THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

EVALUATING THE TLS FAMILY OF PROTOCOLS WITH WEAKEST PRECONDITION REASONING

By

JUSTIN CHILDS

TECH REPORT TR-000703

July 2000

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this paper we show how the Cryptographic Protocol Analysis Language Evaluation System (CPALES) was used in the analysis of the Transport Layer Security (TLS) protocol. CPALES was also used to reveal flaws in one of the security protocols developed by Kao and Chow. The TLS protocol is actually composed of a set of subprotocols. The difficulties of analyzing a protocol of this nature and the resulting improvements made to CPALES are discussed. We demonstrate the ability of the system to analyze TLS for insecurities arising from interactions among subprotocols.

1. INTRODUCTION

Security protocols provide algorithms for communicating securely. An insecure medium is made secure using the tools of cryptography. These tools include symmetric key cryptography, public key cryptography, hashing, signatures, certificates, key agreement, and other mechanisms. These are the tools that allow secure communication. Security protocols attempt to use these tools to generate secure interactions. However, security protocols can be flawed and subverted. One method of finding these flaws involves the use of formal methods, which were originally used to analyze programs and algorithms.

Protocols specify the actions of a set of principals. This resembles the actions and operations of a function, algorithm or a program. Like a program, a protocol can have flaws that prevent it from operating correctly. Many proposed protocols are short and simple. However, even with the most basic specifications, flaws have often been discovered only belatedly. This can be attributed to the environment in which security protocols' operate. Here, the messages sent between principals can be read, modified and deleted by anyone while traveling between the two principals. This was likened to "programming Satan's computer" [1] by Anderson and Needham. This view has been supported by the dis-

covery of security flaws in protocols years after they were initially proposed. The Needham and Schroeder public key protocol [21] is good example. Gavin Lowe discovered a weakness in the protocol seventeen years after its publication [16].

This is not an isolated incident. Flaws have turned up in many protocols, forcing the realization that protocols are difficult to design correctly. Protocols are often utilized in critical systems. Formal methods have been a traditional remedy for the correct design and testing of complex and critical systems. The high cost of formal methods is offset by the necessity for correctness in security applications and the difficulty in verifying security through informal means.

Different methodologies for verifying protocols have been explored. These have been termed “inference construction methods”, “attack construction methods”, and “proof construction methods” by Gritzalis and Spinellis [10]. Attack construction methods attempt to create sets of attacks from the algebraic properties of the protocol. It utilizes models of the actual computations to formally model protocols and to prove theorems about the protocols. The “inference construction method” analyzes the state of belief and/or knowledge of the participants in a protocol. These are usually termed protocol analysis logics. The most recognized logics, BAN [2], GNY [9], and SvO [25] are similar in operation. They deal with authentication and key distribution. The beliefs of the principals in a protocol are examined to see if the appropriate beliefs are reached. The demonstration of this, aids in the discovery of flaws and verification of protocols. The last method is similar to inference construction methods, but typically involves

more involved specification of protocols, and often requires generation of possible actions for analysis.

These analysis methods have enjoyed success in the analysis of traditional protocols found in the literature. Unfortunately for the users of these methods, standards bodies and protocol designers have not limited their designs according to the capabilities of the analysis tools. These new protocols utilize collections of subprotocols that give users the flexibility to choose their method of exchange. This has led to a gap between the complexity of the protocols and the ability of the analysis tools to handle that complexity.

The closing of this gap has just begun. As of yet, only the NRL Protocol Analyzer has provided the necessary means to handle the new protocols. It has been used to analyze IKE [18]. The NRL Protocol Analyzer models attacks with a set of rules that govern the actions of an intruder. This tool falls into the family of tools utilizing attack construction methods. Up to this point, proof construction and inference construction methods have not been created to analyze these families of protocols. To help close this gap even further we have utilized the CPALES system [29], [30], [31] developed by Dr. Yasinsac to analyze one of these new protocols, TLS [6].

The path that led us to the analysis of TLS began with the initial analysis of more traditional protocols with the CPALES system. We hoped to utilize this tool to discover unknown characteristics of selected protocols. In fact, we were able to discover some characteristics of a protocol not mentioned in the literature. This success convinced us to extend the system to handle the complex newer

protocols. This required modifications to the system so the hashing, secure functions, and key agreement as well as subprotocol interactions could be modeled. In particular the analysis of subprotocols also presented a problem with regards to an explosion in predicate size.

The main result of this paper is concerned with analysis of the TLS protocol. The modifications to made CPALES in order to obtain these results will also be discussed. Other topics discussed will provide background information that will explain the CPALES system of analysis and the TLS protocol. An interesting result gathered before work began on the analysis of TLS will also be mentioned. The analysis of TLS will demonstrate the ability of the system to model subprotocol interactions.

The paper will be organized as follows. An overview of the protocol analysis environment will be given in Section 2, which will include an explanation of weakest precondition reasoning, and BAN logic. Then we will go over analysis of several traditional protocols in Section 3. This will include an in depth description of the flaws found in the Kao Chow protocol [12]. We will then briefly describe the TLS protocol in Section 4. The CPAL implementation of the TLS protocol will be explained in Section 5. The problems posed by the analysis TLS protocol and the features implemented and used in CPALES to handle the more modern protocols are covered in section 6. Finally we will discuss the evaluation of TLS in section 7. Section 8 will give a conclusion and some thoughts raised by this work.

2. PROTOCOL ANALYSIS ENVIRONMENT

Yasinsac recently announced a workbench [31] for the evaluation of protocols. This tool is an integration of the inference construction methods with proof construction methods, utilizing BAN logic and weakest precondition reasoning respectively. The workbench has been used to evaluate the effects of protocols upon the principal's state, as well as their inferred beliefs.

The workbench includes a specification language, the weakest precondition reasoning engine, and an implementation of the PVS system to prove predicates produced from BAN specifications run through the reasoning engine. It uses the Cryptographic Protocol Analysis Language (CPAL) to explicitly specify the principal's actions in a protocol. The CPALES software performs "weakest precondition" (WP) evaluation on a CPAL specification. Weakest Precondition reasoning provides a precondition that represents the conditions necessary for the protocol to satisfy its goals. This provides a valuable proving environment for protocols, especially large commercial protocols like SSL, IKE and SET.

The CPALES system also allows the BAN specifications to be added to the protocol actions. CPALES can then create a predicate based on the BAN assumptions and assertions that is dependent upon the position of the BAN state-

ments in the protocol. This makes the resulting proof dependent upon the order in which beliefs are acquired and fixes problems that BAN can have with this [24]. The BAN predicate produced is then proved in the proving software PVS [22]. An overview of the whole analysis process is given in Figure 1.

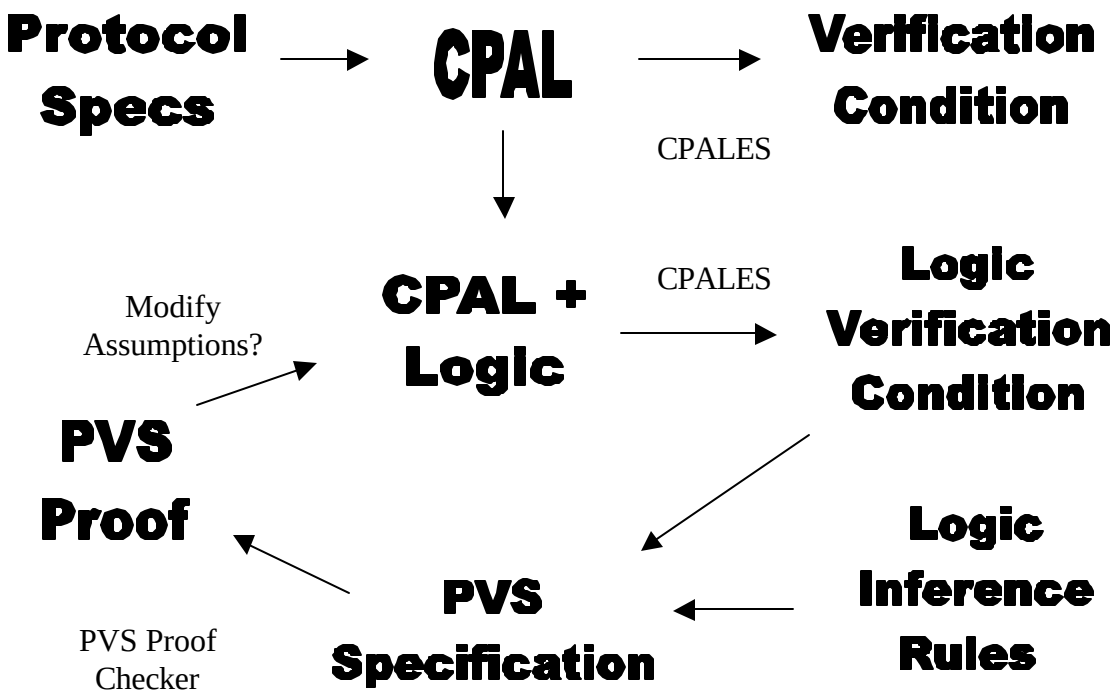

Figure 1. Protocol Analysis Flowchart

2.1 Weakest Precondition Reasoning

This analysis method utilizes predicate transformers for operations that define the weakest precondition for a post-condition [8]. For example, given the statement, X := Y, and the post-condition, X == Z, the predicate transformation for as-

signment would require the substitution of Y for all occurrences of X in the post-condition. This evaluation method is used to automatically verify goals regarding the comparison of data values in protocols. Consider the protocol described in Figure 2.

A -> B: Na
B -> A: {Na, A}K

Figure 2. ISO Sym. Key Two-Pass Unilateral Auth. Protocol

This protocol utilizes a nonce value Na to provide authentication. In order for the protocol to be correct, the key used by B to encrypt Na and the key used by A to decrypt would need to be the same. The above specification is typical for the description of protocols and is often termed standard notation. However, it is not amenable to meaningful analysis with weakest preconditions. By utilizing more detailed specifications, more detailed results may be garnered. In this case, the specification that A must decrypt the message received from B is not specified and is instead assumed. To handle this deficiency, the cryptographic protocol analysis language (CPAL) is utilized to describe protocols. The CPAL description of the above protocol is shown in Figure 3.

The => and <- operators utilized above described the send and receive operations respectively. Encryption and decryption of a term x is described with the e[x]K and d[x]K. The := symbol represents assignment, and the == symbol

represents equivalence. And finally the assert(x) statement is used to describe a goal or post-condition for the protocol. As you can see, the CPAL specification contains much more information. This extra detail allows ambiguities to be avoided. Of special note for analysis with weakest preconditions is the assertion. An assertion is required in order for a meaningful result to be produced. Table 1 describes some weakest precondition transformations [29]. Figure 4 shows the transformation of the predicate as the statements are evaluated. The CPAL specification is given on the left, while the condition that will allow the assertion to be satisfied, also termed the verification condition, is given on the left. Note that the statements are evaluated from the end to the beginning.

```
A:  => B(Na);
B: <-(nonce);
B: => A(e[<nonce, A>]K);
A: <-(msg);
A: (response, A) := d[msg]K;
A: assert(Na == response);
```

Figure 3. CPAL Specification for ISO

```
      (A.K == B.K)->(Na == <Na,A>.1) ∧
     (A.K != B.K)->(Na == d[e[<Na,A>]B.K]A.K.1)
A: => B(Na);                    (Na == d[e[<Na,A>]B.K]A.K.1)
B: <-(nonce);                   (Na == d[e[<Bq.nonce,A>]B.K]A.K.1)
B: => A(e[<nonce,A>]K);         (Na == d[e[<nonce,A>]B.K]A.K.1)
A: <-(msg);                     (Na == d[Aq.emsg]A.K.1)
A: (response,A) := d[msg]K;     (Na == d[msg]A.K.1)
A: assert(Na == response);      (Na == response)
```

Figure 4. Weakest Precondition Reasoning for ISO

2.2 Utilizing Logics with WP

BAN and other analysis logics have been utilized to describe the beliefs that may be acquired within a security protocol. Weakest precondition reasoning can enhance the ability of logics to analyze protocols. Most logics provide few mechanisms to handle the order in which the beliefs of the principals are reached. This was a weakness noticed by Snekkenes [24]. CPALES can rectify this problem through the use of weakest precondition transformations to determine how the ordering of assumes and asserts within the structure affect the verification condition.

Table 1. Selected Weakest Precondition Transformations

Operation	Statement	Result
Assign	wp(“A:x:=e;”,P(A.x)	P(A.e)
Sec Send	wp(“A:->B(msg);”,P(Bq.msg))	P(A.msg)
Receive	wp(“B:<-(msg);”,P(B.msg))	P(Bq.msg)
if - else	wp(if C {S1} else {S2;}”, P)	C->wp(“S1;”,P))&(~C->wp(“S2;”,P))
Stmt Cat	wp(“S1;S2”,P)	wp(S1,wp(“S2;”,P))
Assert	wp(“assert(Q);”,P)	(P and R)
Sym enc	wp(“d[e[X]k1]k2”,P(d[e[X]k1]k2)	(k1==k2)->P(X)&(k1!=k2)->P(old)

To include analysis logics in this system, we must include the presumptions and goals of the protocol. This is done by insertion of the propositions within the

CPAL specification. In this way, the specification of the actions can serve as a framework for the assumption and assertion. Figure 5 gives the CPAL-BAN logic specification for the protocol in Figure 3.

```
A:  assume(A.K == B.K);
A:  assume(X.believes(A.A, X.goodkey(A,K,B)));
A:  assume(X.believes(A.A, X.fresh(A.Na)));
A:  -> B(Na);
B:  <-(nonce);
B:  -> A(e[nonce,A]K);
A:  <-(msg);
A:  assume(X.sees(A.A, msg));
A:  (response,A) := d[msg]K;
A:  assert(Na == response);
A:  assume(X.believes(A.A, X.said(A.B, A.Na)) IMPLIES
      X.believes(A.A, X.said(A.B, (A.Na, goodkey(A.A,A.K,A.B))));
A:  assert(believes(A, believes(B, goodkey(A,K,B))));
```

Figure 5. CPAL BAN logic specification for ISO

All of the objects in the assumptions and assertions are associated with the data specified in the protocol actions. The combined specification can then be evaluated in the CPALES program to provide a verification condition. In this case it adds to the expressiveness of BAN logic by ordering the assumption and assertions. Typically these logics make no distinction upon where in the protocol particular beliefs are reached. This leads to an abstraction. A permutation of the message ordering can result in an insecure protocol. This complicates the analysis since an ordering of the propositions is not done in typical logical analysis.

Here is a demonstration of the ordering done by the CPALES system upon assumptions and assertions. To make this easier to understand, simple equality functions were used. To similar specifications are given in Figures 5 and 6. In the correctly specified example in Figure 5, first assert is the important one. The second assert is just there to keep the simplification of the proposition from removing the last assumption. The verification condition for the correct ordering is given in Figure 8 while the result from the incorrect specification is given in Figure 9.

The difference between the verification conditions is a result of the weakest precondition transformations. The important feature is the way in which assume statements affect the post-condition.

What this indicates is that the assumption made after the assertion cannot be used to prove the assertion. The /\ conjunction separate the eq(b,c) assumption from the eq(a,c) assertion preventing the proof. This can be seen in the sequent produced by PVS shown in Figure 10.

The second proof shown in Figure 11 is incomplete because only the first assumption is in the antecedent of the consequent eq(a,c). The nothing term was added to illustrate that the eq(b,c) formula is only in a position to prove nothing. PVS proofs are constructed by producing terms in the antecedent that match the terms found in the consequent. The second proof cannot be completed because the necessary formula is missing.

```
A: assume(A.eq(A.a,A.b));
A: assume(A.eq(A.b,A.c));
A: A.X := A.eq(A.a,A.c);
A: assert(A.X);
```

Figure 6. Correct Assumption Ordering

```
A: assume(A.eq(A.a,A.b));
A: A.X := A.eq(A.a ,A.c);
A: assert(A.X);
A: assume(A.eq(A.b,A.c));
A: assert(A.nothing);
```

Figure 7. Incorrect Assumption Ordering

```
((A.eq(A.a,A.c) or
not (A.eq(A.b,A.c))) or not (A.eq(A.a,A.b)))
```

Figure 8. WP result from Correct Ordering

```
(((A.nothing or not (A.eq(A.b,A.c)))
  and
A.eq(A.a,A.c)) or not (A.eq(A.a,A.b)))
```

Figure 9. WP result from Incorrect Ordering

```
{-1}  (eq(b, c))
{-2}  (eq(a, b))      // antecedent
  |-------
{1}   eq(a, c)        // consequent
this simplifies to:

goal :
{-1}  eq(a, b) AND eq(b, c) IMPLIES eq(a, c) // This is inst. lemma
[-2]  (eq(b, c))
[-3]  (eq(a, b))
  |-------
[1]   eq(a, c)
Rule? (split -1)
Splitting conjunctions,
this yields  3 sub goals:

goal.1 :
{-1}  eq(a, c)
[-2]  (eq(b, c))
[-3]  (eq(a, b))
  |-------
[1]   eq(a, c)
which is trivially true.

goal.2 :
[-1]  (eq(b, c))
[-2]  (eq(a, b))
  |-------
{1}   eq(a, b)
[2]   eq(a, c)
which is trivially true.

goal.3 :
[-1]  (eq(b, c))
[-2]  (eq(a, b))
  |-------
{1}   eq(b, c)
[2]   eq(a, c)
which is trivially true.
Q.E.D.
```

Figure 10. PVS Proof of Correct Ordering

```
  |-------
{1} (((nothing OR NOT (eq(b, c))) AND eq(a, c)) OR NOT (eq(a, b)))
Rule? (flatten)
Applying disjunctive simplification to flatten sequent,
this simplifies to:

goal :
{-1}  (eq(a, b))
  |-------
{1}   ((nothing OR NOT (eq(b, c))) AND eq(a, c))
Rule? (split +)
Splitting conjunctions,
this yields  2 sub goals:

goal.1 :
[-1]  (eq(a, b))
  |-------
{1}   (nothing OR NOT (eq(b, c)))
Rule? (flatten)
Postponing goal.1.

goal.1 :
{-1}  (eq(b, c))
[-2]  (eq(a, b))
  |-------
{1}   nothing
Rule? (postpone)

goal.2 :
[-1]  (eq(a, b))
  |-------
{1}   eq(a, c)
```

Figure 11. PVS Proof of Incorrect Ordering

2.3 PVS proofs of BAN logic

The specification of BAN logic assumptions and assertions within the CPAL specification allows the production of a logical predicate utilizing BAN logic terminology. This predicate can then be evaluated with BAN logic adapted for the PVS system. BAN logic utilizes logical postulates to enable the belief goals of principals to be proven. The postulates as utilized with PVS are given in functional notation. Typically the function possesses a descriptive name that can be used to explain its meaning. For instance, the function believes(P, goodkey(P ,K ,Q)) can be read as P believes goodkey(P ,K ,Q). Note as well that the goodkey function implies that the specified key is good for communication between the two principals. Figure 13 shows some BAN logic postulates in PVS syntax.

```
((((A.believes(A.A,A.believes(A.B,A.goodkey(A.A,A.K,A.B)))
  or
not ((not (X.believes(A.A,X.said(A.B,A.Na)))
  or
X.believes(A.A,X.said(A.B,X.cat(A.Na,X.goodkey(A.A,A.K,A.B)))))))
  or
not (X.sees(A.A,e[<A.Na,A>]A.K)))
  or
not (X.believes(A.A,X.fresh(A.Na))))
  or
not (X.believes(A.A,X.goodkey(A.A,A.K,A.B))))
```

Figure 12. CPALES BAN Verification Condition for ISO

Once the verification condition has been developed as shown in Figure 12, it can be adapted for use in a theorem proving system like PVS. BAN like logics have typically been proven by hand. The theorem proving system PVS, takes the ordered verification condition to produce a sequent. By utilizing PVS, large verification conditions can be processed, ordering of assumptions can be seamlessly incorporated into proofs, and some automation is possible.

```
msg_m1:    AXIOM FORALL (P,Q,KPQ,X: bool):
  believes(P,(goodkey(P,KPQ,Q))) AND sees(P,e(X,KPQ)) IMPLIES
  believes(P,said(Q,X))
msg_m2: AXIOM  FORALL (P,Q,PUB_Q, PUB_Q_INV, X:bool):
  believes(P,pubkey(Q,PUB_Q)) and sees(P,e(X,PUB_Q_INV))
IMPLIES believes(P,said(Q,X))
N_verif: AXIOM  FORALL (P,Q,X: bool):
  believes(P,fresh(X)) and believes(P,said(Q,X)) IMPLIES
  believes(P,believes(Q,X))
juris: AXIOM  FORALL (P,Q,X: bool):
  believes(P,controls(Q,X)) and believes(P,believes(Q,X)) IMPLIES
believes(P,X)
sees1: AXIOM  FORALL (P,X,Y: bool): sees(P,cat(X,Y)) IMPLIES
  sees(P,X) and sees(P,Y)
```

Figure 13. BAN Logical Postulates for PVS

3. TRADITIONAL PROTOCOL EVALUATION

The major work of this paper is concerned with the analysis of TLS. This was not the initial focus of the work. This work started as a survey of security protocols with the CPALES system in order to learn more about its capabilities as an analysis tool and to hopefully discover unknown characteristics of the analyzed protocols. A couple of flaws were discovered in one of these protocols. The process of discovering this flaw will be used to demonstrate the CPALES system and to highlight some of its abilities.

The protocols that we chose for this analysis were pulled from a set of fifty-one protocols described by Clark and Jacob [5]. The draft by Clark and Jacob described the protocols and the attacks they were susceptible to from their own informal analysis and examination of the literature. This information and a report [4] given by Brackin were used to aid in specifying the protocols and in the comparison of results. Our analysis of the protocols was not comprehensive. We specified thirty-nine of the protocols in CPAL using the CPALES system help with the specifications. Nine of these protocols were also given BAN logic specifications. Proofs using the PVS system were performed on the Carlsen protocol and the second Kao Chow protocol.

Upon surveying the Brackin report, it was noticed that Brackin's system had trouble with some of the protocols. So these protocols were analyzed to see if similar problems were encountered with our analysis. In the case of the Neumann and Stubblebine [20], and KLS protocol [17], both of these protocols are vulnerable to external interleaved reflected replay attacks as classified by Syverson [27]. In this type of attack, the intruder starts a new protocol run after the attacked protocol starts and uses the two sets of messages to construct a successful attack.

In general, logics are not adept at detecting these vulnerabilities. BAN in particular cannot find this type of attack since one of its premises is that a principal can identify its own messages. If it cannot, then BAN will not find an attack using this weakness. Other logics like GNY can represent some similar types of attack. However, all logics of this type encounter difficulty in eliciting their presence. The problem is that the realization of a message's potential to be replayed must be recognized by the analyzer. The logic itself provides no hints.

The Neumann Stubblebine protocol is also subject to a causal consistency attack [26]. This type of attack is also not detected with logical analysis. The Brackin tool assumes that a type check finds these attacks. Again this type of attack cannot even be represented in BAN logic. Syverson suggested a way to represent the attack, however the flaw is not amenable to discovery with this suggestion. These limitations to the BAN system prevented us from proving the attacks shown in Clark and Jacob for the Neumann Stubblebine and the KLS protocols.

The next protocols to be evaluated were the three Kao-Chow protocols [12]. These were chosen since they were purported to fix the problems encountered by KLS and Neumann Stubblebine protocols. Problems with the description given by Clark and Jacob necessitated a reference to the original paper. The first protocol is recognized by Kao and Chow to be susceptible to the same attack as the Needham Schroeder symmetric key protocol [7]. To combat this, they included a second key in the protocol to be used for encryption in the protocol while the session key would be used for encryption upon completion of the protocol. A description of the protocol in standard notation is given in Figure 14. They suggested that the second key (Kt) should be considered secure.

A->S: A, B, Na
S->B: {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
B->A: {A, B, Na, Kab, Kt}Kas, {Na, Kab}Kt, Nb
A->B: {Nb, Kab}Kt

Figure 14. Second Kao Chow Protocol

They reasoned that Kt is only used with a very small amount of material making it difficult to crypto-analyze. In addition, the material it encrypts is very random in nature since Na should be a new random number and Kab should not be guessable. Clark and Jacob do not explain these details. Knowing what the author's intended, I tried to analyze the protocol with the assumption that Kt was secure. I did this by assuming that B believes that the key he receives in the fifth

field of the ticket from the server is good. The full CPAL-BAN logic specification is begun in Figure 15 and finished in Figure 16. The BAN logic goals used in the specification were given by Kao and Chow in their specification for the protocol.

The second Kao Chow protocol does not meet all of its goals. The above assumptions are enough to prove the given assertions only because invalid assumptions were added. The first invalid assumption is given as the 13th statement. The message received from the server (e[<A,B,Na,Kab,Kt>]Kbs) has nothing in it that Bob considers fresh. Therefore the idealized message X.goodkey(B.A,B.Kab,B.B),X.goodkey(B.A,B.Kt,B.B) cannot have the nonce verification rule applied to it. The nonce verification rule allows a principal who receives a message to believe that a principal who said the message believes what is in the message if the receiver believes the message is "fresh."

A message can be considered fresh if it contains something that a principal knows was created during the current run of the protocol. Since this postulate may not be applied, Bob cannot acquire any belief in the goodness of Kab at this time. The unsupported assumption:

believes(B.B,fresh(cat(goodkey(B.A,B.Kab,B.B),goodkey(B.A,B.Kt,B.B)

allows B to eventually believe that Kab and Kt are goodkeys.

In a typical logical analysis it would not matter where an assertion was made in a protocol. With CPALES analysis, it is important to find the earliest time an assertion can be made to take advantage of the ordering of the predicate done by the system. In this case, Bob needs to have a belief in Kab in order for Alice to acquire belief in Bob's belief.

```
  SETUP, not attacked
S: => A(<B, Kas>);
A: <-(msga);
A: (B, Kas) := msga;
A: assume(X.believes(A.A, X.goodkey(A.A, A.Kas, A.S)));
S: => B(<B, Kbs>);
B: <-(msgb);
B: (B, Kbs) := msgb;
B: assume(X.believes(B.B, X.goodkey(B.B, B.Kbs, B.S)));

START
1. A: assume(X.believes(A.A, X.fresh(A.Na)));
A: => S(<A, B, Na>);
S: <-(msg1);
S: (A, B, Na) := msg1;

5. S: => B(<e[<A, B, Na, Kab, Kt>]Kas, e[<A, B, Na, Kab, Kt>]Kbs>);
B: <-(msg2);
B: (tickAS, tickBS) := msg2;
B: assume(X.sees(B.B, B.tickBS));
B: (A, B', Na, Kab, Kt) := d[tickBS]Kbs;
10. B: assume(X.believes(B.B,X.controls(B.S,
      X.goodkey(B.A,B.Kab,B.B))));
11. B: assume(X.believes(B.B,X.controls(B.S,X.goodkey(B.A,B.Kt,B.B))));
12. B: assume(X.believes(B.B,X.said(B.S,
       X.cat(B.A,X.cat(B.B,X.cat(B.Na,X.cat(B.Kab,B.Kt)))))) >>
       X.believes(B.B,X.said(B.S,
       X.cat(X.goodkey(B.A,B.Kab,B.B),X.goodkey(B.A,B.Kt,B.B)))));
13. B: assume(X.believes(B.B, X.fresh(X.cat(X.goodkey(B.A,B.Kab,B.B),
         X.goodkey(B.A,B.Kt,B.B))))); -- *** INVALID belief **
14. B: assert(B == B');
15. B: assert(believes(B, goodkey(A, Kab, B)));
B: assume(X.believes(B.B, X.goodkey(B.A, B.Kt, B.B)));

B: assume(X.believes(B.B, X.fresh(B.Nb)));
```

Figure 15. CPAL-BAN Spec. for 2nd Kao Chow Protocol

```
B: => A(<tickAS, e[<Na, Kab>]Kt, Nb>);
A: <-(msg3);
20. A: (tickAS, chalA, Nb) := msg3;
A: assume(X.sees(A.A, A.tickAS));
A: assume(X.sees(A.A, A.chalA));
A: (A', B', Na', Kab, Kt) := d[tickAS]Kas;
A: assert(<A, B, Na> == <A', B', Na'>);
25. A: assume(X.believes(A.A, X.controls(A.S,
        X.goodkey(A.A, A.Kab, A.B))));
A: assume(X.believes(A.A, X.controls(A.S,
        X.goodkey(A.A, A.Kt, A.B))));
A: assume(X.believes(A.A,X.said(A.S,
        X.cat(A.A,X.cat(A.B,X.cat(A.Na,X.cat(A.Kab,A.Kt)))))) >>
        X.believes(A.A, X.said(A.S,
        X.cat(A.Na,
        X.cat(X.goodkey(A.A, A.Kab, A.B),
              X.goodkey(A.A,A.Kt,A.B))))));
A: assert(believes(A, goodkey(A, Kt, B)));
A: assert(believes(A, goodkey(A, Kab,B)));
30. A: (Na'', Kab') := d[chalA]Kt;
A: assert(<Na,Kab> == <Na'',Kab'>);
32. A: assume(X.believes(A.A, X.said(A.B, X.cat(A.Na,A.Kab))) >>
       X.believes(A.A, X.said(A.B, X.cat(A.Na,
       X.goodkey(A.A,A.Kab,A.B)))));
A: assert(believes(A, believes(B, goodkey(A, Kab, B))));

A: => B(e[<Nb, Kab>]Kt);
35. B: <-(msg4);
B: assume(X.sees(B.B, B.msg4));
B: Nb_Kab := d[msg4]Kt;
B: assert(Nb_Kab ==  <Nb, Kab>);
B: assume(X.believes(B.B,X.controls(B.A,
   X.fresh(X.goodkey(B.A,B.Kab,B.B)))));
B: assume(X.believes(B.B,X.said(B.A,X.cat(B.Nb,B.Kab))) >>
        X.believes(B.B,X.said(B.A,
        X.cat(B.Nb,X.cat(X.goodkey(B.A,B.Kab,B.B),
        X.fresh(X.goodkey(B.A,B.Kab,B.B)))))));
40. B: assert(believes(B, believes(A, goodkey(A, Kab, B))));
```

Figure 16. CPAL-BAN Spec. for 2nd Kao Chow protocol (cont.)

Bob's belief in Kab should be transferred in the third message given on line eighteen. In a traditional BAN analysis, a belief cannot be asserted within the protocol. In our specification, we make the assertion on line fifteen, that "B: assert(believes(B, goodkey(A, Kab, B)));." This means that Bob believes that Kab is a good key for Alice and Bob to use. The assertion is made before Bob encrypts Kab with Kt. This encryption of Kab with the good key Kt allows Alice to eventually assume that Bob believes Kab is a good key. If in the analysis Bob's assertion that it believes Kab is a "goodkey" was not made until after the fourth message of the protocol, then there would be no need for the invalid assumption of the 13th statement. Thus the ordering of the Ban logic goals has an effect upon the analysis. If the assertion is made at the end of the protocol, then the information available from the last message from Alice to Bob may also be utilized to satisfy the assertion.

A problem still remains with this protocol. This is discovered if we remove the requirement that Alice believe that Bob believes the key Kab is good. This should also mean that Bob could drop the invalid assumption made with the thirteenth statement regarding the freshness of the message Bob receives from the server. Actually, Bob still needs some help to acquire the belief that Kab is a goodkey. This may be achieved with a somewhat weaker, but still invalid assumption regarding Bob's trust in Alice.

When Bob receives the fourth message from Alice, it contains Bob's nonce and the session key Kab encrypted with Kt. Bob already believes that Kt is a "goodkey" for communication between him and Alice. However, he has not yet

acquired any confidence in Kab without the invalid assumption. Typically, part of the process of transferring a belief between principals requires that the receiving principal believes that the sending principal has jurisdiction over that particular belief. For instance, it is typical for a key distribution center to have jurisdiction over the "goodness" of keys. As was stated earlier, Bob is prevented from believing that the server believes the key is good because Bob does not believe anything in the message from the server is fresh. So only the last message from Alice can give Bob the beliefs necessary to acquire belief in the goodness of Kab. One solution is for Bob to trust Alice to provide a "goodkey." In this case, there is no reason for the server in the protocol. This is too strong of an assumption to make.

Bob's lack of belief in the freshness from the server is the root of this problem. A weaker assumption can be made to solve this proof. That is that Alice says the key is fresh by sending to Bob in the fourth message. In order for this assumption to make a difference, Bob must also trust Alice to have jurisdiction over the freshness of the message that Kab is good. This is specified in the 38th and 39th statements shown in Figure 16. The first statement assumes that Bob can utilize Alice as a server for the freshness of Kab. Typically, protocols try to avoid giving principals this kind of power. This is the second invalid assumption of the protocol. After the fourth message, Bob will be able to acquire belief in the "goodness" of Kab if it trusts Alice to insure its freshness.

The authors (Kao & Chow) appeared to realize that Bob could not believe Kab until Alice endorsed the freshness of Kab when they said Kt is "used by Bob

to encrypt Na and Kab to tell Alice that Bob temporarily trusts both keys Kab and Kt." Does this mean that this temporary belief should convey a permanent belief in Alice? Since Alice does not receive any more messages from Bob, there can be no further confirmation provided by the second message. The temporary belief is in fact permanent. For the protocol to reach its goal of Alice acquiring a second level belief in the goodness of Kab, requires a fifth message from Bob to Alice for Alice to acquire this belief. There is no fifth message, so the protocol fails to achieve this goal.

These vulnerabilities are real, if somewhat weak. However, both could lead to attacks on the protocol. Now that these weaknesses have been revealed by our analysis with CPALES with BAN, we can consider some possible attacks to exploit these two weaknesses. The second vulnerability arises from Bob's trust of Alice to guarantee the freshness of Kab. This means that Alice could endorse an old Kab without Bob knowing that the key was old. The initial weakness arises from the fact that Alice acquires the invalid belief that Bob believes Kab is a goodkey. Believing that Bob will accept the encrypted message Alice will feel safe to start a session. Then Alice may send a long, expensive encryption to Bob after she sends the fourth message. Bob will not trust the contents of the message if the confirmation provided by Alice in message four is blocked. This attack is detailed in Figure 17.

The second attack shown in Figure 18 allows Alice to create an insecure session. Alice does not need to retransmit the information in order to reveal the information transmitted in the session. She could choose an old key that had been

broken by a third party. Then the session itself would reveal the information, reducing the risk that an additional outside message would reveal the untrustworthiness of Alice. The initial message could also be sent, if it was felt the blocking of the second message from the server to Bob was less noticeable than Alice not sending a message to the server in the first place.

1) A=>S: A, B, Na
2) S => B: {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
3)B => A: {A, B, Na, Kab}Kas, {Na, Kab}Kt
4)A => B: (Blocked)
4a)A => B: {long message}Kab

Figure 17. 1st attack on Kao Cow 2

1)Skipped
2)S(A) => B: {A, B, Na', Kab, Kt}Kas, {A, B, Na', Kab, Kt}Kbs
3)B => A: {A, B, Na', Kab}Kas, {Na', Kab}Kt
4)A => B: {Nb, Kab}Kt

Figure 18. 2nd attack on Kao Chow 2

The Brackin report and the Clark and Jacob review do not expose these flaws of the Kao Chow protocols. At least partly Brackin misses this feature of the protocol since his specification does not allow a key to be considered immune to an old session key attack. The ability to make that assumption allowed other weaknesses in the protocol to be discovered. The position in which beliefs were acquired was also shown to be important through the use of weakest pre-

condition reasoning to arrange the verification condition. With this ordering, the BAN logic system was enhanced by the forcing the specification to be ordered. This allowed the reviewer to examine the importance of order in the proof.

4. TRANSPORT LAYER SECURITY STANDARD

The Transport Layer Security standard (TLS) [6] is a refinement of the Secure Socket Layer (SSL). According to the TLS specification, this standard was produced by the Internet Engineering Task Force (IETF) to provide "privacy and data integrity between two communicating applications." Like the SSL protocol it was based on, TLS provides a suite of protocols utilizing various cryptographic functions in different combinations. This allows users of the standard the flexibility to choose a solution from this set that best fits their needs or capabilities. The danger here is that interactions between the subprotocols could lead to new insecurities.

Until the recently, most security protocols had only one sequence of messages that would lead to a satisfied condition. This was true for protocols proposed in the literature and early implemented protocols like Kerberos. More recent protocols such as SSL have proposed a suite of protocols that could be agreed upon by two principals. This allowed different key systems, levels of security and levels of authentication to be agreed upon by the two principals. These choices were added to increase the functionality of the system. They also make the specifications complex in comparison to the older protocols.

Figure 19 describes a limited protocol suite. In this case, Alice and Bob can choose from three subprotocols to run their session. Once a principal has chosen a subprotocol he cannot switch to another subprotocol. Since both principals decide independently which subprotocol to run, they may decide to run subprotocols, which were not designed to interact. In the TLS protocol, the subprotocols to be run are negotiated in the clear. Therefore, interference with these negotiation messages can cause inappropriate protocol interactions. Later in the paper, we will show how this leads to a flaw in the TLS protocol.

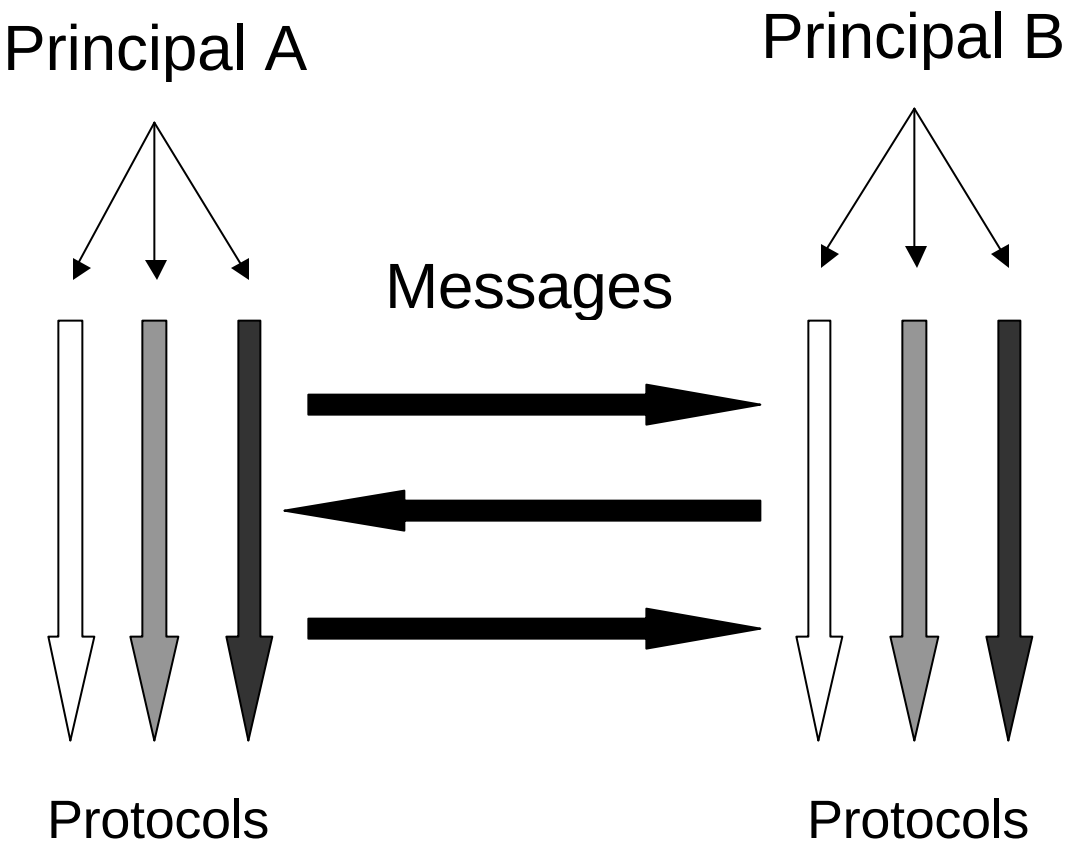

Figure 19. Protocol Suites and Subprotocol Interactions

The TLS protocol is a proposed protocol standard of the IETF task force. The "… goal of the TLS protocol is to provide privacy and data integrity between two communicating applications." Two layers, the record protocol and the handshake protocol, make up the TLS protocol. The record protocol uses an encryption key negotiated by handshake protocol to provide security. It also provides services to handle compression, fragmentation, and message authentication. The handshake protocol handles negotiation between the two parties to determine the characteristics of the session including, compression methods, encryption methods, the protocol version, the session identifier, peer certificate, and whether it is the resumption of an old session.

These negotiations are handled by subprotocols that make up TLS, and use standard practices such as public key encryption, public key signatures, Diffie-Hellman key agreement, certificates, nonces, hashing and other tools to achieve such goals as authentication, privacy and key exchange. Each negotiation session will result in the choice of and use of one of these subprotocols, which are termed *ciphersuites*. It is these tools that make up the composition of the subprotocols.

Key establishment protocols typically allow two principals to securely negotiate a shared key for communication session. This is one of the features of TLS. In order to make TLS as flexible as possible, the protocol also allows the secure negotiation of other security and communication mechanisms. These include the key exchange algorithms, the signature schemes, the accepted certificates, the encryption algorithm, the hashing function, and the compression algorithm to be

used. The implementation of these different choices can be characterized by a set of subprotocols, which encompass these choices.

The TLS specification gives twenty-eight different ciphersuite definitions. In TLS a ciphersuite defines the key exchange method, a signature method, the authentication, the encryption algorithm to be used for the key that is established, and the hashing function used. Typically, the server or the client can do authentication. The ciphersuite may specify whether the server will be authenticated. The server may also request authentication of the client after the ciphersuite has been negotiated. The form of the certificate that provides authentication must be determined. Thus this adds even more possibilities.

5. IMPLEMENTATION OF TLS IN CPAL

In our implementation of TLS in CPAL, we made certain choices regarding the representation of the protocol. These choices were made to create a manageable specification and to eliminate the details that had no effect upon the security of the protocol given our assumptions. In this section we will how TLS was specified, the reasoning behind our implementation. To get a general idea of the subprotocols that are represented look at table 2. For the most part, the subprotocols utilize different cryptographic algorithms, but also require different sets of actions to describe that protocol. The Resume protocol utilizes an old session identifier and a shared secret between the two parties to resume the session. In the Anonymous protocol, neither the server nor the client provides a certificate to assure their identities. This leaves the anonymous protocol vulnerable to man in the middle attacks.

The DHE protocol utilizes Diffie-Helman key agreement to exchange keys. DHE key exchange uses new Diffie Helman parameters for each session, whereas DH key exchange utilizes fixed parameters that are the same for each session and produce the same value for each connection between the same principals. The DHE_CA protocol requires authentication of the client as well as

the server. This is also the case for other protocols with the CA string in their names.

Table 2. Subprotocol Attributes

Subprotocol Name	Key Exchange	Authentication	Public Key Use
Resume	None	Old Secret	No
Anonymous	DH	None	No
DHE_CA	DH ephemeral	Both	Signatures
DHE	DH ephemeral	Server Only	Signatures
DH_CA	DH fixed	Both	Signatures
DH	DH fixed	Server Only	Signatures
RSA_EXP_LS_CA	PK w/short enc. key	Both	Enc & Sigs w/ long PK
RSA_EXP_LS	PK w/short enc. key	Sever Only	Enc & Sigs w/ long PK
RSA_EXP_CA	PK w/short enc. key	Both	Enc & Sigs w/ short K
RSA_EXP	PK w/short enc. key	Sever Only	Enc & Sigs w/ short K
RSA_CA	PK w/long key	Both	Enc & Sig w/ long key
RSA	PK w/long key	Sever Only	Enc & Sig w/ long key

The RSA protocols utilize public key encryption for the client to send a session key to the server. The differences between the protocols entail whether the client is authenticated, whether the encryption key is shorter than 512 bits so that it would be legal for export by the old standards, and whether it used a signature key longer than 512 bits. When TLS was specified, legal restrictions prevented

asymmetric encryption keys longer than 512 bits. However, they did not prevent signature keys longer than 512 bits. Therefore, the TLS protocol allows the client and server to utilize the longer key for authentication, while a temporary shorter public encryption key signed by the server is used to encrypt the secret.

Combined, these protocols describe the TLS protocol. In order to determine which protocol actions are executed as the principals proceed through the negotiation, conditional branches are needed to describe a comprehensive implementation of the specification. Appendix A describes the general flow of the protocol in an extended standard notation. Conditional branches and goto statements are utilized to control which protocol is executed. The important characteristic of this specification versus other protocols is the requirement for conditional branches.

If you examine the specification in appendix A, you should notice that there are conditional branches throughout the protocol. This is not the only way the protocol could be implemented. Another way to describe the protocol would be to move all of the tests to the beginning of the protocol when the cipher-suite is selected. Once the selection has been made and a subprotocol is chosen, no more choices need to be made regarding which subprotocol is being executed. In terms of network programming, once a subprotocol was selected a thread could be spawned to handle that connection. Then that thread would execute the appropriate actions for that cipher-suite. No more references to which ciphersuite was selected need to be referenced by the program's control.

The CPAL representation of TLS described in appendix B is different from that given in appendix A. Looking at the standard notation for TLS, you may not realize that any control decisions made by one principal can only affect the other through a message. In the standard notation, only one principal would perform a conditional test. In the CPAL notation, each principal tests its state to determine which actions to execute when a message is received. Notice also that a new set of conditional branches must be specified by each principal each time a message is received even though they may have already done these branches before. This repetition is required since CPAL cannot specify blocking or execution functions.

These restrictions on the capabilities of CPAL do not prevent the representation of TLS. The standard does not specify many implementation details regarding how principals are to control their sessions. They can use a single, or a multi-threaded one if they wish. The fact that CPAL cannot model these different implementations is not a problem. The important characteristic of the CPAL specification is that correctly models the TLS standard and not a particular implementation of it.

A cursory examination of the CPAL specification for TLS may indicate to the examiner that the specification allows switching between the subprotocols after the cipher-suite has been determined. If it did, the CPAL specification would not match the TLS standard. This is not the case. The variable that determines the subprotocol characteristics is not modified after selection. Therefore, if all of the conditionals depend on this ciphersuite value, and the value is not modified later

in the protocol, the branches to be executed have been determined even though those branches may have not been reached. This makes the CPAL specification logically equivalent to an implementation that spawned a thread to run a particular ciphersuite.

In the general CPAL TLS specification, each message except for the initial one from the client requires a conditional test to determine the message composition. The first conditional encountered is utilized by the server to determine if the proposed session identifier (SID) is from a valid old session. If it is, then that session identifier will be sent back to the client otherwise a new one will be generated. The initial message sent from the client also specifies a list of acceptable ciphersuites (CSL). Our specification does not model the effects of this list. Instead the server is free to choose any cipher suite. Therefore, the server is not limited in his choices. For this general specification, this should not affect the rest of the protocol, since the analysis should examine all of the subprotocols. Therefore, it can be considered that the client sent a list containing all of the ciphersuites. In order to model a partial list, a different specification would need to be made.

The ciphersuite specifies a number of different subprotocols. The conditions needed to specify the different subprotocols utilize a set of values. These values are generated by a specific function that models the extraction of a particular ciphersuite characteristic from the CS value. This allows one value to be used to create the set of conditions. Since functions of the same name are identical for all principals, this can be used to create identical values for the two principals.

As noted previously, the different subprotocols of TLS are implemented in the specification utilizing the if/else branch of the CPAL language. Nested branches are utilized to group similar subprotocols in the same block. They are constructed in a fashion so that only one subprotocol may be entered in the block. Each subprotocol describes the message it will send. When the block is left, the message is sent. The other principal receives the message and the next conditional block is entered. This continues until the end of the protocol.

The protocol described leaves out some details of the TLS specification. Certain capabilities of protocols do not affect the analysis of the protocol. For example, the TLS specification can indicate that a subprotocol will use DSS or RSA signing. Although these signing mechanism utilize different algorithms, use of one over the other does not change any of the operations specified by CPAL. In general protocol analysis considers that all cryptographic algorithms are secure. Therefore, there is no effect upon our analysis if RSA signing is used instead of DSS signing or vice versa. These and other choices were left out of the specification in CPAL since they would just be adding different names to the same operations. If there were known interactions between various cryptographic algorithms, then the specification would need to algorithms in question. This reasoning is also used to exclude representations of MD5 vs. SHA hashing, DES vs. (DES_CBC or 3DES) and others.

The different digital encryption algorithms are also precluded from representation since their only affect on the handshake protocol is the size and format of the key. No encryption is performed with DES keys in the handshake protocol.

There is a caveat to this. Some algorithms provide more security than others. One argument for the inclusion of choices that provide different levels of security is if those choices affect the operation of the protocol being analyzed. The RSA_EXP (w/short signature keys) and RSA cases have been separated in the specification even though their implementations are identical except for key size. This would not appear to be meaningful however. One could just as easily describe the RSA protocol as using long and short encryption keys.

6. MODIFICATIONS TO CPALES

Implemented protocols such as TLS and IKE are based on the same concepts as the simpler protocols listed by Clark and Jacob. These protocols typically implement a suite of protocols so that a range of features may be chosen for each session of the protocol. However since most analysis tools have only examined sequential protocols, analysis of these newer protocols has been restricted to analysis of the subprotocols. This analysis is certainly useful, however formal analysis of the entire protocol is often left wanting. Without consideration of interactions between the subprotocols, the reviewers have not formally analyzed the entire protocol. Authors utilizing formal methods have looked at the protocols suites SSL 3.0 [19] and TLS [23]. However, in both of these cases, the protocol was not treated as a whole. Although analysis of IKE by Meadows' [18] did examine the whole protocol, the system uses different methods with different capabilities.

The combination of the subprotocols can make the analysis considerably more complicated. CPAL was developed with the functionality to handle this type of protocol. Specifically, CPAL contains if/else statements that allow branching between the different states of the protocol. Unfortunately, although the ability to

handle this feature was implemented in CPALES, it was not utilized in the initial work. One reason being that most analysis had of course not included these types of protocols before. Traditional protocols analysis has focused on simple protocols. These protocols use no branching and are often quite short. Of course, even these small protocols have proven to pose a considerable challenge. With the advent of SSL and other protocol suites, the need for formal analysis tools capable of analyzing these systems has arisen.

The specific threat to these protocol suites is the potential for attacks that interleave the subprotocols. A single protocol is susceptible to attacks that interleave messages from one subprotocol with another. In a protocol composed of subprotocols it is possible that the subprotocol will be susceptible to attacks using messages from different protocols as well as its own. Kelsey and Schneier have argued that given the same key material, a protocol may always be constructed which can be used to subvert the security of a given protocol [15].

Most protocol suites are designed to allow different levels of security in a session. This allows the possibility that a weak protocol may be switched for a stronger protocol if strong safeguards are not in place. One problem of an earlier version of SSL, the precursor to TLS, was its potential to be tricked into running an earlier version of the protocol even though both parties could use the later and stronger protocol. This type of attack would also be applicable to TLS since it allows the specification of the protocol version to be utilized.

A large variety of cryptographic functions and algorithms are implemented in TLS. Earlier work with the CPALES system had focused on the more traditional

protocols. In these protocols, the ability to handle symmetric key encryption, public key encryption was usually enough. Newer protocols have added features such as hashing, key agreement, and the aforementioned subprotocols. All of these features are utilized by TLS. In order to analyze TLS and other protocols like it, CPALES needed to be extended.

6.1 Advanced Cryptographic Algorithms

CPALES does not implement function passing. This makes it difficult to compare hashes for authentication and integrity purposes. The comparison of two hashes would not be found identical by CPALES since one principal could not send a hash function to another. Assuming the equivalence of two functions was also problematical since CPAL syntax required functions to specify their arguments. Therefore, any assumption regarding a function would only be valid for the arguments chosen. A simple solution utilized the encryption operation. This is possible since a secure hash function has similar properties to one-way encryption. Both produce a result that should not be invertible. The encryption operation is standard between participants and the operations can be compared and found to be equal if the operations were done with the same key. So to simulate a hash one does the operation e[message]hash. Then if another principal wants to do the same operation, you just send the hash key to that principal and he can do the same operation. A more accurate implementation utilized public key encryption without a defined decryption key. This way, a secure one-way hash could be modeled.

The above solution is far from ideal. It has the potential to confuse a hashing operation with public key encryption. Further examination of the problem indicated that instead of sending functions between principles, a simpler solution would make all functions identical across the principal's environments.

```
pubA := f(prime, gen, secA)
pubB := f(prime, gen, secB)
f(prime, pubA, secB) == f(prime, pubB, secA)
f(prime, f(prime, gen, secA), secB) == f(prime, f(prime, gen, secB), secA)
```

Figure 20. Diffie-Hellman Key Agreement

This does not accurately model reality, yet it is no weaker than assuming encryption operations are identical across boundaries. So to handle this, the prefix names for the function identifiers were changed in the CPALES code. Instead of assigning functions the prefix for the principal they were created in, all prefixes received the same string as their id. Now when functions with the same name are used in different environments, their results can be the same. This removes the need to utilize the encryption operators. A drawback or an advantage of this system is that obfuscation can no longer be modeled as a security method. This could be considered an advantage since obfuscation is generally derided as a means of providing security.

The implementation of key agreement at first appeared to be difficult. If it were implemented like the encryption operation, it would have required changes

in most of the sections of the CPALES code. A simpler solution utilized the capability for comparison of functions. For it to work, the relationships shown if Figure 20 had to be followed

The solution used a special case for comparison of functions with the name “dhy”. In this case, special positional considerations were needed to model the characteristics of the Diffie-Hellman functions. These match the comparisons shown in Figure 20.

6.2 Improvement to Composite Assignment

One enhancement dealt with how mismatched composite assignments were handled (e.g. (A,B,C) := <X,Y>;). In a linear protocol, this case should not occur in a correctly functioning protocol that is not being attacked. The original CPALES program treated this as a specification problem and would terminate the analysis when this occurred. This was helpful for finding incorrect specifications. However, when subprotocols were specified, this feature had to be changed. Along different branches, messages would be created with different numbers of elements. Typically two principals would reach an agreed upon state so that a message sent for one state would be received in a matching state for the receiving principal. However, the weakest precondition system does not recognize if these states are matching to determine whether the predicate should be modified, instead, it modifies the predicate so that if they are not matching, the resulting predicate will be trivially true. The result of this is that, during the processing of the protocol statements, assignment of different sized lists cannot be avoided.

A new way to handle this situation needed to be devised. The specification fragment given in Figure 21 illustrates an occurrence of this problem.

```
B: =>A(<cond,a,b,c>);
A: <-(list);
A:(cond,a,b,c) := list;
A: if (cond) then {msg := <a,b>;} else {msg := <a,b,c>;}
A: =>B(msg);
B: <-(msg);
B: if (cond) then {
       (x,y) := msg;
       assert(<x,y> == <a,b>);}
   else {
       (x,y,z) := msg;
       assert(<x,y,z> == <a,b,c>);}
```

Figure 21. List Assignments in the if/else Statement

The difficulty is that msg may contain a two or three element list. Therefore, (x,y,z) may be assigned to (a,b) in the system. The behavior of a system when this situation is encountered needs to be considered. If this is not an error, then once the entire predicate is simplified, then mismatched conditionals should cause the affected predicate to be trivially true, e.g. ~P \/ P => TRUE. When this was an error it needed to be noticed. In this case the affected predicate would not be trivially true. A naming system was devised for these values. The values were given unique names so that they would not be removed from the predicate by simplification. In an actual implementation of the protocol in an application, the behavior of the protocol would depend on whether the value z was overwrit-

ten with an unspecified piece of memory, kept the same, or an error condition would develop. Our solution does not allow the value to keep the same value it had before the assignment, however, in all likelihood; a protocol should not exhibit this type of behavior, and so finding the irreducible comparison in the simplified predicate should indicate a problem with the protocol.

6.3 CPAL branching limitations and Simplification improvements

When the specification of TLS was begun, the incomplete protocol was tested as the protocol grew. It was during this time that several challenges of analyzing collections of protocols with CPALES were encountered.

One limitation of CPALES encountered was inability to model messages sent inside an “if/else” block. The implementation as it stands cannot handle this without having misaligned send and receives. The protocol shown in Figure 22 causes an imbalance in the send/receive queue utilized by the CPALES system to keep track of the alignment between send and receive operations. The send/receive queue is actually not a queue at all. It is a numbering system that ensures send and receive operations are not applied out of order. In this system, each send, must be followed by one receive operation. The above specification would cause a problem since it would find two sends before a receive was encountered. The system does not discern that only one message will be sent. Figure 23 displays another complication.

S may or may not have sent a message. However, the system will still “send” the message even though f(x) may not be true. Therefore, C may then

receive a message that was never sent. To avoid adding a potential complicated feature, it was decided to not use send and receive operations within if/else blocks. Instead messages are assembled inside conditional blocks, to be sent when the conditional block is left. This restricts the syntax of send and receives operations. However, it should still allow enough flexibility for protocols to be described in a manner equivalent to the proscribed forms.

```
C: => S(X);
S: <-(X);
S: if (f(X)) then {=>C(Y);} else {=>C(Z);}
C: <-(msg);
```

Figure 22. Misaligned Send/Receive Example: 1

```
C: => S(X);
S: <-(X);
S: if (f(X)) then {=>C(Y);}
C: <-(msg);
```

Figure 23. Misaligned Send/Receive Example 2

Another problem encountered with the analysis of subprotocols deals with the simplification of the verification condition. In this case, the reduction routines in place did not have power to perform the simplifications necessary for analysis of TLS with CPALES. Improved simplification routines needed to be written.

Before delving into these improvements to CPAL, lets examine the necessity for simplification during generation of the weakest precondition. It was after partial CPAL specifications of the TLS protocol were run through the evaluation program that this condition was recognized. For the protocols evaluated previously, simplification could wait until the weakest precondition had been generated. At worst each statement in these protocols would only increase the number of predicates in linear fashion.

In contrast if/else statements are much more expensive. An if-else or if-no-else statement will roughly cause a doubling in the size of a large predicate. Consequently, a series of branching statements can cause the size of the predicate to grow exponentially.

Initial partial specifications of TLS encountered this growth problem when the system where the analysis was executing ran out of memory. Less catastrophic was the use of virtual memory by the program as the predicate increased in size. Although this did not stop the program, the use of the disk slowed the speed the program to a crawl.

To fight this problem the predicate size needed to be reduced as the weakest precondition engine was generating the predicate. Initially, the size of problem was not appreciated. Initially the simplification algorithms in place were utilized during predicate generation process. Some simplifications were made, however the process was not powerful enough to stem the growth of the predicate.

As TLS was being specified, the size of the predicate grew larger and larger. However, when ported to PVS, this predicate was simplified substantially, if not

completely. PVS's ability to reduce the predicate after reductions were attempted in CPALES indicated that the creation of a more effective simplification engine would be possible.

A series of improvements were attempted to create a more powerful simplification engine. Most of these initial efforts failed, however they did indicate the weaknesses of the system. This led to improvements that finally led to a workable system.

A function to put the predicate into conjunctive normal form had been written for CPALES. Putting the predicate in this form optimizes the simplification of disjuncts. Unfortunately, the original simplification routines could not take advantage of this. The effectiveness of the simplification routines was hampered by the fact that it only looked at the two children of a predicate. This restricted it from reducing some predicates. For example, look at this predicate: (A or (B or ~A)). This should simplify to true. However, the original routine could only compare A with (B or ~A), and B with ~A, so no simplification could take place.

In order to perform the above simplification with the restricted power of the simplification routine requires that predicates be distributed. Therefore, (A or (B or ~A)) would be transformed to (A or B) or (A or ~A). However, it was slow, could cause tremendous growth in predicate size, and it still did not allow identity simplification. That is, that (A or (B or A)) should reduce to (A or B). The distribution of A on (A or B) still does not permit the simplification with the limited routines, which only look at the two children of a predicate.

To simplify this, a routine needed to be created, which could cross over parent nodes to compare formulas. For example, (A or (B or A)) would normally be seen as (A or B or A) which can easily be seen to reduce to (A or B). This process when broken down requires that all possible two-argument combinations of the predicate must be compared. Notice that the relationships between the elements in the predicate (A or (B or A)) are same. That is why it can be written as (A or B or A) without changing the meaning.

We decided to change this structure in order to facilitate the comparisons between elements in predicate fragments constructed with the same connective. A list fits this requirement. With a list structure, there is no hierarchy to worry about when simplifications are done. Once this list had been created, all the possible combinations could be searched for with loops. As simplifications were made, formulas could be removed from the list without consideration of position in the predicate. Then when simplifications were finished, a normal hierarchical predicate was created from the list.

There are some limitations to this strategy. For one, it can only be complete if all possible combinations among formulas with the same connective are made available. To make the simplification job easier, the predicate needed to be normalized. In particular, it needed to be placed in conjunctive normal form for simplification of $\vee$ lists and in disjunctive normal form for the simplification of $\wedge$ lists. These forms allowed the lists to be easily extracted from the predicate.

At this point the basic elements to provide effective simplification were in place. Now it was only a question of how these routines should be used. After

some experimentation it was discovered that the disjunctive normalization was too expensive to be effective, so this was omitted. Simplification of ∨ lists and then ∧ lists was performed after conjunctive normalization. The new predicate was replaced with the old if normalization and simplification resulted in a larger predicate. At first, simplification was called after every statement evaluation. When this was found to be overzealous, simplifications calls were limited to WP statement transformations that caused the predicate to double in size.

Before this process of simplification was begun, the program would terminate before completion when it ran out of memory. Once the modifications described above were performed, the complete TLS specification could be evaluated in fourteen hours. A final optimization was realized by examination of debugging statements. That revealed that the ∧ list simplification was only producing simplifications of the form (P ∧ TRUE) to P. This type of simplification does not require the expense of the comparison of all the possible two element combinations in a list to perform a complete simplification. The cost of performing the combinations of a list grows with the square of the number of elements of the list. For example, the possible combinations in an *n* element list are (1 + 2 + … + *n*-1) or *n*(*n*-1). Finding all of the TRUE elements in a list requires that all of the one-element combinations be found. In an *n* list of course require *n* operations. This optimization reduced the evaluation time to approximately three hours from fourteen hours.

7. ANALYSIS OF TLS

The goal of our analysis of TLS with the CPALES system was to discover harmful interactions between the various subprotocols. Just as importantly, we wished to demonstrate how these interactions could be represented and modeled by the CPALES system. Our first task was to create a correct implementation of TLS. Some of the details regarding this specification are explained in section 5. Our complete specification of TLS is in appendix B.

The CPAL specification provides a base for the analysis of TLS. Analysis with CPALES reveals whether the TLS protocol satisfies its goals. All of the actions and goals of all of the principals specified in the protocol are examined in this process. This can be helpful in the search for protocol flaws. The ability to specify a correct protocol provides a good working environment from which to examine the protocol for security flaws. The specification provides an environment in which attacks can be added to determine their effectiveness. In addition, once a correct specification has been made, assumptions regarding logical beliefs can be added within this framework.

In the remainder of this analysis portion, we will discuss the examination of interactions between TLS protocols. We will also discuss how to produce a

predicate that examines whether an assertion for one protocol may be satisfied by messages from another protocol. This is done through the creation of special tests that reveal unspecified interactions between subprotocols. Finally, we will discuss a specification that models an attack developed by Wagner and Schneier [WS96]. This attack was initially discovered in the SSL 3.0 protocol, but remains effective in the TLS protocol.

7.1 Understanding the Subprotocol Interactions in TLS

In the TLS protocol there are limitations upon the interactions between subprotocols. One of these limits is that the ciphersuite is chosen at one point in the protocol and cannot be changed later on. This means that a principal's actions are restricted to those of one subprotocol once a ciphersuite value has been accepted. Therefore, a principal may not send a message from one subprotocol in one session, and send out a message from a different subprotocol later on in that same session. Therefore, trustworthy principals cannot be tricked into switching between subprotocols more than once.

This does not mean that the subprotocols will always match up between principals in a session that is under attack. An attacker could still modify the original ciphersuite specification so that the two communicating principals run different subprotocols. Since the ciphersuite value is not a part of every message, a principal may not know if a message received was necessarily generated for that protocol. Consequently, there is the possibility that a message from the

wrong subprotocol may be accepted. If so, this might lead to a weakness that can be exploited.

The TLS specification in appendix B is comprehensive and meets its goals as specified. Examination of how this occurs can be instructive to understanding how the different subprotocols manipulate the predicate as it is being modified by weakest precondition transformations. As the reasoning engine evaluates the protocol, the predicate generated by earlier statements will be passed through every subprotocol. A simple example of this process is shown in Figure 24. The specification was designed so that if A and B are equivalent that predicate will simplify to true. The if conditionals were given different names, A and B, to highlight their origin. The top line of Figure 24 shows the weakest precondition for the two if else statements on the bottom left portion of the Figure.

(~A \/ ~B \/ (y==y)) /\ (~A \/ B \/ (y==z)) /\ (A \/ ~B \/ (z==y)) /\ (A \/ B \/ (z==z))

~A \/ ((~B \/ (y==y)) /\ (B \/ (y==z))) /\ (A \/ ((~B \/ (z==y)) /\ (B \/ (z==z)))

if (A) then {x := y}	A=>((~B \/ (y==y)) /\ (B \/ (y==z))) /\
else {x := z}	~A=>((~B \/ (z==y)) /\ (B \/ (z==z)))
	~B \/ (x==y) /\ (B \/ (x==z))
if (B) then {assert(x == y)}	B=>(x==y) /\ (~B \/ (x==z))
else {assert(x == z);	

Figure 24. WP Reasoning with if else branches

Note in the top line of Figure 24 that when the two conditional values A and B are both preceded or not preceded by the ~ symbol that the comparisons specified by the assert statements are equivalent. This corresponds to the correct branches being taken. When the wrong branches are taken, the comparisons do not match up. These portions of the predicate will still simplify to true since if A and B are equivalent, A \/ ~B must be TRUE. So in brief, in a correctly specified predicate logically inverted conditional statements cancel out interactions between inappropriate subprotocols. When the appropriate subprotocols interact the assertion comparison must match for the protocol to work as designed.

Sometimes we do not desire the elimination of inappropriate interactions. Instead we want to get a look at the comparisons produced by the interaction of the mismatched subprotocols. These interactions will become visible in the predicate if simplification of the mismatched conditionals is prevented. These predicate fragments will be eliminated only if the interaction between the two subprotocols satisfies the goals for those subprotocols. These goals are typically comparisons that examine whether two values are equal. A flaw may be discovered through the discovery of identical or near identical comparisons. A flaw may also be discovered if a predicate fragment that should be produced by an interaction is not found. This means that the interaction between two subprotocols satisfied the goal for the asserted protocol.

This analysis strategy was utilized on TLS. An intruder was added to the specification so that the initial ciphersuite message could be intercepted and replaced with another ciphersuite chosen by the intruder. This change in values al-

lows the effects of subprotocol interactions to be retained within the predicate by preventing A \/ ~A simplification mentioned before. The predicates fragments produced by the subprotocol interactions are now visible in the resulting predicate.

This type of analysis was performed for the client's RSA_EXP subprotocol, which produced a verification condition that could be examined for close message matches. In order to simplify the resulting predicate, only the verification performed by the client on the server's key exchange message is considered. A fragment of the predicate produced can be seen in Figure 25. To see how this predicate describes the protocol interactions, the meaning of a fragment will be explored. The CS value provided by the intruder, termed "I.iCS" in the Figure, prevents the conditionals below from eliminating one another. A combination of this type:

(S.Anon==f.Auth(S.CS)) or ~(S.Anon==f.Auth(S.CS))

i.e. ~ P \/ P => TRUE, would normally eliminate incorrect mixing of predicates. With one of the S.CS values replaced by I.iCS, this reduction cannot occur. As a result the states produced by running mismatched subprotocols are displayed.

In the two \/ lists seen in Figure 25, the initial equivalence comparisons are composed of the values used to determine which subprotocol each principal runs. The comparisons containing the S.CS value correspond to the conditionals that determine the subprotocol used by the server. The comparisons containing the I.iCS value determine the subprotocol executed by the client. These two values are the arguments to functions that represent the extraction of the appropri-

ate ciphersuite value from the CS variable. The values these functions are compared with, S.Anon, S.DH, S.DHE, S.ClientCert, S.RsaExp, S.RSA, S.SignOnly indicate whether the subprotocol utilizes server authentication, Diffie-Hellman key exchange, ephemeral Diffie-Hellman keys, client authentication, export RSA key exchange, strong RSA key exchange, and strong RSA keys for signatures only respectively.

The last of these equivalence comparisons originates from the assertion specified in the protocol. The logical relation between the assertion comparison with the subprotocol conditional comparisons, links the asserted condition to a particular interaction of subprotocols. For the first disjunction in Figure 25 we can ascertain that the server was running a subprotocol with the following characteristics: new session, server authentication, Diffie-Hellman key exchange, and ephemeral keys without client authentication. The client was running a subprotocol with these characteristics: server authentication, Diffie-Hellman key exchange, and fixed keys without client authentication. To discern this, the reviewer must realize that each of the disjuncts except for the assertion must be false for the assertion to affect the logical value of the predicate. Then examination of last comparisons in the disjunction will reveal whether the interaction of these two protocols could produce an unwanted satisfaction of the protocol's requirements.

The first of the two subprotocol interactions that were responsible for the predicate fragment shown in Figure 25 is caused by an unforeseen interaction between the subprotocols of TLS while the second interaction is attributed to the

appropriate interaction. This interaction results when the client uses ephemeral RSA key exchange and the server uses ephemeral Diffie-Hellman key exchange.

```
  …AND
((S.oldSession(C.SID)) or
((S.Anon == f.Auth(I.iCS)) or
((S.DH == f.KEM(I.iCS)) or
(not ((S.RsaExp == f.KEM(I.iCS)) or
(not ((S.SignOnly == f.PK(I.iCS)) or
((S.ClientCert == f.Cert(I.iCS)) or
((S.Anon == f.Auth(S.CS)) or
(not ((S.DH == f.KEM(S.CS)) or
(not ((S.DHE == f.DH(S.CS)) or
((S.ClientCert == f.Cert(S.CS)) or
(f.hash(<C.Nc,S.Ns,<S.P,S.G,f.dhy(<S.P,S.G,S.X>)>>) ==
f.hash(<C.Nc,S.Ns,<S.P,S.G,f.dhy(<S.P,S.G,S.X>)>>))))))))))))
  AND
(S.oldSession(C.SID) or
((S.Anon == f.Auth(I.iCS)) or
((S.DH == f.KEM(I.iCS)) or
(not ((S.RsaExp == f.KEM(I.iCS))) or
(not ((S.SignOnly == f.PK(I.iCS))) or
((S.ClientCert == f.Cert(I.iCS)) or
((S.Anon == f.Auth(S.CS)) or
((S.DH == f.KEM(S.CS)) or
(not ((S.RsaExp == f.KEM(S.CS))) or
(not ((S.SignOnly == f.PK(S.CS))) or
((S.ClientCert == f.Cert(S.CS)) or
(f.hash(<C.Nc,S.Ns,<S.mod,S.exp>>) ==
f.hash(<C.Nc,S.Ns,<S.mod,S.exp>>))))))))))))
  AND …
```

Figure 25. Pred. Fragment of Subprotocol Interactions

The last comparison models the checking of the key exchange values sent by the server with the signature of the hash of those same values. As you can

see, the two values in the comparison are identical. Therefore, the check does not recognize that these values are actually Diffie-Helman values, not RSA values. This makes the Wagner and Schneier attack possible. This predicate result indicates the same the weakness in SSL 3.0 found by Wagner and Schneier [28]. Thus this result demonstrates that CPALES can be used to analyze subprotocol interactions for unforeseen interactions that compromise a protocol's security.

7.2 Modeling the Wagner and Schneier Attack

CPALES allows the representation of the interaction between two subprotocols. In other words, it allows one to see how the protocol suites of two different principals interact. This provides a good environment for the search for weaknesses in a protocol. An attack may be modeled not only against a single subprotocol, but the entire protocol suite used by a principal. One should be able to create an attack on a subprotocol, and then run it against the entire suite. The fact that the subprotocol is part of a suite should not make the subprotocol any weaker. The inclusion of all the subprotocols within one specification also allows attacks utilizing interactions among subprotocols to be accurately specified.

To demonstrate the ability to represent protocol interactions, the Wagner Schneier attack was specified in CPAL for the TLS protocol. In SSL this attack could be launched against ephemeral RSA and ephemeral Diffie-Hellman public keys. The attack exploited the fact that the key exchange mechanism for these two algorithms used very similar methods to exchange the parameters. This allowed an intruder to convince one principal to use the key exchange message for

Diffie-Hellman while the other expects the RSA key exchange. Wagner and Schneier explain that this attack could be prevented if the method of key exchange was signed as well as the key exchange parameters themselves. As it is, Diffie-Helman values supplied for the RSA public key will be insecure given their mathematical properties when used in the RSA public key algorithm. There is still a vulnerability to this attack in TLS. The export version of the RSA algorithm can still utilize ephemeral keys, which allows the attack to proceed.

Our representation of this attack, allows for the choice between subprotocols. A model that did not permit branching could not fully represent the attack. The modeling of if/else branches within CPALES allows the values that determine which subprotocol is chosen to have an actual effect in the model. This allows our specification to include a set of subprotocols. In a more traditional representation of the attack, the client and the server would not have this choice available. Instead they would only carry out the actions of the particular subprotocol required by the attack. Meaning that the specification could only include one subprotocol for each principal.

For your reference the CPAL specification of Wagner and Schneier attack on the TLS protocol is given in Appendix C. This specification is limited to two subprotocols to make it easier to examine the critical portions of this attack. The first job of the intruder in this attack is to intercept the client hello and server hello messages. This allows the attacker to change the ciphersuites specified by the client and the server.

Our representation deviates from this by not having the intruder intercept the client hello message. This means that the attack depends upon the client specifying RSA and the Diffie-Helman as choices in the client hello message. If it only sent the RSA ciphersuite in the list then the server would not choose Diffie-Hellman as the ciphersuite. This does not prevent the attack from occurring or reflect any weakness in the representation; it is just done to simplify the specification of the attack. It also demonstrates that the attack can be run even without interception of the initial message.

The server should believe it is running the Diffie-Helman subprotocol with ephemeral keys and no client authentication. The server chooses its Diffie-Hellman parameters and sends them to the client. The intruder listens to this message to acquire the parameters. The client receives the Diffie-Hellman parameters and interprets them as the RSA public key. Note that at this time a meticulous application could discover this attack. First of all, the structure it receives contains three parameters instead of the two required for the public key. The first two parameters are the correct size for the public key parameters so there is no discrepancy there. The third parameter should not be expected and makes the collection of parameters larger than a public key structure would. In addition, when the sent parameters are checked with the signed hash of the parameters there is another discrepancy. If the client assembles the hash from the extracted parameters instead of the sent structure, the two hashes will not match. The extracted parameters do not include the third Diffie-Hellman parameter so the hash

of the extracted parameters will not match the signature. This difference will not be detected if the parameter structure sent by the server is validated instead.

The client will then generate a secret and encrypt it with the insecure public key material. This is sent along with the message authentication code generated from the secret and the messages it sent and received. The intruder blocks this message and deciphers the weak encryption of the secret. The deciphering was difficult to model given that it depended upon mathematical properties. To allow the intruder to decrypt the message, an assumption was made that the intruder could decrypt a message encrypted with the weak key provided by the server. The assumption is weak since it only works if the first two Diffie-Hellman parameters chosen by the server are utilized in a public key encryption.

The intruder must then create Diffie-Hellman parameters to send to the server. Since the intruder chose the parameters it will, by the property of the Diffie-Hellman key generation system, be able to generate the same secret that the server will. The intruder can then utilize this secret to generate message authentication codes that will spoof the server. The server could prevent being deceived here if it had demanded that the client authenticate itself. The intruder would not have been able to generate the verification message since it does not posses the client's private key.

The server will then be satisfied when it receives the message authentication codes for the session. The intruder can then block the server's message authentication code and create its own using the secret intercepted when it deci-

phered the weak encryption. This completes the attack, which completely subverts the protocol when the client accepts the RSA export ciphersuite.

8. CONCLUSIONS

CPALES allows the combination of logical analysis with weakest precondition reasoning to produce a verification condition. The operation of several protocols found in the Clark and Jacob library were specified and evaluated in CPAL/BAN. This process was able to elicit a weakness in the Kao Chow protocol that has not been noted before. The ordering of assumptions and assertions in the specification allowed by CPAL aided the elicitation of our specification.

Formal analysis of collections of protocols has lagged behind their development. Most formal analysis of these protocols has focused on the security of the subprotocols. These results were used to make informal judgments about the entire protocol's security. Until recently examining protocol families has not been a requirement of protocol analysis. It can be expected that as time goes by, security applications will offer more choices that will complicate this analysis further. In this case, CPALES is well suited to handle the increased complexity. The ability to model the flows of data with CPALES should become more useful as it becomes more difficult to verify this correctness by hand. Currently, logical analysis of these types of protocols has been limited to the subprotocols. Hopefully, this

can change with the use of the weakest precondition engine to analyze logical specifications of the whole protocol.

Although Meadows has already utilized the NRL protocol analyzer the IKE family of protocols. Our system should add a new dimension to the analysis of protocol suites. This should be expected given that our system utilizes different methodologies to examine the protocols.

Currently, logical analysis of TLS is restricted by the lack of support for key agreement within BAN logic. One powerful extension would be the addition of the SvO logic to the workbench. This logic is a refined version of the BAN logic utilized in our system. It utilizes an independent semantics that can aid in the verification process. It also allows key agreement protocols to be analyzed. Since protocols like TLS and IKE utilize Diffie-Helman key agreement. It will allow these protocols to be analyzed without having to make our own extension to the BAN system. Other logics such as Autlog [13], BGNY[3], Kailar [11], Kessler-Neumann [14], could also be adapted to provide additional features for the logic and our system. Logics to handle protocols other than key exchange and authentication such as auctions, fair exchange, or electronic commerce would extend the ability of the system to analyze these types of protocols. The ability to handle a wide range of logical analysis schemes with the addition of their PVS specification gives the CPALES system great flexibility.

Our analysis of TLS and other security protocols in this work is valuable for several reasons. It allowed us to learn more about the protocols analyzed and the capabilities of CPALES. We were able to find weaknesses in the Kao Chow

protocols. Through the analysis of TLS, it was shown that harmful interactions between subprotocols could be discovered with CPALES. The interaction between subprotocols that are the basis of Wagner Schneier attack were accurately specified and proven. With the extensions made to CPALES, other protocols complex such as IKE and SET can also be analyzed with this system. This analysis should provide greater confidence in the security of these collections of protocols.

APPENDIX A

TLS Handshake Protocol in Std. Notation

C -> S: ProtVer, Rc, SID, CSL, CML
S: If (SID NEW) then goto Negotiate New Session

Resume Session
S -> C: ProtVer, Rs, SID(old),CS, CM
S -> C: X
S -> C: PRF(master, “finished”, hash(Previous Messages))
C -> S: X
C -> S: PRF(master, “finished”, hash(Previous Messages))
End

Negotiate New Session
S -> C: (ProtVer, Rs, SID (new), CS, CM)
S: if (Key Exch. is Auth) then goto Authenticated Server

Anonymous Server
S -> C: p, g, Ys
S -> C: X
C -> S: Yc
C -> S: X
C -> S: PRF(master, “finished”, hash(Previous Messages))
S -> C: X
S -> C: PRF(master, “finished”, hash(Previous Messages))
End

Authenticated Server
S: if (Key Exch is ~RSA) then goto Diffie-Helman

RSA
S: if (Key Exch is RSA_EXP & Long Sign K) then goto Temp Key

Encryption Key in Certificate
S -> C: {S, Ks+}Kca-
S: goto Client Response

Temp Key
S -> C: {S, sKs+}Kca-

S -> C: Ks+, {hash(Rc + Rs + Ks+)}sKs-
S: goto Client Response

Diffie-Helman

S: if (Key Exch is DHE) then goto DHE

DH (RSA & DSS signing)

S -> C: {S, (p, g, Ys), sKs+}Kca-
S: Goto Client Response

DHE (RSA & DSS signing)

S -> C: {S, sKs+}Kca-,(p, g, Ys), {hash(Rc + Rs + (p, g, Ys))}sKs-

Client Response

S:if (No Client Cert) THEN goto Client Key Exchange
S -> C: (CTL, CAL)
S -> C: X
C -> S: {C, Kc+}Kca-

Client Key Exchange

C: if (Key Exch is RSA or RSA_EXP) then
    C -> S: {Prot. Ver., PreMaster}Ks+
C: else if (Key Exch is DHE) THEN
    C -> S: Yc
C: else if (Key Exch is DH) THEN
    C -> S: 0

C: if (no Client Cert or Key Exch DH) then goto Client Finished

Certificate Verify

C -> S: {hash(PreviousMessages)}sKc-

Client Finished

C -> S: X
C -> S: PRF(master, “finished”, hash(Previous Messages))
S -> C: X
S -> C: PRF(master, “finished”, hash(Previous Messages))
End

APPENDIX B

TLS Specification in CPAL

```
S: => C(oldMaster);
C: <-(oldMaster);
X: assume(global.decrypt(CA.sKs+,S.sKs-));
X: assume(global.decrypt(S.sKs-,CA.sKs+));
X: assume(global.decrypt(S.Ks-,S.Ks+)); -- temporary KE key
X: assume(global.decrypt(S.sKca+,CA.sKca-));
X: assume(global.decrypt(C.sKca+,CA.sKca-));
X: assume(global.decrypt(CA.sKc+,C.sKc-));
S: modexp := <mod,exp>;

CA: => S(ep[<S, sKs+>]sKca-);
S: <- (certS);
CA: => C(ep[<C, sKc+>]sKca-);
C: <- (certC);
CA: fdhpS := <P,G,dhy(<P,G,X>)>;
CA: fdhpC := <P,G,dhy(<P,G,Y>)>;
CA: => S(<ep[<S, fdhpS,sKs+>]sKca-,X>); -- assume X is secure.
S: <- (CertNSecret);
S: (DHcertS, Xf) := CertNSecret; -- Cert with fixed DH params + fixed secret
S: (S,fdhpS,sKs+) := dp[DHcertS]sKca+;
S: (Pf,Gf,dhYsf) := fdhpS;
CA: => C(<ep[<C, fdhpC,sKc+>]sKca-,Y>);
C: <- (CertNSecret);
C: (DHcertC, Yf):= CertNSecret;
C: (C,fdhpC,sKc+) := dp[DHcertC]sKca+;
C: (Pf,Gf,dhYcf) := fdhpC;

-- Get equivalent test cases.
S: => C(<Anon,DH,DHE,RsaExp,SignOnly,Rsa,ClientCert>);
C: <-(checks);
C: (Anon,DH,DHE,RsaExp,SignOnly,Rsa,ClientCert) := checks;

-- START

C: CH := <PVc, Nc, SID, CSL, SML>;
C: => S(CH);
S: <-(CH);
```

```
S: (PVc, Nc, tSID, CSL, SML) := CH;
S: if (oldSession(tSID)) then {SID := tSID;} else {SID := new;}
S: SH := <PVs, Ns, SID, CS, CM>;

S: =>C(SH);
C: <-(SH);
C: (PV, Ns, sSID, CS, CM) := SH;

S: KEmeth := KEM(CS);
S: AuthMeth := Auth(CS);
S: CertMeth := Cert(CS);
S: PKmeth := PK(CS);
S: DHmeth := DH(CS);

S: if (SID == tSID) then {
     mac := hash(<CH,SH>);
     FINs := prf(<oldMaster,mac>);
     msg3 := FINs;
   }
   else {if (AuthMeth == Anon) then {
     SKE := <P,G,dhy(<P,G,X>)>;
     msg3 := SKE;
   }
   else {if (KEmeth == DH) then {
     if (DHmeth == DHE) then {
       if (CertMeth == ClientCert) then {
         SC := certS;
         dhp := <P,G,dhy(<P,G,X>)>;
         SKE := <dhp,ep[hash(<Nc,Ns,dhp>)]sKs->;
         CR := <CertTypeLst,CertAuthLst>;
         msg3 := <SC,SKE,CR>;
       }
       else {
         SC := certS;
         dhp := <P,G,dhy(<P,G,X>)>;
         SKE := <dhp,ep[hash(<Nc,Ns,dhp>)]sKs->;
         msg3 := <SC,SKE>;
       }
     }
     else { -- DH
       if (CertMeth == ClientCert) then {
         SC := DHcertS;
         CR := <CertTypeLst,CertAuthLst>;
         msg3 := <SC,CR>;
       }
       else {
         SC := DHcertS;
         msg3 := SC;
       }
     }
   }
   else {if (KEmeth == RsaExp) then {
     if (PKmeth == SignOnly) then { -- sKs+ is longer than 512 bits.
       if (CertMeth == ClientCert) then {
```

```
            SC := certS;
            Ks+ := <mod,exp>;
            SKE := <Ks+,ep[hash(<Nc,Ns,Ks+>)]sKs->; -- Ks+ is enc pk
            CR := <CertTypeLst,CertAuthLst>;
            msg3 := <SC,SKE,CR>;
          }
          else {
            SC := certS;
            Ks+ := <mod,exp>;
            SKE := <Ks+,ep[hash(<Nc,Ns,Ks+>)]sKs->; -- Ks+ is enc pk
            msg3 := <SC,SKE>;
          }
        }
        else { -- sKs+ is <= 512 bits
          if (CertMeth == ClientCert) then {
            SC := certS;
            CR := <CertTypeLst,CertAuthLst>;
            msg3 := <SC,CR>;
          }
          else {
            SC := certS;
            msg3 := SC;
          }
        }
      }
      else {if (KEmeth == Rsa) then {
        if (CertMeth == ClientCert) then {
          SC := certS;
          CR := <CertTypeLst,CertAuthLst>;
          msg3 := <SC,CR>;
        }
        else {
          SC := certS;
          msg3 := SC;
        }
      } -- END of else if (KEmeth == Rsa)
      else {
        msg3 := nothing3;
      }
      } -- END of else if (KEmeth == RsaExp)
      } -- END of else if (KEmeth == DH)
      } -- END of else if (AuthMeth == Anon)
      } -- END of if (SID == tSID)
S: => C(msg3);

C: <-(msg3);
C: KEmeth := KEM(CS);
C: AuthMeth := Auth(CS);
C: CertMeth := Cert(CS);
C: PKmeth := PK(CS);
C: DHmeth := DH(CS);
C: if (SID == sSID) then {
     FINs := msg3;
     mac := hash(<CH,SH>);
```

```
    test := prf(<oldMaster,mac>);
    assert(FINs == test);
    prvMsgs := hash(<CH,SH,FINs>);
    FINc := prf(<oldMaster,prvMsgs>);
    msg4 := FINc;
  }
  else {if (AuthMeth == Anon) then { -- DH_anon
    SKE := msg3;
    (P,G,dhYs) := SKE;
    CKE := dhy(<P,G,Y>);
    preMaster := dhy(<P,dhYs,Y>);
    master := prf(<preMaster,Nc,Ns>);
    msgCheck := hash(<CH,SH,SKE,CKE>);
    FINc := prf(<master,msgCheck>);
    msg4 := <CKE,FINc>;
  }
  else {if (KEmeth == DH) then {
    if (DHmeth == DHE) then {  -- DHE_X*
      if (CertMeth == ClientCert) then {
        (SC,SKE,CR) := msg3;
        (S, sKs+) := dp[SC]sKca+;
        (dhp,signH) := SKE;
        keyCheck := dp[signH]sKs+;
        keyTest := hash(<Nc,Ns,dhp>);
        assert(keyCheck == keyTest);
        (CertTypeLst,CertAuthLst) := CR;
        (P,G,dhYs) := dhp;
        CC := certC;
        CKE := dhy(<P,G,Y>);
        CV := ep[hash(<CH,SH,SC,SKE,CR,CC,CKE>)]sKc-;
        preMaster := dhy(<P,dhYs,Y>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CC,CKE,CV,FINc>;
      }
      else {
        (SC,SKE) := msg3;
        (S, sKs+) := dp[SC]sKca+;
        (dhp,signH) := SKE;
        keyCheck := dp[signH]sKs+;
        keyTest := hash(<Nc,Ns,dhp>);
        assert(keyCheck == keyTest);
        (P,G,dhYs) := dhp;
        CKE := dhy(<P,G,Y>);
        preMaster := dhy(<P,dhYs,Y>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CKE,FINc>;
      }
    }
    else { -- DH_DSS & DH_RSA
      if (CertMeth == ClientCert) then {
```

```
        (SC,CR) := msg3;
        (S,fdhpS,sKs+) := dp[SC]sKca+;
        (CertTypeLst,CertAuthLst) := CR;
        CC := DHcertC;
        CKE := null;
        (Ps,Gs,dhYs) := fdhpS;
        assert(<Ps,Gs> == <Pf,Gf>);
        preMaster := dhy(<Pf,dhYs,Yf>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CC,CKE,FINc>;
      }
      else { -- DH_anon
        SC := msg3;
        (S,fdhpS,sKs+) := dp[SC]sKca+;
        (Ps,Gs,dhYs) := fdhpS;
        assert(<Ps,Gs> == <Pf,Gf>);
        --dhYc := dhy(<Pf,Gf,Yf>);
        CKE := <Pf,Gf,dhYcf>;
        preMaster := dhy(<Pf,dhYs,Yf>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CKE,FINc>; -- there is no CV for DH
      }
    }
  }
  else {if (KEmeth == RsaExp) then {
    if (PKmeth == SignOnly) then { -- sKs+ > 512 bits
      if (CertMeth == ClientCert) then {
        (SC,SKE,CR) := msg3;
        -- sKs+ is > 512 bits, too large to enc. by old US law.
        (S,sKs+) := dp[SC]sKca+;
        -- Ks+ is <= 512 bit enc. key, compliant to old enc. laws.
        (keyparams,signH) := SKE;
        (mod,exp) := keyparams;
        Ks+ := <mod,exp>;
        keyCheck := dp[signH]sKs+;
        keyTest := hash(<Nc,Ns,keyparams>);
        assert(keyCheck == keyTest);
        (CertTypeLst,CertAuthLst) := CR;
        CC := certC;
        preMaster := new;
        CKE := ep[<PVc,preMaster>]Ks+; -- diff with below
        CV := ep[hash(<CH,SH,SC,SKE,CR,CC,CKE>)]sKc-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CC,CKE,CV,FINc>;
      }
      else {
        (SC,SKE) := msg3;
        (S,sKs+) := dp[SC]sKca+;
```

```
        -- Ks+ is < 512 bit enc. key, compliant to old enc. laws.
        (keyparams,signH) := SKE;
        (mod,exp) := keyparams;
        Ks+ := <mod,exp>;
        keyCheck := dp[signH]sKs+;
        keyTest := hash(<Nc,Ns,keyparams>);
        assert(keyCheck == keyTest);
        preMaster := new;
        CKE := ep[<PVc,preMaster>]Ks+;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CKE,FINc>;
      }
    }
    else { -- sKs+ is <= 512 bits
      if (CertMeth == ClientCert) then {
        (SC,CR) := msg3;
        -- sKs+ is < 512 bit enc. key, compliant to old enc. laws.
        (S,sKs+) := dp[SC]sKca+;
        -- assertion of peer identity here?
        (CertTypeLst,CertAuthLst) := CR;
        CC := certC;
        preMaster := new;
        CKE := ep[<PVc,preMaster>]sKs+;
        CV := ep[hash(<CH,SH,SC,CR,CC,CKE>)]sKc-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CC,CKE,CV,FINc>;
      }
      else {
        SC := msg3;
        (S,sKs+) := dp[SC]sKca+;
        -- Ks+ is < 512 bit enc. key, compliant to old enc. laws.
        preMaster := new;
        CKE := ep[<PVc,preMaster>]sKs+;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINc := prf(<master, msgCheck>);
        msg4 := <CKE,FINc>;
      }
    }
  }
  else { if (KEmeth == Rsa) then { -- sKs+ <= 512, so can encr & sign
    if (CertMeth == ClientCert) then {
      (SC,CR) := msg3;
      (S, sKs+) := dp[SC]sKca+;
      CC := certC;
      preMaster := new;
      CKE := ep[<PVc,preMaster>]sKs+;
      CV := ep[hash(<CH,SH,SC,CR,CC,CKE>)]sKc-;
      master := prf(<preMaster,Nc,Ns>);
      msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV>);
```

```
        FINc := prf(<master,msgCheck>);
        msg4 := <CC,CKE,CV,FINc>;
      }
      else {
        SC := msg3;
        (S, sKs+) := dp[SC]sKca+;
        -- assertion of peer identity?
        preMaster := new;
        CKE := ep[<PVc,preMaster>]sKs+;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CKE,FINc>;
      }
    }
    }
    }
    }
    }

C: => S(msg4);
S: <-(msg4);
S: if (SID == tSID) then {
      FINc := msg4;
      msgCheck := hash(<CH,SH,FINs>);
      FINcheck := prf(<oldMaster,msgCheck>);
      assert(FINc == FINcheck);
    }
    else {if (AuthMeth == Anon) then { -- DH_anon
      (CKE,FINc) := msg4;
      dhYc := CKE;
      preMaster := dhy(<P,dhYc,X>);
      master := prf(<preMaster,Nc,Ns>);
      msgCheck := hash(<CH,SH,SKE,CKE>);
      FINcheck := prf(<master,msgCheck>);
      assert(FINc == FINcheck);
      msgCheck := hash(<CH,SH,SKE,CKE,FINc>);
      FINs := prf(<master,msgCheck>);
      msg5 := FINs;
    }
    else {if (KEmeth == DH) then {
      if (DHmeth == DHE) then {  -- DHE_X*
        if (CertMeth == ClientCert) then {
          (CC,CKE,CV,FINc) := msg4;
          (C, sKc+) := dp[CC]sKca+;
          dhYc := CKE;
          certCheck := dp[CV]sKc+;
          certTest := hash(<CH,SH,SC,SKE,CR,CC,CKE>);
          assert(certCheck == certTest);
          preMaster := dhy(<P,dhYc,X>);
          master := prf(<preMaster,Nc,Ns>);
          msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV>);
          FINcheck := prf(<master,msgCheck>);
          assert(FINc == FINcheck);
```

```
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
      else {
        (CKE,FINc) := msg4;
        dhYc := CKE;
        preMaster := dhy(<P,dhYc,X>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
    }
    else { -- DH_DSS & DH_RSA
      if (CertMeth == ClientCert) then {
        (CC,CKE,FINc) := msg4;
        (C,fdhpC,sKc+) := dp[CC]sKca+;
        (Pc,Gc,dhYc) := fdhpC;
        assert(<Pf,Gf> == <Pc,Gc>);
        preMaster := dhy(<Pf,dhYc,Xf>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
      else { -- DH_anon
        (CKE,FINc) := msg4;
        --dhYs := CKE; incorrect
        (Pc,Gc,dhYc) := CKE;
        assert(<Pf,Gf> == <Pc,Gc>);
        preMaster := dhy(<Pf,dhYc,Xf>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
    }
  }
  else {if (KEmeth == RsaExp) then {
    if (PKmeth == SignOnly) then { -- sKs+ > 512 bits
      if (CertMeth == ClientCert) then {
        (CC,CKE,CV,FINc) := msg4;
        (C, sKc+) := dp[CC]sKca+;
        certCheck := dp[CV]sKc+;
```

```
        certTest := hash(<CH,SH,SC,SKE,CR,CC,CKE>);
        assert(certCheck == certTest);
        (PVc,preMaster) := dp[CKE]Ks-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
      else {
        (CKE,FINc) := msg4;
        (PVc,preMaster) := dp[CKE]Ks-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
    }
    else { -- sKs+ is <= 512 bits
      if (CertMeth == ClientCert) then {
        (CC,CKE,CV,FINc) := msg4;
        (C, sKc+) := dp[CC]sKca+;
        certCheck := dp[CV]sKc+;
        certTest := hash(<CH,SH,SC,CR,CC,CKE>);
        assert(certCheck == certTest);
        (PVc,preMaster) := dp[CKE]sKs-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
      else {
        (CKE,FINc) := msg4;
        (PVC,preMaster) := dp[CKE]sKs-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
    }
  }
  else { if (KEmeth == Rsa) then { -- sKs+ <= 512, so can enc & sign
    if (CertMeth == ClientCert) then {
```

```
        (CC,CKE,CV,FINc) := msg4;
        (C, sKc+) := dp[CC]sKca+;
        certCheck := dp[CV]sKc+;
        certTest := hash(<CH,SH,SC,CR,CC,CKE>);
        assert(certCheck == certTest);
        (PVc,preMaster) := dp[CKE]sKs-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
      else {
        (CKE,FINc) := msg4;
        (PVc,preMaster) := dp[CKE]sKs-;
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,CKE>);
        FINcheck := prf(<master,msgCheck>);
        assert(FINc == FINcheck);
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINs := prf(<master,msgCheck>);
        msg5 := FINs;
      }
    }
    }
    }
    }
    }
    }
S: => C(msg5);
C: <-(msg5);
C: if (SID == sSID) then {
      we := are_finished;
    }
    else {
      if (AuthMeth == Anon) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SKE,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
      else {if (KEmeth == DH) then {
        if (DHmeth == DHE) then {
          if (CertMeth == ClientCert) then {
            FINs := msg5;
            msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV,FINc>);
            FINCheck := prf(<master,msgCheck>);
            assert(FINs == FINCheck);--
          }
          else {
            FINs := msg5;
            msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
            FINCheck := prf(<master,msgCheck>);
```

```
        assert(FINs == FINCheck);
      }
    }
    else { -- DH_DSS & DH_RSA
      if (CertMeth == ClientCert) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck); -- prob
      }
      else {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
    }
  }
  else {if (KEmeth == RsaExp) then {
    if (PKmeth == SignOnly) then { -- sKs+ > 512 bits
      if (CertMeth == ClientCert) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,SKE,CR,CC,CKE,CV,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
      else {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
    }
    else { -- sKs+ is <= 512 bits
      if (CertMeth == ClientCert) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
      else {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
      }
    }
  }
  else { if (KEmeth == Rsa) then { -- sKs+ <= 512
    if (CertMeth == ClientCert) then {
      FINs := msg5;
      msgCheck := hash(<CH,SH,SC,CR,CC,CKE,CV,FINc>);
      FINCheck := prf(<master,msgCheck>);
      assert(FINs == FINCheck); --
```

```
      }
      else {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck); --
      }
    }
    }
    }
    }
}
```

APPENDIX C

Wagner and Schneier
Key Exchange Algorithm Rollback attack
On TLS

```
X: assume(global.decrypt(CA.sKs+,S.sKs-));
X: assume(global.decrypt(C.sKca+,CA.sKca-));
S: assume(S.RsaExp <> S.DH);
S: PG := <P,G>;
X: assume(global.decrypt(I.Ks-,S.PG));

CA: => S(ep[<S, sKs+>]sKca-);
S: <- (certS);
CA: => C(ep[<C, sKc+>]sKca-);
C: <- (certC);

S: -> C(<Anon,DH,DHE,RsaExp,SignOnly,Rsa,ClientCert>);
I: <-(checks);
I: (Anon,DH,DHE,RsaExp,SignOnly,Rsa,ClientCert) := checks;
I: =>C(checks);
C: <-(checks);
C: (Anon,DH,DHE,RsaExp,SignOnly,Rsa,ClientCert) := checks;

-- START

C: CH := <PVc, Nc, SID, CSL, SML>;
C: -> S(CH);
I: <-(CH);  -- skipped interception since current spec. doesn't use it.
I: (PVc, Nc, tSID, CSL, SML) := CH;
I: => S(CH);
S: <-(CH);
S: (PVc, Nc, tSID, CSL, SML) := CH;
S: SID := tSID;
S: CS :=  <NO,DH,DHE,NO,NO>;
S: (AuthMeth, KEmeth, DHmeth, CertMeth, PKmeth) := CS;
S: SH := <PVs, Ns, SID, CS, CM>;

S: ->C(SH);
I: <-(SH); -- message interception
```

```
I: (PV, Ns, sSID, CS, CM) := SH;
I: CS := <NO,RsaExp,NO,NO,SignOnly>;
I: iSH := <PV, Ns, sSID, CS, CM>;
I: =>C(iSH);
C: <-(SH);
C: (PV, Ns, sSID, CS, CM) := SH;
C: (AuthMeth, KEmeth, DHmeth, CertMeth, PKmeth) := CS;

S: if (KEmeth == DH) then {
        SC := certS;
        dhYs := dhy(<P,G,X>);
        dhp := <P,G,dhYs>;
        SKE := <dhp,ep[hash(<Nc,Ns,dhp>)]sKs->;
        msg3 := <SC,SKE>;
  }
  else {if (KEmeth == RsaExp) then {
        SC := certS;
        Ks+ := <mod,exp>;
        SKE := <Ks+,ep[hash(<Nc,Ns,Ks+>)]sKs->; -- Ks+ is enc pk
        msg3 := <SC,SKE>;
  }
  }

S: -> C(msg3);
I: <-(msg3); -- message interception
I: (SC,SKE) := msg3;
I: (PGdhYs,sig) := SKE; -- use own ciphersuite choice
I: (P,G,dhYs) := PGdhYs;
I: Ks+ := <P,G>;

I: =>C(msg3);
C: <-(msg3);
C: if (KEmeth == DH) then {
        (SC,SKE) := msg3;
        (S, sKs+) := dp[SC]sKca+;
        (PGdhp,signH) := SKE;
        (P,G,dhp) := PGdhp;
        keyCheck := dp[signH]sKs+;
        keyTest := hash(<Nc,Ns,PGdhp>);
        assert(keyCheck == keyTest);
        dhYs := dhp;
        CKE := dhy(<P,G,Y>);
        preMaster := dhy(<P,dhYs,Y>);
        master := prf(<preMaster,Nc,Ns>);
        msgCheck := hash(<CH,SH,SC,SKE,CKE>);
        FINc := prf(<master,msgCheck>);
        msg4 := <CKE,FINc>;
  }
  else {if (KEmeth == RsaExp) then {
        (SC,SKE) := msg3;
        (S,sKs+) := dp[SC]sKca+;
        -- Ks+ is < 512 bit enc. key, compliant to old enc. laws.
        (modexp,signH) := SKE;
        (mod,exp) := modexp;
```

```
          Ks+ := <mod,exp>;
          keyCheck := dp[signH]sKs+;
          keyTest := hash(<Nc,Ns,modexp>);
          assert(keyCheck == keyTest);
          preMaster := new;
          CKE := ep[<PVc,preMaster>]Ks+;
          master := prf(<preMaster,Nc,Ns>);
          msgCheck := hash(<CH,SH,SC,SKE,CKE>);
          FINc := prf(<master,msgCheck>);
          msg4 := <CKE,FINc>;
  }
  }

C: -> S(msg4);
I: <-(msg4);
I: (CKE,FINc) := msg4;
I: (PVc, preMasterC) := dp[CKE]Ks-;
I: gassert(I.preMasterC == C.preMaster);
I: iCKE := dhy(<P,G,Y>);
I: preMasterS := dhy(<P,dhYs,Y>);
I: msgCheck := hash(<CH,SH,SC,SKE,iCKE>);
I: masterS := prf(<preMasterS,Nc,Ns>);
I: FINi := prf(<masterS,msgCheck>);
I: msg4 := <iCKE,FINi>;

I: => S(msg4);
S: <-(msg4);
S: if (KEmeth == DH) then {
          (CKE,FINc) := msg4;
          dhYc := CKE;
          preMaster := dhy(<P,dhYc,X>);
          master := prf(<preMaster,Nc,Ns>);
          msgCheck := hash(<CH,SH,SC,SKE,CKE>);
          FINcheck := prf(<master,msgCheck>);
          assert(FINc == FINcheck);
          msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
          FINs := prf(<master,msgCheck>);
          msg5 := FINs;
  }
  else {if (KEmeth == RsaExp) then {
          (CKE,FINc) := msg4;
          (PVc,preMaster) := dp[CKE]Ks-;
          master := prf(<preMaster,Nc,Ns>);
          msgCheck := hash(<CH,SH,SC,SKE,CKE>);
          FINcheck := prf(<master,msgCheck>);
          assert(FINc == FINcheck);
          msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
          FINs := prf(<master,msgCheck>);
          msg5 := FINs;
  }
  }
S: -> C(msg5);
I: <-(msg5);
I: master := prf(<preMasterC,Nc,Ns>);
```

```
I: msgCheck := hash(<CH,iSH,SC,SKE,CKE,FINc>);
I: FINi := prf(<master,msgCheck>);
I: msg5 := FINi;

I: =>C(msg5);
C: <-(msg5);
C: if (KEmeth == DH) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
  }
  else {if (KEmeth == RsaExp) then {
        FINs := msg5;
        msgCheck := hash(<CH,SH,SC,SKE,CKE,FINc>);
        FINCheck := prf(<master,msgCheck>);
        assert(FINs == FINCheck);
  }
  }
```

REFERENCES

[1] R. Anderson, R. M. Needham. Programming Satan's Computer, in *Computer Science Today*, Springer LNCS v 1000, pages 426-441, 1995

[2] M. Burrows, M. Abadi, Roger Needham. A Logic of Authentication. Research Report 39, Digital Systems Research Center, February 1989.

[3] S. Brackin. A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In *Proceedings of the 1996 IEEE Computer Security Foundations Workshop IX*, pages 62-76, IEEE Computer Society Press, 1996.

[4] S. H. Brackin. Empirical Tests of the Automatic Authentication Protocol Analyzer, 2nd Version (AAPA2). Arca Systems / Exodus Communications Ithaca, NY June 1999

[5] J. Clark and J. Jacob, A Survey of Authentication Protocol Literature: Version 1.0, Unpublished Manuscript, November 17 1997

[6] T. Dierks and C. Allen. The TLS protocol: Version 1.0. Request for Comments: 2246, available at ftp://ftp.isi.edu/in-notes/rfc2246.txt.

[7] D. E. Denning, G. M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, Vol. 24, Num 8. August 1981.

[8] E. W. Dijkstra. A Discipline of Programming. Prentice Hall Series in Automatic Computation, Prentice-Hall Inc. Englewood Cliffs, NJ, 1976.

[9] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*. pages 234-248, IEEE Computer Society Press, Los Alamitos, California, 1990.

[10] S. Gritzalis, D. Spinellis, and P. Georgiadis. Security Protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications*, 22(8): 695-707, May 1999.

[11] R. Kailar. Reasoning about Accountability in Protocols for Electronic Commerce. *In Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 236-250, IEEE Computer Society Press, 1995

[12] I. Kao, R. Chow An Efficient and Secure Authentication Protocol Using Uncertified Keys, *Operating Systems Review 29*, pages14-21, July 1995.

[13] V. Kessler, G. Wedel. AUTLOG-An advanced Logic of Authentication. In *Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII*, pages 90-99, IEEE Computer Society Press, 1994.

[14] K. V., H. Neumann, A Sound Logic for Analysing Electronic Commerce Protocols. In *ESORICS'98 Proceedings of the Fifth European Symposium on Research in Computer Security*, pages 345-360, Springer- Verlag, 1998.

[15] J. Kelsey, B. Schneier. Chosen interactions and the chosen protocol attack. *In Security Protocols, 5th International Workshop April 1997 Proceedings*, pages 91-104, Springer-Verlag, 1998.

[16] G. Lowe. An Attack on the Needham Schroeder Public Key Protocol. *Information Processing Letters*, 56:131-133, 1995.

[17] A. Kehne, J. Schonwalder, H. Langendorfer. A Nonce-Based Protocol For Multiple Authentications. *Operating Systems Review*, 26(4): 84-89,1992.

[18] Catherine Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. *IEEE Symposium on Security and Privacy*, 1999.

[19] J. C. Mitchell, V. Shmatikov, U. Stern. Finite State Analysis of SSL 3.0 and Related protocols. *Workshop on Design and Formal Verification of Security Protocols*, DIMACS September 1997.

[20] B. C. Neuman, S. G. Stubblebine. A Note on the Use of Timestamps as Nonces. *Operating Systems Review*, 27(2):10-14, April, 1993.

[21] R. M. Needham, M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, Vol. 21,Num. 12, December 1978

[22] S. Owre, N. Shankar, and J. M. Rushby. User Guide for the PVS Specification and Verification System (Draft). Computer Science Laboratory, SRI International, Menlo Park, CA, March 1, 1993.

[23] L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. Technical Report 440, Cambridge University Computer Science Laboratory, 1998.

[24] E. Snekkenes. Exploring the ban approach to protocol analysis. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 171--181, May 1991.

[25] P. F. Syverson and P. C. van Oorschot. On Unifying Some Cryptographic Protocol Logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Los Alamitos, California, 1994

[26] P. F.Syverson. Adding Time to a Logic of Authentication. In *Proceedings of the First ACM Conference on Computer and Communications Security*, Fairfax VA, ACM Press, New York, 1993.

[27] P. F. Syverson. A Taxonomy of Replay Attacks. *Proceedings of the Computer Security Foundations Workshop VII,* Franconia NH, IEEE CS Press, Los Alamitos, 1994.

[28] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. In D. Tygar Ed., *USENIX Workshop on Electronic Commerce*, pages 29-40. USENIX Association, 1996.

[29] A. Yasinsac. A Formal Semantics for Evaluating Cryptographic Protocols. Ph.D. Dissertation, University of Virginia, January 1996

[30] A. Yasinsac and W. A. Wulf. Using Weakest Preconditions to Evaluate Cryptographic Protocols. *Cambridge International Workshop on Cryptographic Protocols*, March 1996

[31] A. Yasinsac and W. A. Wulf. A Framework for A Cryptographic Protocol Evaluation Workbench. *Proceedings of the Fourth IEEE International High Assurance Systems Engineering Symposium* (HASE99), Washington D.C., Nov. 1999

BIOGRAPHICAL SKETCH

The author acquired a degree in applied biology from the Georgia Institute of Technology in 1995. For a short time he worked as a lab assistant analyzing the anaerobic respiration of *Shewanella putrefaciens* at Georgia Tech before beginning his studies in Computer Science at Florida State University.