Project: DRA FRONT END FILTER PROJECT

Title: Phase II Proof Finale

Ref: DS/FMU/FEF/036 *Issue: Revision* : *2.1* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Authors:

Name	*Location*	*Signature*	*Date*
R. D. Arthan	WIN01		
G. M. Prout	WIN01		

Authorisation for Issue:

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

Abstract: This document completes the table computations security proof and uses it with the partial results on the execution model security to complete the programme of partial proof work 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 **4**

3 TABLE COMPUTATIONS SECURITY PROOF **5**
3.1 Induction Principle . . . 5
3.2 Proof of *TableComputationsSecure* . . . 6

4 PHASE II PROOF SUMMARY **11**
4.1 Problems Discovered During Proof Work . . . 12
4.2 Phase II Partial Proof . . . 12

5 THE THEORY fef036 **14**
5.1 Parents . . . 14
5.2 Children . . . 14
5.3 Theorems . . . 14

6 INDEX **17**

0.2 Document Cross References

[1] DS/FMU/FEF/002. *Errors in the Specifications.* G.M. Prout, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/018. *Proposal for Phase 2.* G.M. Prout, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/031. *Execution Model Security Proofs.* R.D. Arthan, ICL Secure Systems, WIN01.

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

[5] DS/FMU/FEF/033. *Value Computation Security Proofs.* R.D. Arthan, ICL Secure Systems, WIN01.

[6] DS/FMU/FEF/034. *Phase II Proof Strategy.* R.D. Arthan, ICL Secure Systems, WIN01.

[7] DS/FMU/FEF/035. *Table Computation Security Proofs.* R.D. Arthan and G.M. Prout, ICL Secure Systems, WIN01.

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

0.3 Changes History

Issue 1.1 (4 November 1993) First draft.

Issue 1.2 (5 November 1993) First draft for review by DRA.

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 provides a formal proof of the conjecture *TableComputationsSecure* from the proof strategy, [6]. This result is used, together with the partial proof achieved in [3] of the execution model conjecture *EM_SecureE* of [6], to produce the best partial proof for Phase II. It constitutes part of deliverable D13 of work package 3, as given in section 3 of the Proposal for Phase 2, [2].

1.2 Introduction

In "Table Computations for SWORD", [4], semantic functions for the value and table computations are defined together with corresponding critical properties OK_VC_d, OK_VC_c, OK_TC_d and OK_TC_c. Documents [5] and [7] provide formal proofs that all these semantics functions satisfy their identified critical properties (which constitute inductive steps in a proof by induction over the syntax of SSQL). We now bring together these proofs to prove the conjecture *TableComputationsSecure* of the proof strategy, [6]. This is achieved via an induction principle for the set of semantic functions, in a similar manner to that of the example in section 3.3 of [4].

During the production of the specifications of the semantic functions and during the process of proving that they satisfy their identified critical properties, errors in the underlying query transformation specifications of [8] came to light. These errors are significant in that they if they are mirrored in the implementation then, the implementation will not be secure. We give a brief account of measures taken to correct the specifications.

Finally, we provide our sharpest partial result on the overall system security. The theorems proved in [3, 5, 7] are used to improve on *Theorem4* of the proof strategy, [6].

2 PRELIMINARIES

The following ProofPower instructions set up a new theory *fef036* to hold the theorems to be proved and set up a proof context in which to carry out the proofs.

SML

```
open_theory "fef035";
(force_delete_theory "fef036" handle _ => ());
new_theory"fef036";
set_pc"hol";
```

3 TABLE COMPUTATIONS SECURITY PROOF

3.1 Induction Principle

[4] gives an account of reducing the critical requirements on the SSQL Transformation Processor by placing a bound on the allowed TSQL queries it produces. This involved a reformulation of the classification computations which underlie the query transformations defined in [8] in terms of a relational algebra model of SSQL execution. The semantic functions defined in [4] bring out the security checks which correspond to each of the constructors of the SSQL language. The overall security checks required by the SSQL semantics are then reduced to properties of the individual constructors by an induction principle for the set of semantic functions.

We first state and prove the induction principle as a HOL theorem.

SML

```
set_goal([], ⌜∀cc:Class•
        ∀tcs : TABLE_COMP SET; vcs : VALUE_COMP SET•
                (∀ci• DenoteConstant ci ∈ vcs)
        ∧       (∀i• Contents i ∈ vcs)
        ∧       (∀i• Classification i ∈ vcs)
        ∧       CountAll ∈ vcs

        ∧       (∀f e• e ∈ vcs ⇒ MonOp f e ∈ vcs)
        ∧       (∀f e1 e2• e1 ∈ vcs ∧ e2 ∈ vcs ⇒ BinOp f e1 e2 ∈ vcs)
        ∧       (∀f e1 e2 e3• e1 ∈ vcs ∧ e2 ∈ vcs ∧ e3 ∈ vcs ⇒ TriOp f e1 e2 e3 ∈ vcs)
        ∧       (∀el• Elems el ⊆ vcs ⇒ BinOpAnd cc el ∈ vcs)
        ∧       (∀el• Elems el ⊆ vcs ⇒ BinOpOr cc el ∈ vcs)

        ∧       (∀te cel ee• te ∈ vcs ∧ Elems(Map Fst cel) ⊆ vcs ∧
                                Elems(Map Snd cel) ⊆ vcs ∧ ee ∈ vcs ⇒
                                        CaseVal cc te cel ee ∈ vcs)
        ∧       (∀cel ee• Elems(Map Fst cel) ⊆ vcs ∧
                                Elems(Map Snd cel) ⊆ vcs ∧ ee ∈ vcs  ⇒
                                        Case cc cel ee ∈ vcs)

        ∧       (∀e• e ∈ vcs ⇒ SetFuncAllAnd cc e ∈ vcs)
        ∧       (∀e• e ∈ vcs ⇒ SetFuncAllOr cc e ∈ vcs)
        ∧       (∀e• e ∈ vcs ⇒ CountNonNull e ∈ vcs)
        ∧       (∀e• e ∈ vcs ⇒ CountDistinct e ∈ vcs)
        ∧       (∀e• e ∈ vcs ⇒ CommonValue e ∈ vcs)

        ∧       (∀f e• e ∈ vcs ⇒ SetFuncAll f e ∈ vcs)
        ∧       (∀f e• e ∈ vcs ⇒ SetFuncDistinct f e ∈ vcs)

        ∧       (∀te• te ∈ tcs ⇒ ExistsTuples cc te ∈ vcs)
```

```
|       ∧       (∀te• te ∈ tcs ⇒ SingleValue cc te ∈ vcs)
|
|       ∧       (JoinedRowExistence cc ∈ vcs)
|       ∧       (∀i• TableContents i ∈ tcs)
|
|       ∧       (∀esl tel e1 ml nl e2• Elems(Map Fst esl) ⊆ vcs
|                       ∧ Elems tel ⊆ tcs ∧ e1 ∈ vcs ∧ e2 ∈ vcs
|                               ⇒ AllTuples cc esl tel e1 ml nl e2 ∈ tcs)
|
|⇒              TableComputations cc ⊆ tcs
|       ∧       ValueComputations cc ⊆ vcs
|⌝);
```

SML

```
|a(REPEAT ∀_tac THEN strip_tac
|       THEN rewrite_tac[rewrite_rule[get_spec⌜⋂2⌝] (get_spec⌜TableComputations⌝)]
|       THEN PC_T1"sets_ext1" REPEAT strip_tac);
|(* *** Goal "1" *** *)
|a(POP_ASM_T (strip_asm_tac o ∀_elim⌜tcs⌝)
|       THEN POP_ASM_T (ante_tac o ∀_elim⌜vcs⌝)
|       THEN asm_rewrite_tac[]);
|(* *** Goal "2" *** *)
|a(POP_ASM_T (strip_asm_tac o ∀_elim⌜vcs⌝)
|       THEN POP_ASM_T (ante_tac o ∀_elim⌜tcs⌝)
|       THEN asm_rewrite_tac[]);
|val table_computation_induction_thm =
|       save_pop_thm"table_computation_induction_thm";
```

3.2 Proof of *TableComputationsSecure*

We now wish to prove the conjecture *TableComputationsSecure* from [6], which asserts the following:

```
|?⊢     ∀ cc• TableComputations cc ⊆ OkTableComputation cc
```

Now the following theorem , $OK\_TC_d\_lemma$, has been proved in [5]:

```
|⊢      ∀ c• OK_TC_d c ⊆ OkTableComputation c
```

To prove *TableComputationsSecure*, it is therefore sufficient to prove:

```
|?⊢     ∀ cc•  TableComputations cc ⊆ OK_TC_d cc
```

We prove the above by an induction over the syntax of table computations. Now the set of table computations is defined in [4] simultaneously with the set of value computations. Moreover, it

turns out that in the inductive steps for the *AllTuple* form of table computation we need a rather stronger inductive hypothesis than the above. The sets OK_VC_d and OK_VC_c defined in [4] embody the hypotheses required on the value computations and the set OK_TC_c gives the necessary strengthening of the hypotheses on the table computations. Thus we shall prove the following stronger result:

$$\begin{array}{|ll} ?\vdash & \forall\ cc \bullet\ \ TableComputations\ cc \subseteq OK\_TC_d\ cc \cap OK\_TC_c\ cc \\ & \wedge\ \ \ ValueComputations\ cc \subseteq OK\_VC_d\ cc \cap OK\_VC_c\ cc \end{array}$$

Now in the proofs of the inductive steps in [5, 7], the $OK\_\mathrm{x}C_d$ and $OK\_\mathrm{x}C_c$ properties are handled separately and, by and large, only those parts of the hypotheses which are required are stated and used. This may be seen in the following extract from the listing of theory *fef033*:

$$\begin{array}{|l} BinOp\_OK_d\_lemma \\ \quad \vdash \forall\ c\ f\ vc1\ vc2 \\ \quad\ \bullet\ vc1 \in OK\_VC_d\ c \wedge vc2 \in OK\_VC_d\ c \\ \quad\quad \Rightarrow BinOp\ f\ vc1\ vc2 \in OK\_VC_d\ c \\ ... \\ BinOp\_OK_c\_lemma \\ \quad \vdash \forall\ c\ f\ vc1\ vc2 \\ \quad\ \bullet\ vc1 \in OK\_VC_c\ c \wedge vc2 \in OK\_VC_c\ c \\ \quad\quad \Rightarrow BinOp\ f\ vc1\ vc2 \in OK\_VC_c\ c \\ ... \\ Case\_OK_d\_lemma \quad \vdash \forall\ c\ cel\ ee \\ \quad\ \bullet\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_d\ c \\ \quad\quad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq OK\_VC_d\ c \\ \quad\quad \wedge\ ee \in OK\_VC_d\ c \\ \quad\quad \Rightarrow Case\ c\ cel\ ee \in OK\_VC_d\ c \\ ... \\ Case\_OK_c\_lemma \quad \vdash \forall\ c\ cel\ ee \\ \quad\ \bullet\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_d\ c \\ \quad\quad \wedge\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_c\ c \\ \quad\quad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq OK\_VC_c\ c \\ \quad\quad \wedge\ ee \in OK\_VC_c\ c \\ \quad\quad \Rightarrow Case\ c\ cel\ ee \in OK\_VC_c\ c \end{array}$$

So, for example, $Case\_OK_c\_lemma$, the part of the inductive step for the *Case* (conditional) value form concerned with OK_VC_c, requires the constituents giving the boolean conditions (first components) to lie in both OK_VC_d and OK_VC_c, whereas the constituents giving the result values (second components) need only lie in OK_VC_c. Thus, the results proved in [5, 7] yield slightly more detailed information than is actually required for the inductive proof.

To apply the induction principle of the previous section, we need to use the various lemmas of [5, 7] to derive the precise form needed for the theorems giving the inductive steps. For example, we need the following for the *Case* form:

$$
\begin{array}{l}
\vdash \forall\ c\ cel\ ee \\
\quad \bullet\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow Case\ c\ cel\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c
\end{array}
$$

Since there are 23 inductive steps to be dealt with, it is convenient to use the programmable access to the theory database that ProofPower provides to assemble the proofs of the inductive steps in the desired form. In the rest of this section, we do this.

First of all, we set an ML variable *ok_vc_tc_conjs* to hold terms expressing the desired inductive steps. At the cost of some clarity these terms could alternatively all be computed by substitution into the constituents of the theorem expressing the induction principle.

SML

$$
\begin{array}{l}
val\ \mathbf{ok\_vc\_tc\_conjs} : TERM\ list = map\ (curry\ mk\_\forall\ \ulcorner c{:}Class \urcorner) \\
[\\
\ulcorner \forall ci \bullet\ DenoteConstant\ ci \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall i \bullet\ Contents\ i \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall i \bullet\ Classification\ i \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner CountAll \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner,
\end{array}
$$

SML

$$
\begin{array}{l}
\ulcorner \forall f\ e \bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow MonOp\ f\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall f\ e1\ e2 \bullet\ e1 \in OK\_VC_d\ c \cap OK\_VC_c\ c \wedge e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow \\
\qquad BinOp\ f\ e1\ e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall f\ e1\ e2\ e3 \bullet\ e1 \in OK\_VC_d\ c \cap OK\_VC_c\ c \wedge e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \wedge \\
\quad e3 \in OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow \\
\quad TriOp\ f\ e1\ e2\ e3 \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall el \bullet\ Elems\ el \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow BinOpAnd\ c\ el \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall el \bullet\ Elems\ el \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow BinOpOr\ c\ el \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner,
\end{array}
$$

SML

$$
\begin{array}{l}
\ulcorner \forall te\ cel\ ee \bullet \\
\quad te \in OK\_VC_d\ c \cap OK\_VC_c\ c \wedge Elems(Map\ Fst\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \wedge \\
\quad Elems(Map\ Snd\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \wedge ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \Rightarrow \\
\qquad CaseVal\ c\ te\ cel\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner, \\
\ulcorner \forall cel\ ee \bullet\ Elems(Map\ Fst\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \wedge \\
\quad Elems(Map\ Snd\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \wedge ee \in OK\_VC_d\ c \cap OK\_VC_c\ c\ \Rightarrow \\
\qquad Case\ c\ cel\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \urcorner,
\end{array}
$$

SML

```
⌜∀e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ SetFuncAllAnd c e ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ SetFuncAllOr c e ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ CountNonNull e ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ CountDistinct e ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ CommonValue e ∈ OK_VC_d c ∩ OK_VC_c c⌝,

⌜∀f e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ SetFuncAll f e ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀f e• e ∈ OK_VC_d c ∩ OK_VC_c c ⇒ SetFuncDistinct f e ∈ OK_VC_d c ∩ OK_VC_c c⌝,

⌜∀te• te ∈ OK_TC_d c ∩ OK_TC_c c ⇒ ExistsTuples c te ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜∀te• te ∈ OK_TC_d c ∩ OK_TC_c c ⇒ SingleValue c te ∈ OK_VC_d c ∩ OK_VC_c c⌝,
⌜JoinedRowExistence c ∈ OK_VC_d c ∩ OK_VC_c c⌝,
```

SML

```
⌜∀ i• TableContents i ∈ OK_TC_d c ∩ OK_TC_c c⌝,
⌜∀ esl tel e1 ml nl e2•
        Elems (Map Fst esl) ⊆ OK_VC_d c ∩ OK_VC_c c ∧
        Elems tel ⊆ OK_TC_d c ∩ OK_TC_c c ∧    e1 ∈ OK_VC_d c ∩ OK_VC_c c ∧
        e2 ∈ OK_VC_d c ∩ OK_VC_c c
                ⇒ AllTuples c esl tel e1 ml nl e2 ∈ OK_TC_d c ∩ OK_TC_c c⌝
];
```

We will need some theorems to use as rewrite rules to eliminate ∩. The following commands cause these theorems to be proved automatically and held in the ML variable ∩_*thms* for future reference.

SML

```
val ∩_thms : THM list = map (pc_rule1"sets_ext1"prove_rule[]) [
        ⌜∀x a b•x ∈ a ∩ b ⇔ x ∈ a ∧ x ∈ b⌝,
        ⌜∀c a b•c ⊆ a ∩ b ⇔ c ⊆ a ∧ c ⊆ b⌝];
```

We now retrieve all the lemmas from [5] and [7] where the right hand side of the conclusion is of the form $\in OK\_VC_d$, $\in OK\_VC_c$, $\in OK\_TC_d$, or $\in OK\_TC_d$. We set an ML variable *ok_vc_tc_thms* to hold this list of lemmas, each lemma having any uses of ∩ expanded out and each being tagged with a term indicating which language construct the lemma is concerned with.

SML
```
val ok_vc_tc_thms : (TERM * THM) list =
        map
        (fn (_, t) =>
                ((hd o rev o strip_⇒ o snd o strip_∀ o concl) t, rewrite_rule ∩_thms t
                                                                handle Fail _ => t))
        ((get_thms"fef033" @ get_thms"fef035")
        drop
        (fn t =>
                let     val x = (fst o dest_app o snd o dest_app o hd o rev o
                                        strip_⇒ o snd o strip_∀ o concl o snd) t
                in      not(x =$ ⌜OK_VC_d⌝) andalso not(x =$ ⌜OK_VC_c⌝)
                        andalso not(x =$ ⌜OK_TC_d⌝) andalso not(x =$ ⌜OK_TC_c⌝)
                end     handle Fail _ => true));
```

Given one of the subgoals arising when ∩ is eliminated from one of the inductive steps, the following tactic, *ok_vc_tc_tac*, finds the appropriate theorem in *ok_vc_tc_thms* to solve the subgoal and uses it to do just that:

SML
```
val ok_vc_tc_tac : TACTIC = (fn gl as (_, cnc) =>
        let     val th = snd(find ok_vc_tc_thms
                        (fn (t, _) =>
                                (term_match cnc t; true) handle Fail _ => false));
        in      SOLVED_T (all_fc_tac[th] THEN asm_rewrite_tac[])
                        ORELSE rewrite_tac[th]
        end     gl
);
```

The following rule (i.e. theorem-computing function) is intended to be given a term expressing one of the inductive steps as its argument; it eliminates ∩ from the term and then uses *ok_vc_tc_tac* to prove it.

SML
```
fun ok_vc_tc_rule (t : TERM) : THM = (
        tac_proof(([], t), once_rewrite_tac ∩_thms THEN REPEAT strip_tac THEN ok_vc_tc_tac)
);
```

The list of theorems giving the inductive steps are now proved by mapping *ok_vc_tc_rule* over *ok_vc_tc_conjs*. The results are then conjoined to give a single theorem (using *list_∧_intro*) for later use. The theorem is saved in the theory as *ok_vc_tc_lemmas* and may be seen in the theory listing at the end of this document.

SML
```
val ok_vc_tc_lemmas : THM = save_thm("ok_vc_tc_lemmas",
        list_∧_intro(map ok_vc_tc_rule ok_vc_tc_conjs));
```

Now we can carry out the inductive proof of *TableComputationsSecure*.

```
SML
set_goal([], ⌜∀ cc•
                TableComputations cc ⊆ OK_TC_d cc ∩ OK_TC_c cc
        ∧       ValueComputations cc ⊆ OK_VC_d cc ∩ OK_VC_c cc
⌝);
a ∀_tac;
a(ante_tac(list_∀_elim[
                        ⌜cc:Class⌝,
                        ⌜OK_TC_d cc ∩ OK_TC_c cc⌝,
                        ⌜OK_VC_d cc ∩ OK_VC_c cc⌝]
                table_computation_induction_thm)
        THEN asm_rewrite_tac [ok_vc_tc_lemmas]);
val fef036_main_lemma = save_pop_thm"fef036_main_lemma";
```

The proof of *TableComputationsSecure* is now easily completed:

```
SML
set_goal([], ⌜TableComputationsSecure⌝);
a(rewrite_tac[get_spec⌜TableComputationsSecure⌝]);
a(strip_asm_tac OK_TC_d_lemma);
a(strip_asm_tac fef036_main_lemma);
a(PC_T1 "sets_ext1" asm_prove_tac[]);
val TableComputationsSecure_thm = save_pop_thm"TableComputationsSecure_thm";
```

4 PHASE II PROOF SUMMARY

In the proof strategy, [6], three main areas were identified as amenable to formal reasoning :

1. Architectural Model
2. *TSQL* ExecutionModel
3. Table Computations

Most of the proof effort has been concerned with area (3). The conjecture *TableComputationsSecure* has been proved. In addition work in area (2) has resulted in a partial proof of the security of the TSQL Execution Model. The result of this is the replacement of the conjecture *EM_SecureE*,defined in [6], by two more detailed assumptions, $View_t$_*secureE* and *outputFilter_secureE*. The first of these is an information flow property stating that if two *SSQL* states are the same when viewed at a particular classification then their representations as *TSQL* states when viewed as lists of derived tables are also the same at that classification. The second is an information flow property on the output filter.

4.1 Problems Discovered During Proof Work

One of the most important outcomes of the formal proof work carried out in phase II was the discovery of problems in the specifications of [4] which model the query transformations of [8].

Initially, we identified the requirement of the critical property OK_TC_d on table computations and the associated OK_VC_d on value computations. Essentially OK_TC_d and OK_VC_d capture the required information flow properties. However, we discovered during proof work that the classification label which is output by the system could itself be a covert channel. It was necessary to strengthen the requirements on table and value computations; OK_TC_c and OK_VC_c were introduced for this purpose. Further problems emerged as a result of strengthening the requirements on table and value computations. In particular, the specifications of *Group* and *AllTuples* were changed to ensure that output clearances were not a source of insecurity.

In addition, an error in the specification of *Where* of [4] was discovered during proof work. It was necessary to change the specification so that the value computation only used rows about whose existence the client is cleared to know. This ensures, for example, that aggregate functions such as $COUNT(*)$ inside the *WHERE* clause itself are not a source of covert channels.

A fuller account of problems encountered during proof work may be found in [1].

4.2 Phase II Partial Proof

We now use the proof of *TableComputationsSecure* together with the partial proof from [3] of *EM_SecureE* to update *Theorem4* of [6] to give the best partial proof of the overall system security.

SML

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

SML

```
a(strip_asm_tac EM_SecureE_thm THEN
        asm_tac TableComputationsSecure_thm THEN
        strip_asm_tac Theorem4);
val Theorem5 = save_pop_thm "Theorem5";
```

ProofPower output

$$
\begin{array}{l}
Theorem5 = \\
\quad Architecture\_Secure, \\
\quad Subsys\_SecureA, \\
\quad Subsys\_SecureB, \\
\quad Subsys\_SecureC, \\
\quad Subsys\_SecureD, \\
\quad View_t\_secureE, \\
\quad outputFilter\_secureE, \\
\quad Correct\_Compile\_OkSTP \\
\vdash FE\_SWORD\_SYSTEM\_secure
\end{array}
$$

5 THE THEORY fef036

5.1 Parents

fef035

5.2 Children

fef042

5.3 Theorems

table_computation_induction_thm

$$
\begin{array}{l}
\vdash \forall\ cc\ tcs\ vcs \\
\quad \bullet\ (\forall\ ci \bullet\ DenoteConstant\ ci \in vcs) \\
\qquad \wedge\ (\forall\ i \bullet\ Contents\ i \in vcs) \\
\qquad \wedge\ (\forall\ i \bullet\ Classification\ i \in vcs) \\
\qquad \wedge\ CountAll \in vcs \\
\qquad \wedge\ (\forall\ f\ e \bullet\ e \in vcs \Rightarrow MonOp\ f\ e \in vcs) \\
\qquad \wedge\ (\forall\ f\ e1\ e2 \\
\qquad \bullet\ e1 \in vcs \wedge e2 \in vcs \Rightarrow BinOp\ f\ e1\ e2 \in vcs) \\
\qquad \wedge\ (\forall\ f\ e1\ e2\ e3 \\
\qquad \bullet\ e1 \in vcs \wedge e2 \in vcs \wedge e3 \in vcs \\
\qquad\quad \Rightarrow TriOp\ f\ e1\ e2\ e3 \in vcs) \\
\qquad \wedge\ (\forall\ el \bullet\ Elems\ el \subseteq vcs \Rightarrow BinOpAnd\ cc\ el \in vcs) \\
\qquad \wedge\ (\forall\ el \bullet\ Elems\ el \subseteq vcs \Rightarrow BinOpOr\ cc\ el \in vcs) \\
\qquad \wedge\ (\forall\ te\ cel\ ee \\
\qquad \bullet\ te \in vcs \\
\qquad\quad \wedge\ Elems\ (Map\ Fst\ cel) \subseteq vcs \\
\qquad\quad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq vcs \\
\qquad\quad \wedge\ ee \in vcs \\
\qquad\quad \Rightarrow CaseVal\ cc\ te\ cel\ ee \in vcs) \\
\qquad \wedge\ (\forall\ cel\ ee \\
\qquad \bullet\ Elems\ (Map\ Fst\ cel) \subseteq vcs \\
\qquad\quad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq vcs \\
\qquad\quad \wedge\ ee \in vcs \\
\qquad\quad \Rightarrow Case\ cc\ cel\ ee \in vcs) \\
\qquad \wedge\ (\forall\ e \bullet\ e \in vcs \Rightarrow SetFuncAllAnd\ cc\ e \in vcs) \\
\qquad \wedge\ (\forall\ e \bullet\ e \in vcs \Rightarrow SetFuncAllOr\ cc\ e \in vcs) \\
\qquad \wedge\ (\forall\ e \bullet\ e \in vcs \Rightarrow CountNonNull\ e \in vcs) \\
\qquad \wedge\ (\forall\ e \bullet\ e \in vcs \Rightarrow CountDistinct\ e \in vcs) \\
\qquad \wedge\ (\forall\ e \bullet\ e \in vcs \Rightarrow CommonValue\ e \in vcs) \\
\qquad \wedge\ (\forall\ f\ e \bullet\ e \in vcs \Rightarrow SetFuncAll\ f\ e \in vcs) \\
\qquad \wedge\ (\forall\ f\ e \bullet\ e \in vcs \Rightarrow SetFuncDistinct\ f\ e \in vcs) \\
\qquad \wedge\ (\forall\ te \bullet\ te \in tcs \Rightarrow ExistsTuples\ cc\ te \in vcs) \\
\qquad \wedge\ (\forall\ te \bullet\ te \in tcs \Rightarrow SingleValue\ cc\ te \in vcs) \\
\qquad \wedge\ JoinedRowExistence\ cc \in vcs
\end{array}
$$

$$
\begin{array}{l}
\quad \wedge\ (\forall\ i \bullet\ TableContents\ i \in tcs) \\
\quad \wedge\ (\forall\ esl\ tel\ e1\ ml\ nl\ e2 \\
\quad \bullet\ Elems\ (Map\ Fst\ esl) \subseteq vcs \\
\qquad \wedge\ Elems\ tel \subseteq tcs \\
\qquad \wedge\ e1 \in vcs \\
\qquad \wedge\ e2 \in vcs \\
\qquad \Rightarrow\ AllTuples\ cc\ esl\ tel\ e1\ ml\ nl\ e2 \in tcs) \\
\Rightarrow\ TableComputations\ cc \subseteq tcs \\
\quad \wedge\ ValueComputations\ cc \subseteq vcs
\end{array}
$$

ok_vc_tc_lemmas

$$
\begin{array}{l}
\vdash\ (\forall\ c\ ci \bullet\ DenoteConstant\ ci \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ i \bullet\ Contents\ i \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ i \bullet\ Classification\ i \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c \bullet\ CountAll \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ f\ e \\
\quad \bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ MonOp\ f\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ f\ e1\ e2 \\
\quad \bullet\ e1 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ BinOp\ f\ e1\ e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ f\ e1\ e2\ e3 \\
\quad \bullet\ e1 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ e3 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ TriOp\ f\ e1\ e2\ e3 \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ el \\
\quad \bullet\ Elems\ el \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ BinOpAnd\ c\ el \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ el \\
\quad \bullet\ Elems\ el \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ BinOpOr\ c\ el \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ te\ cel\ ee \\
\quad \bullet\ te \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ CaseVal\ c\ te\ cel\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ cel\ ee \\
\quad \bullet\ Elems\ (Map\ Fst\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ Elems\ (Map\ Snd\ cel) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \wedge\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ Case\ c\ cel\ ee \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ e \\
\quad \bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\qquad \Rightarrow\ SetFuncAllAnd\ c\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\quad \wedge\ (\forall\ c\ e \\
\quad \bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c
\end{array}
$$

$$
\begin{array}{l}
\quad \Rightarrow SetFuncAllOr\ c\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ e \\
\bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow CountNonNull\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ e \\
\bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow CountDistinct\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ e \\
\bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow CommonValue\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ f\ e \\
\bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow SetFuncAll\ f\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ f\ e \\
\bullet\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow SetFuncDistinct\ f\ e \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ te \\
\bullet\ te \in OK\_TC_d\ c \cap OK\_TC_c\ c \\
\quad \Rightarrow ExistsTuples\ c\ te \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ te \\
\bullet\ te \in OK\_TC_d\ c \cap OK\_TC_c\ c \\
\quad \Rightarrow SingleValue\ c\ te \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c \\
\bullet\ JoinedRowExistence\ c \in OK\_VC_d\ c \cap OK\_VC_c\ c) \\
\wedge\ (\forall\ c\ i\bullet\ TableContents\ i \in OK\_TC_d\ c \cap OK\_TC_c\ c) \\
\wedge\ (\forall\ c\ esl\ tel\ e1\ ml\ nl\ e2 \\
\bullet\ Elems\ (Map\ Fst\ esl) \subseteq OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \wedge\ Elems\ tel \subseteq OK\_TC_d\ c \cap OK\_TC_c\ c \\
\quad \wedge\ e1 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \wedge\ e2 \in OK\_VC_d\ c \cap OK\_VC_c\ c \\
\quad \Rightarrow AllTuples\ c\ esl\ tel\ e1\ ml\ nl\ e2 \\
\quad \in OK\_TC_d\ c \cap OK\_TC_c\ c)
\end{array}
$$

fef036_main_lemma

$$
\begin{array}{l}
\vdash\ \forall\ cc \\
\bullet\ TableComputations\ cc \subseteq OK\_TC_d\ cc \cap OK\_TC_c\ cc \\
\quad \wedge\ ValueComputations\ cc \subseteq OK\_VC_d\ cc \cap OK\_VC_c\ cc
\end{array}
$$

TableComputationsSecure_thm

$$\vdash\ TableComputationsSecure$$

Theorem5

$$
\begin{array}{l}
Architecture\_Secure, \\
Subsys\_SecureA, \\
Subsys\_SecureB, \\
Subsys\_SecureC, \\
Subsys\_SecureD, \\
View_t\_secureE, \\
outputFilter\_secureE, \\
Correct\_Compile\_OkSTP \\
\vdash\ FE\_SWORD\_SYSTEM\_secure
\end{array}
$$

6 INDEX

fef036_main_lemma 11
fef036 4
ok_vc_tc_conjs 8
ok_vc_tc_lemmas 10
ok_vc_tc_rule 10
ok_vc_tc_tac 10
ok_vc_tc_thms 10
TableComputationsSecure_thm 11
table_computation_induction_thm 6
Theorem5 12
∩*_thms* 9