys_SecureA : *BOOL*;
Subsys_SecureB : *BOOL*;
Subsys_SecureC : *BOOL*;
Subsys_SecureD : *BOOL*;
Subsys_SecureE : *BOOL*

(*Subsys_SecureA* ⇔ (*TSQLtf*, *STP*) ∈ *subsys_secureA reprState*) ∧
(*Subsys_SecureB* ⇔ (*TSQLtf*, *STP*) ∈ *subsys_secureB reprState*) ∧
(*Subsys_SecureC* ⇔ (*TSQLtf*, *STP*) ∈ *subsys_secureC reprState*) ∧
(*Subsys_SecureD* ⇔ (*TSQLtf*, *STP*) ∈ *subsys_secureD reprState*) ∧
(*Subsys_SecureE* ⇔ (*TSQLtf*, *STP*, *outputFilter*) ∈ *subsys_secureE reprState*)

Of these five, it is part E which constrains the flow of information into the output of a transition; parts C and D constrain the information flows into the result state of a transition; parts A and B are, in essence, structural requirements ensuring that the subsystems fit together sensibly.

We may now give the following partial proof of the overall system security property, given the architectural security property and the security properties of the subsystems as assumptions.

SML

```
set_goal(
        [⌜Architecture_Secure⌝,
         ⌜Subsys_SecureA⌝,
         ⌜Subsys_SecureB⌝,
         ⌜Subsys_SecureC⌝,
         ⌜Subsys_SecureD⌝,
         ⌜Subsys_SecureE⌝],
                              ⌜FE_SWORD_SYSTEM_secure⌝);
```

SML

```
a(all_asm_ante_tac THEN rewrite_tac (map get_spec
        [⌜Architecture_Secure⌝, ⌜Subsys_SecureA⌝, ⌜Subsys_SecureB⌝,
        ⌜Subsys_SecureC⌝, ⌜Subsys_SecureD⌝, ⌜Subsys_SecureE⌝,
        ⌜subsys_secure⌝,
        ⌜FE_SWORD_SYSTEM_secure⌝]));
a(PC_T1 "sets_ext" REPEAT strip_tac);
val Theorem1 = save_pop_thm "Theorem1";
```

Only the SSQL *SELECT* query falls within the scope of phase II of the project. Consequently, further proof work concentrates on the conjecture *Subsys_SecureE*.

5 EXECUTION MODEL

The formal definition of the Output Filter, the SSQL Transformation Processor and the TSQL semantics for SWORD is given in the documents [8, 9, 5, 3]. This material is related to the architectural model by the execution model defined in [7]. This execution model serves both to bring these pieces together and to relate the specification of the transformations to a conceptually simpler semantic description of how SSQL queries are intended to be executed on the target database. This is achieved, in effect, by proposing an alternative means of constructing the TSQL database semantics. This construction, formalised as the function *EM* of [7], is parameterised by two things: a compiler mapping a TSQL query to the function of the database state it computes, and an update operation which computes the state transformation resulting from the application of a compiled query in a given state. The desired relationship between the compiler and update functions and the TSQL semantics as specified in [3] is captured in the relation *Correct_Compile* of [7].

The notion of compiler used in the Execution Model is also used in [7] to formalise the requirements on the results of the transformation processing in semantic terms, independent of the syntactic details of the transformations as (partially) dealt with in [9]. This is captured in the function *STP_secure_E* of [7]. The main conjecture associated with the security properties of the Execution Model as regards the *SELECT* query is then captured by the following definitions, the first of which asserts of the SSQL Transformation Processor, *STP*, that there exist compiler and update subsystems from which it could be constructed.

HOL Constant

Correct_Compile_STP_secure_E : *BOOL*

Correct_Compile_STP_secure_E ⇔
(∃ *compile upd* •
    (*compile*, *upd*) ∈ *Correct_Compile* ∧ *STP* ∈ *STP_secure_E compile*)

HOL Constant

EM_SecureE : *BOOL*

EM_SecureE ⇔ (*Correct_Compile_STP_secure_E* ⇒ *Subsys_SecureE*)

We can use the above definitions to produce a partial proof of overall security in which these critical properties replace *Subsys_SecureE*

SML
```
set_goal(
        [⌜Architecture_Secure⌝,
         ⌜Subsys_SecureA⌝,
         ⌜Subsys_SecureB⌝,
         ⌜Subsys_SecureC⌝,
         ⌜Subsys_SecureD⌝,
         ⌜EM_SecureE⌝,
         ⌜Correct_Compile_STP_secure_E⌝],
                              ⌜FE_SWORD_SYSTEM_secure⌝);
```

SML
```
a(all_asm_ante_tac
        THEN rewrite_tac (map get_spec [⌜EM_SecureE⌝, ⌜Correct_Compile_STP_secure_E⌝]));
a(REPEAT strip_tac
        THEN all_asm_fc_tac[all_⇒_intro Theorem1] THEN all_asm_fc_tac[]);
val Theorem2 = save_pop_thm "Theorem2";
```

6 TABLE COMPUTATIONS

The requirement on the results of the transformation processing captured by *STP_secure_E* is still rather implicit and is expressed in terms of properties of query execution as a whole rather than in terms of the information flow properties which individual SSQL language constructs must possess. The intention of [10] is, in effect, to construct a stronger requirement than *STP_secure_E* by induction over the abstract syntax of SSQL and by means which are independent of the compilation function. This gives a model of query execution which can, at least informally, be related quite directly to the transformation processing.

The main goal of [10] is the construction of a set *TableComputations* of functions whose signatures are the same as (a slight simplification of) those of the result of compiling queries using the compilers considered in the Execution Model. This set is, to be more precise, a family of sets parameterised by the client clearance. The set is constructed by induction over a collection of functions each of which represents the semantics of an SSQL construct, viewed as being executed using the techniques implicit in the transformations. The computation performed by each of these semantic functions may be readily checked against the transformations for the corresponding SSQL construct.

After constructing the set *TableComputations*, [10] then defines two sets which may be used to relate it to the considerations of [7]. The first of these, *OkTableComputation*, is the set of all possible table computations which are secure by dint of a characterisation like that given in [7]. The second, *OkSTP*, is, roughly speaking, the set of objects of the same type as the SSQL Query Transformation Processor, *STP*, which always produce query sequences whose compiled forms perform computations contained in the set *TableComputations*.

We can now define two assertions which we conjecture to be true of these sets. The first of these conjectures asserts that the *OkSTP* is contained in the condition *STP_Secure_E* of [7]:

HOL Constant

OkSTP_Secure_E_Lemma : *BOOL*

OkSTP_Secure_E_Lemma $\Leftrightarrow$
$\forall$ *compile* $\bullet$
$((OkSTP\ compile)$: $(Query,\ FILTER\_PARS)STP\_TYPE\ \mathbb{P}) \subseteq$
$STP\_secure\_E\ compile$

In the above, the type constraint is required, since for technical reasons connected with the primitive definitional mechanisms of ProofPower, it is necessary to constrain the assertion to the particular type of SSQL Transformation Processors with which we are concerned rather than leaving the set as a polymorphic one[2].

The second conjecture asserts that the table computations defined in the previous section do have the *OkTableComputation* property.

HOL Constant

TableComputationsSecure : *BOOL*

TableComputationsSecure $\Leftrightarrow$
$\forall cc \bullet\ TableComputations\ cc \subseteq OkTableComputation\ cc$

The definitions in [10] of the terms involved in these two conjectures have been chosen to make the following proof fairly straightforward:

SML

```
set_goal([], ⌜TableComputationsSecure ⇒ OkSTP_Secure_E_Lemma⌝);
```

[2] ProofPower enforces this restriction since, otherwise, the definition might be inconsistent, because the value of the constant defined might depend on how the polymorphic defining property was instantiated.)

SML

```
a(PC_T1 "hol2"
        rewrite_tac(map get_spec[
                ⌜TableComputationsSecure⌝, ⌜OkTableComputation⌝,
                ⌜OkSTP_Secure_E_Lemma⌝, ⌜ConditionE⌝,
                ⌜OkSTP⌝, ⌜STP_secure_E⌝, ⌜Let⌝]));
a(REPEAT_UNTIL is_∧ strip_tac THEN all_asm_fc_tac[]);
a(lemma_tac⌜compile (Fst (destVal (x (q, c)))) = λ tl• (Snd (dte tl), [])⌝
        THEN1 (asm_rewrite_tac[ext_thm]
                THEN PC_T1"prop_eq_pair" REPEAT strip_tac
                THEN asm_rewrite_tac[]));
a(asm_ante_tac ⌜ri ∈ RiskInputs c (compile (Fst (destVal (x (q, c)))))⌝
        THEN asm_rewrite_tac[]);
a(strip_tac THEN all_asm_fc_tac[] THEN all_asm_fc_tac[]);
a(REPEAT strip_tac);
val Theorem3 = save_pop_thm "Theorem3";
```

We now give the lowest level of partial proof of the overall system security for this document. We use the following definition in stating the result to be proved.

HOL Constant

```
        Correct_Compile_OkSTP : BOOL
─────────────────────────────────────────────────────

        Correct_Compile_OkSTP ⇔
        (∃ compile upd •
                (compile, upd) ∈ Correct_Compile ∧ STP ∈ OkSTP compile)
```

SML

```
set_goal(
        [⌜Architecture_Secure⌝,
         ⌜Subsys_SecureA⌝,
         ⌜Subsys_SecureB⌝,
         ⌜Subsys_SecureC⌝,
         ⌜Subsys_SecureD⌝,
         ⌜EM_SecureE⌝,
         ⌜Correct_Compile_OkSTP⌝,
         ⌜TableComputationsSecure⌝],
                        ⌜FE_SWORD_SYSTEM_secure⌝);
```

SML

```
a(all_asm_ante_tac THEN rewrite_tac (map get_spec
        [⌜Correct_Compile_OkSTP⌝]));
a(REPEAT strip_tac
        THEN PC_T1 "hol2" all_fc_tac[rewrite_rule[get_spec⌜OkSTP_Secure_E_Lemma⌝]
                Theorem3]);
a(fc_tac[rewrite_rule[get_spec⌜Correct_Compile_STP_secure_E⌝](all_⇒_intro Theorem2)]);
a(all_asm_fc_tac[]);
val Theorem4 = save_pop_thm "Theorem4";
```

So, in *Theorem4* the assumption *Subsys_SecureE* has been replaced by three new assumptions, of which we believe that *Correct_Compile_OkSTP* is at least a highly plausible (and certainly desired) property of the implementation, and that the other two are likely to be amenable to further, perhaps partial, proof work.

7 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

SML

```
pop_pc();
```

8 THE THEORY fef034

8.1 Parents

fef032

8.2 Children

fef031

8.3 Constants

FE_SWORD_SYSTEM
(*Query*, *ANSWER*) *BEHAVIOURS*
FE_SWORD_SYSTEM_secure
Bool
Architecture_Secure
Bool
Subsys_SecureE
Bool
Subsys_SecureD
Bool
Subsys_SecureC
Bool
Subsys_SecureB
Bool
Subsys_SecureA
Bool
Correct_Compile_STP_secure_E
Bool
EM_SecureE *Bool*
OkSTP_Secure_E_Lemma
Bool
TableComputationsSecure
Bool
Correct_Compile_OkSTP
Bool

8.4 Definitions

FE_SWORD_SYSTEM
$\vdash$ *FE_SWORD_SYSTEM*
= *FE_SWORD reprState TSQLtf STP outputFilter*
FE_SWORD_SYSTEM_secure
$\vdash$ *FE_SWORD_SYSTEM_secure* $\Leftrightarrow$ *FE_SWORD_SYSTEM* $\in$ *secure*
Architecture_Secure
$\vdash$ *Architecture_Secure*

$$\Leftrightarrow (TSQLtf,\ STP,\ outputFilter) \in subsys\_secure\ reprState \Rightarrow FE\_SWORD\_SYSTEM \in secure$$

Subsys_SecureA
Subsys_SecureB
Subsys_SecureC
Subsys_SecureD
Subsys_SecureE

$$\vdash (Subsys\_SecureA \Leftrightarrow (TSQLtf,\ STP) \in subsys\_secureA\ reprState) \wedge (Subsys\_SecureB \Leftrightarrow (TSQLtf,\ STP) \in subsys\_secureB\ reprState) \wedge (Subsys\_SecureC \Leftrightarrow (TSQLtf,\ STP) \in subsys\_secureC\ reprState) \wedge (Subsys\_SecureD \Leftrightarrow (TSQLtf,\ STP) \in subsys\_secureD\ reprState) \wedge (Subsys\_SecureE \Leftrightarrow (TSQLtf,\ STP,\ outputFilter) \in subsys\_secureE\ reprState)$$

Correct_Compile_STP_secure_E

$$\vdash Correct\_Compile\_STP\_secure\_E \Leftrightarrow (\exists\ compile\ upd \bullet (compile,\ upd) \in Correct\_Compile \wedge STP \in STP\_secure\_E\ compile)$$

EM_SecureE $\vdash EM\_SecureE \Leftrightarrow Correct\_Compile\_STP\_secure\_E \Rightarrow Subsys\_SecureE$

OkSTP_Secure_E_Lemma

$$\vdash OkSTP\_Secure\_E\_Lemma \Leftrightarrow (\forall\ compile \bullet OkSTP\ compile \subseteq STP\_secure\_E\ compile)$$

TableComputationsSecure

$$\vdash TableComputationsSecure \Leftrightarrow (\forall\ cc \bullet TableComputations\ cc \subseteq OkTableComputation\ cc)$$

Correct_Compile_OkSTP

$$\vdash Correct\_Compile\_OkSTP \Leftrightarrow (\exists\ compile\ upd \bullet (compile,\ upd) \in Correct\_Compile \wedge STP \in OkSTP\ compile)$$

8.5 Theorems

Theorem1

$$Architecture\_Secure, Subsys\_SecureA, Subsys\_SecureB, Subsys\_SecureC, Subsys\_SecureD, Subsys\_SecureE \vdash FE\_SWORD\_SYSTEM\_secure$$

Theorem2 $Architecture\_Secure$,
$Subsys\_SecureA$,
$Subsys\_SecureB$,
$Subsys\_SecureC$,
$Subsys\_SecureD$,
$EM\_SecureE$,
$Correct\_Compile\_STP\_secure\_E$
$\vdash FE\_SWORD\_SYSTEM\_secure$

Theorem3 $\vdash TableComputationsSecure \Rightarrow OkSTP\_Secure\_E\_Lemma$

Theorem4 $Architecture\_Secure$,
$Subsys\_SecureA$,
$Subsys\_SecureB$,
$Subsys\_SecureC$,
$Subsys\_SecureD$,
$EM\_SecureE$,
$Correct\_Compile\_OkSTP$,
$TableComputationsSecure$
$\vdash FE\_SWORD\_SYSTEM\_secure$

9 INDEX

Architecture_Secure 7
Correct_Compile_OkSTP 11
Correct_Compile_STP_secure_E 8
EM_SecureE 8
fef034 5
FE_SWORD_SYSTEM_secure 5
FE_SWORD_SYSTEM 5
OkSTP_Secure_E_Lemma 10
Subsys_SecureA 7
Subsys_SecureB 7
Subsys_SecureC 7
Subsys_SecureD 7
Subsys_SecureE 7
SWORD_architecture_secure_conj 6
TableComputationsSecure 10
Theorem1 8
Theorem2 9
Theorem3 11
Theorem4 12