A Formal Semantics for Evaluating Cryptographic Protocols

Alec F. Yasinsac and William A. Wulf

Abstract

Much research in the field of network security is concentrated on the technology and application of cryptography. While the cryptographic methods are heavily investigated to ensure confidence in the security of the code, many cryptographic schemes are vulnerable due to the protocols used to implement communication in a cryptographic environment. Much work has been done to develop secure protocols, but protocols, like software, are very difficult to verify. Recent research is aimed at finding methods of verifying cryptographic protocols, though no method has achieved widespread acceptance and use. The research presented here is focused on developing a methodology for verifying cryptographic protocols based on the classical program verification technique of "weakest precondition". A specification language based on an ad hoc standard "pseudo code" will be proposed and a formal semantics given to allow evaluation of cryptographic protocols.

1.0 Cryptographic Protocols

Much research in the field of network security is concentrated on the technology and application of cryptography. This work centers on the investigation of cryptographic methods to ensure confidence in the strength of the encryption algorithms. However, many cryptographic schemes are vulnerable due to weaknesses in protocols used to implement communication in a cryptographic environment. There are many protocols in literature [NEED78], [OTWY87], [DENN81], and [DOL83] for example, which were designed to provide the needed security. Unfortunately, cryptographic protocols, like computer software, are very difficult to verify. Recent research is aimed at finding methods of verifying cryptographic protocols, though no method has achieved widespread acceptance and use. The research presented here focuses on developing a methodology for verifying cryptographic protocols based on the classical program verification technique of "weakest precondition". We give a specification language based on an ad hoc standard "pseudo code" and propose a methodology for developing a formal semantics of that language to allow evaluation of cryptographic protocols.

Cryptography is useful in the network communications environment because it provides secrecy of data transmitted between sender and receiver for some period of time. If perfectly implemented, only the intended recipient can read an encrypted message unless an intruder is able to compromise the key. Encryption is designed so that such cryptanalysis is expected to take sufficiently long that any data compromised through cryptanalysis will be useless. Unfortunately, cryptology alone does not answer all the problems of network security. In order to address problems of integrity, authentication, key distribution, etc., the cryptographic algorithm must be combined with a valid handshaking scheme, or protocol, which establishes the rules governing interactions between legal participants in the communication (principals). Only when combined with a valid protocol can cryptography provide a principal a "secure channel"; that is, a channel over which authenticated, private communications may occur.

In order to illustrate the need for cryptographic protocols, consider an example of a stock brokerage firm using cryptography to communicate from its main office, where each broker shares a private key with the main office. On a given day, the main office (say M) may desire that a given broker (B) buy or sell a particular stock (S). The first thing in the morning, the main office might send a message encrypted under B's private key (kmb) instructing B to buy:

M: SND(B|e{'Buy $10,000 of S'}\kmb)

The notation is taken from our Cryptographic Protocol Analysis Language (CPAL). 'M' is the party taking the action, SND is the action taken, 'B' is the destination of the SND, and the literal message 'Buy $10,000 of S' is encrypted under key 'kmb' before transmission.

This is a simple protocol. When B receives the message addressed to him, the common key 'kmb' can be used to decrypt the message and the action described in the message can be taken. But what if an astute adversary is listening on the net and recording the encrypted messages as they occur? By watching the subsequent actions of the broker on the floor, the adversary could reasonably predict what the messages said, after the fact. If this adversary could subsequently intercept messages from the main office and substitute recorded messages either at random or with some

selectivity based on his own insight, he could potentially do damage to the brokerage firm without ever compromising an encryption key or penetrating the encryption algorithm.

Though this vulnerability might be resolved by appending a time/date stamp to the message, it clearly illustrates that authenticating users across a network is not as straight forward as one might think. Unfortunately, the problem quickly becomes more complicated as the varied goals and environments are considered. For example, cryptographic protocols are frequently used for authentication such as in the example just presented. However, this example illustrated only "one-way" authentication. Only the broker needed to be sure that the main office sent the message. In another environment, it may be important for "two-way" authentication to occur, that is, for each party in the communication to know who is on the other end. It may also be important to be able to prove at a later date that a particular communication came from its signed originator. This can be accomplished by utilizing a third variety of authentication protocol called "digital signatures".

We have described three types of authentication that can be accomplished using cryptographic protocols and there are others. What is more, cryptographic protocols are not only used for authentication. They can also be utilized for key distribution, to ensure integrity of messages, to detect denial of service and for variations on and combinations of all the above objectives.

Goals of cryptographic protocols are further complicated by the varied security environments existing in distributed systems. Much of the work in literature has focused on a security environment that includes an authentication server [DENN81], [NEED78], [OTWY87]. Authentication servers are principals on the communication network who assist other principals in establishing secure communication channels. The traditional function of authentication servers is to assist principals in authentication by providing such simple tasks as directory services for public keys and in creation and distribution of valid cryptographic keys. However, authentication servers create several security problems themselves. In order to accomplish its functions, the authentication server must have a valid shared key with each distributed principal. Distribution and protection of these keys is a major security problem. Additionally, having a central authority which principals must communicate through is contrary to the philosophy of distributed systems. In this scenario, the authentication server is a single point of failure of the system and is also a potential system bottleneck, which are two critical flaws that distributed systems are designed to avoid. There are cryptographic environments that do not use the authentication server model. These systems are supported by protocols which distribute the security functions of authentication and creation of keys to the distributed principals. Cryptographic protocols in an authentication server environment differ significantly from those without an authentication server.

A further complicating factor in evaluating whether or not cryptographic protocols meet their goals is the cryptographic technique utilized. Protocols for public and private keys systems differ in both the assumptions that can be made, and in the steps required to accomplish various goals. Private, or shared key systems, require that any two principals desiring to have secure communication must share a single private key agreed to by the parties before the protocol begins. That key is used by the sender for encryption and by the receiver for decryption. Public key systems, on the other hand, have separate keys for encryption and decryption, called the "public key" and its "inverse". Each station's public key is generally distributed universally to all other principals. Stations wishing to communicate secretly with another principal encrypt the message using that prin-

cipal's public key and send it on the net. Since only the holder of the "inverse" key can decrypt the message, the communication is private. The original message may contain a shared key to be used for the session, or the recipient may respond using the public key of the originator. Authentication and other cryptographic protocols in public and private key environments may differ significantly.

1.1 Previous Research Regarding Cryptographic Protocols

The foundation for the research presented here lies in the 1978 paper by Needham and Schroeder [NEED78]. In that paper, the authors present protocols for systems with authentication servers for shared and public key authentication and signatures. These protocols effectively highlight many of the important issues in cryptographic protocols and are frequently used in current literature for purposes of illustration. Needham and Schroeder propose three protocols: one for private key systems, another for public key systems, and a signature protocol. Each of these protocols are written for systems which utilize authentication servers.

In 1981, Denning and Sacco [DENN81] utilized the Needham and Schroeder private key protocol to illustrate the problem of replay. They showed that if an intruder that is recording messages can compromise a session key over a period of time, then he can begin an erroneous communication session by intercepting a message from a current run of the protocol and replaying the corresponding message from a previous secure protocol run for which he has compromised the key. Needham and Schroeder acknowledge this weakness [NEED87] in the protocol and introduce a minor modification inserting a timing mechanism into the protocol to prevent "replay attacks".

In [NEED78], Needham and Schroeder conclude, "... [cryptographic] protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. *The need for techniques to verify the correctness of such protocols is great...*" [emphasis is mine]. It is ironic that this point is illustrated by the flaw detected in their own protocol. For years, computer programmers have relied on layers of testing to address the similar problem of finding errors in programs. Unfortunately, testing of protocols cannot provide a complete answer. Testing looks for errors, but failure to find errors does not mean that errors do not exist [DIJK76]. Accordingly, extensive effort has been focused upon finding methods to verify cryptographic protocols formally. There are many papers in the literature describing techniques proposed to meet the need that Needham and Schroeder identify, [DOL83], [GLAS88], [GNY90], [KEM89], [MEAD89], [MEAD91], [MIL87], [MOS89], [SYV91], [RANG88]. Many techniques are based on creation of an epistemic logic, a logic of knowledge and belief, used to reason about protocols. These logics are modal logics, concerned with truth and falsity in a "possible worlds" scenario. The possible worlds scenario allows the logician to reason more effectively about the real world by emphasizing three possible modal categories of propositions: impossible, possible, and necessary.

While these efforts have contributed significantly to the understanding of cryptographic protocol complexity, no method has yet achieved widespread acceptance and use in verifying cryptographic protocols. In this paper, we propose such a method.

2.0 Logics for Evaluating Cryptographic Protocols

What is quickly becoming the classic paper in protocol verification is by Burrows, Abadi, and

Needham [BAN90]. The “Logic of Authentication” (hereafter referred to as “BAN logic”) they propose is a straightforward mechanism that allows reasoning about beliefs that principals may have during a protocol run. BAN logic has achieved widespread acceptance as the “standard” logic for protocol verification, though it has not achieved widespread implementation. BAN’s strengths lie in the simplicity of its logical language and the small number of axioms. Its main weakness is the fact that it is not complete; that is, while evaluation with BAN logic can detect some errors, protocols with some flaws can pass the BAN logic test.

Other logics have been proposed. Some, as Syverson’s [SYV91], deal with knowledge, others such as BAN, with belief. The key difference between knowledge and belief systems is that knowledge systems have an axiom of the following form: “If you know p, then p is true.” Belief systems do not have this axiom, so belief in p says nothing about the truth or falsity of p. In [SYV91], Syverson argued that logics of belief are not suited for reasoning about security, but can be used to evaluate trust in a protocol, while security as well as trust of cryptographic protocols can be evaluated with logics of knowledge. This topic continues to be a matter of research [SYV93].

Many of the logics designed to verify cryptographic protocols [BAN90], [GNY90], [MOS89] are used in the same three steps:

1 - The protocol is transformed into some “idealized” form
2 - Assumptions are made about the state and environment of the system
3 - A mechanical, step by step analysis of the protocol is conducted

The benefit of these systems lie in the last step. Logical analysis based on an axiomatized language allows a logician to mechanically prove theses specified within the language. Unfortunately, this step is critically dependent upon the first two steps, which are much less effective in meeting their purpose.

Idealization involves translating the meaning of the protocol specification from its expression of the principal’s *actions* to a corresponding expression of the principal’s *beliefs* and intentions. Effecting this translation is more complex than one might expect. There is no mechanism yet developed that allows translation of protocols from actions to beliefs in a formal, mechanical way, largely because global knowledge of the intent of the protocol is required to make this transformation. According to Burrows, et al. “... the idealized form of each message cannot be determined by looking merely at a single protocol step by itself. Only knowledge of the entire protocol can determine the essential logical contents of the message.” [BAN89]. The authors then provide three non-mechanical guidelines for making the idealization transformation:

1 - “... a real message *m* can be interpreted as a formula *X* if whenever the recipient gets *m* he may deduct that the sender must have believed *X* when he sent *m.”*

2 -” Real nonces are transformed into arbitrary new formulas; throughout, we assume that the sender believes these formulas.”

3 - “... for the sake of soundness, we always want to guarantee that each principal believes

the formulas that he generates as messages."

Evaluating cryptographic protocols is further complicated by the fact that protocols are routinely specified in one of several pseudocode languages with no agreed formal definition. Hence, the intent of a principal's actions in the protocol can only be truly known by the protocol specifier. If the interpreter of the protocol accurately judges the meaning of the protocol action, this still does not guarantee that the idealization will accurately reflect what the specifier intended. In fact, it is likely that two different idealizations of the same protocol may exist even though there is no ambiguity in the actions of the principals.

Another shortcoming of the pseudocode languages used to specify cryptographic protocols is that none of them provide mechanisms to assist in the determination of valid assumptions. We propose a language in which assumptions may be formalized in terms of actions of participants. For example, the assumption that two principals establish a private key before a run of the protocol can be expressed as "off the net send and receive" operations. We believe this explicit declaration of actions will formalize and simplify the designation of assumptions.

Another shared characteristic of many existing protocol verification logics is that they are monotonic; that is, knowledge and beliefs cannot be changed or refuted during analysis. Moser [MOS89] proposes a non-monotonic logic that allows reasoning to continue in the presence of new evidence that effectively refutes previously held beliefs/knowledge. She contends that complete knowledge is rarely available and always expensive. Hence, it is not only prudent, but necessary, to be able to continue reasoning if an assumption is refuted. Moser's nonmonotonicity is accomplished by introduction of an "unless" operator, which adds complexity to the logic, and which Syverson contends in [SYV91] is not needed or used. The debate continues.

Snekkenes [SNEK91] identifies the sequencing of the protocol statements in the language as one aspect of protocol verification that is particularly difficult for logics to deal with. BAN Logic, like many of the other proposed logics has no mechanism for enforcing strict ordering on the steps in the protocol. Snekkenes shows that this creates a dilemma for BAN Logic, in that "step permutable" protocols that have even obvious flaws may pass the BAN Logic evaluation. In [BAN90b], Burrows et. al. acknowledge the fact that BAN Logic is not an all encompassing method for ensuring protocol security and state that the problem of providing a mechanism that can detect all errors in a protocol is "quite difficult". This suggests that protocol verification, like program testing, is an intractable problem, and that we should not seek a perfect solution, but attempt to find ways to gain confidence in the security of protocols.

3.0 Reasoning Based on Weakest Preconditions

Our research is motivated from the weaknesses of existing methods of protocol verification. As previously stated, the logics and languages that have been developed require that a protocol specified in an informal "pseudo-code" be transformed into an "idealized" form, i.e., the language of the logic. This idealization cannot be performed mechanically and, in fact, requires intellectual understanding of the goals of the protocol to effectively make the transformation. Our solution begins with the design of a formal language, CPAL, suited to the needs of specification of cryptographic protocols. CPAL gives the designer the constructs needed to effectively express the goals,

actions, and results of protocols while requiring explicit entry of information needed to evaluate the security of the protocols. The semantics of this simple, sequential language in weakest precondition format eliminates the need for idealization and addresses the problem of sequencing of protocol steps.

Our review of the problem of protocol verification has brought us back repeatedly to the field of program verification. Actions of principals in a protocol run can be thought of analogous to operations of programs. For example, users sending and receiving messages can be thought of as performing memory accesses and assignment statements, while creating a key or a nonce is similar to generating a random number, and encrypting/decrypting data may be thought of as structuring data into fields. It is clear that protocols are a small subset of the class of algorithms that are implemented as computer programs. In fact, the subset is small in a number of important ways.

One important distinction is that protocols do not require iteration. A run of a protocol is a sequential process that can be modelled without loop structures. Absence of a loop structure alone reduces complexity significantly. Additionally, branching can be limited to a simple if-then-else structure, eliminating the complex "case" decision structure common to many programming languages. As a last example, we have not seen a need for "procedure calls" in specifying cryptographic protocols, so problems of parameter passing and side effects do not apply. A protocol can simply be thought of as a sequential program, using single assignment of variables with no parameter passing or side effect issues to worry about. For these reasons, we expect that the same methods that are used to verify programs may be even more valuable in evaluating cryptographic protocols.

3.1 The Cryptographic Protocol Analysis Language (CPAL)

Selecting the language constructs is important to our ability to verify the protocols specified in CPAL. The logical languages, such as that in BAN Logic, are concerned with the beliefs of the principals sending messages. We are concerned with determining exactly what *actions* a user can take, on and off the net, to implement a protocol and providing constructs in CPAL that effectively describe those actions. It is from these actions that we can reason about the beliefs of the principals.

We believe the list of actions to be short.

1 - Sending/receiving messages on and off the net
2 - Encrypting/decrypting messages
3 - Creating keys, timestamps, and nonces
4 - Computing functions
5 - Making comparisons and simple decisions

There are two perspectives that must be considered when evaluating the effectiveness of a protocol. The first view we should consider is what the protocol accomplishes with regards to the *trusted principals*. If we do not understand what a protocol accomplishes in a trusted environment, we certainly cannot evaluate what it does in the face of dishonest intruders. In considering the results of proper actions of trusted principals, it is reasonable for us to view and evaluate the

beliefs and actions of both principals during the protocol run, since we are assuming that each principal is doing what they are supposed to be doing. Accordingly, we may desire to check the "belief set" of principals against each other to determine if they correspond. In order to be able to evaluate protocols from this view, it is evident that CPAL should provide a mechanism to allow explicit designation of who accomplishes a specific action. The question of which party created a nonce, for example, can have a major impact on whether or not the nonce accomplishes its objective of ensuring message freshness. Without this feature, it could also be difficult to determine who has jurisdiction over data items or who accomplished the encryption of a data item in a protocol.

The second perspective we must consider is how an *intruder* might be able to compromise the protocol. From this view, it is only reasonable to consider the beliefs of one principal at a time, since the principal cannot assume that they are communicating with a trusted principal. This evaluation must be made based exclusively upon what the principal sends and receives on the net. To accomplish this, each user must be able to remember things that have happened by storing and later retrieving messages, nonces, and keys in a private address space. Otherwise, it will be impossible to effectively reason about what a user knows or believes based on the information gathered via a protocol.

In order to isolate each principal's "belief sets", we place them in a virtual "address space" environment, where each participant can record all data that they acquire, to model the actual user environment where cryptographic protocols are utilized. In such an environment, the user knows very little *a priori*. In fact, all they know is what they see come across the network, along with a few simple protocol rules, and a bit about cryptographic effectiveness. By recording the traffic from the net and analyzing the contents, they can decide what they believe or know to be true. We assume that there is nothing "inherent" in the messages that can give the user any information, and further hold the generally accepted assumption that an intruder has complete access to the net. Intruders can passively listen to all network traffic, or can intercept, modify and reinsert messages, or remove messages without detection. In other words, provision for privacy and integrity of messages and prevention of denial of service are the responsibility of the users and must be implemented in cryptographic protocol/technique if these features are needed or desired.

Aside from its constructs and general structure, in order for CPAL to be useful in verifying protocols there must be a straightforward method to translate protocols specified in any of the popular "pseudo-codes" into CPAL. We should then be able to map statements in the new language directly to their origin in the pseudo code. These characteristics ensure that we do not lose expressiveness and that the language is usable in the cryptographic environment.

The first version of CPAL has been designed. The characteristics just discussed are incorporated into the language. A description of CPAL is provided as Appendix A. CPAL contains the constructs necessary to express the protocols in a form similar to several of the ad hoc pseudocodes that are seen in the literature. To illustrate its expressiveness, several classic and standard protocols have been encoded in CPAL and are listed in Appendix B. The comparative lengths of CPAL and pseudo-code representations are approximately five to one. When it is considered that CPAL allows encoding of assumptions into the protocol while the pseudocodes usually do not, and that most protocols involve less than ten messages, this is an acceptable ratio.

We find that CPAL is flexible in that it allows arbitrary identifiers for keys, parties, and messages and by enforcing few structural requirements. The readability of a protocol definition in CPAL is enhanced by the structure imposed by requiring the identification of the principal conducting each action, and by the selection of mnemonic operators for the send, receive, encrypt, and decrypt operators. Assigning actions directly to principals is also essential to protocol evaluation. It is particularly important in evaluating the security of keys and nonces and in reasoning about the validity of a message based upon who created it. Providing accountability for actions also allows the system to assign values in the appropriate address space so the formal semantic of a statement may use this information.

3.2 Formal Semantics

With the formal mechanism (CPAL) that allows a designer to completely specify protocols is in place, a formal semantic representation of the language can be derived. A classic approach to program verification is based on work by C.A.R. Hoare [HOAR69] in which he describes precondition/postcondition reasoning about programs. Briefly, Hoare suggests that we can define the meaning of a program statement or segment by defining what [pre] conditions must hold such that if that statement or segment is executed, a certain [post] condition will hold after the execution. The notation used to describe that P is a precondition for segment S and postcondition R is:

$$P \{S\} R$$

For example, in order for the program statement "y := sqrt(x)" to cause the postcondition "$x == y^2$", the precondition "$x > 0$" must hold before the statement is executed. Thus,

$$x > 0 \{y := sqrt(x)\} x == y^2$$

One way to use technique is to first select the desired postcondition and work in reverse, evaluating each program segment, in reverse order, to find the preconditions that must hold in order to ensure the postconditions. Those preconditions then become the desired postconditions for the previous program segment. The program proof (or verification) is then completed when the beginning of the program is reached and it is shown that the required preconditions are met in the operating environment.

In [DIJK76] Dijkstra describes an extension of Hoare's precondition/postcondition mechanism called the *weakest precondition* for a program statement or segment. Here, Dijkstra introduces the concept of a "predicate transformer". A predicate transformer is "a rule telling us how to derive for any [arbitrary] post-condition R the corresponding weakest precondition for the initial state such that activation will lead to a properly terminating activity that leaves the system in a final state satisfying R." The weakest precondition Q for segment S and postcondition R is written:

$$wp(S, R) \equiv Q$$

The precondition is considered to be weakest if its predicate transformer describes *all* states that enable the program segment to result with the desired outcome. By describing preconditions in

their weakest form, the most general case is represented, a fact which we can use in the final proof step (showing that current conditions satisfy those preconditions). All that is required is to show that the derived weakest preconditions are implied by the assumptions of the start state. Much of this process may be accomplished by substitution of identities.

Weakest preconditions are often recursively defined; that is, the weakest precondition for one segment may be dependent upon, or defined in terms of one or more weakest preconditions of other segments. As an example, the weakest precondition definition of the IF statement may be expressed as:

$$wp(\text{if } C\ S1, Q) \equiv (C \supset wp(S1,Q)) \Lambda\ (\sim C \supset wp(skip,Q)) \qquad (1.0)$$

The weakest precondition mechanism for expressing formal semantics of languages has other properties that may be advantageous for describing CPAL. First, with weakest precondition reasoning, the “idealization” process of transforming the protocol actions into corresponding beliefs of principals is not needed. As noted earlier, idealization does not lend itself well to mechanical or formal methods. One of the reasons for this is that the actions of the principals are accomplished locally, while the beliefs of the participants are based on global knowledge. Our method eliminates the human error possible from the transformation process. Furthermore, this mechanical transformation supplants the need for interpretation of the user’s specification by a “systems specialist”. The user is responsible for specifying the protocol, and the tool will provide a mechanical evaluation of how well the protocol meets its desired ends.

Secondly, one of the key shortcomings of existing logical evaluation tools is the difficulty in dealing with message sequencing in the protocol. Since strict sequencing cannot be enforced, the logical evaluation considers actions as timeless, when in truth, the sequence in which the actions occur are essential to the security of the protocol. If there exists an ordering of the protocol steps that does produce a secure protocol, then the protocol will pass the BAN logic evaluation regardless of whether the steps are correctly ordered in the evaluated protocol [SNEK91]. In his critique of BAN logic, Nessett [NESS90] argues that this is a critical flaw, since the positive evaluation may give a user a false confidence in the flawed protocol.

The weakest precondition methodology provides an inherent sequencing mechanism for protocols by allowing us to describe the statement separator as a sequence discriminator. This is accomplished by defining statement concatenation recursively as:

$$wp(S1; S2, R) \equiv wp(S1, wp(S2, R)) \qquad (1.1)$$

The recursive nature of this definition forces evaluation of the statements in the order specified, so a different ordering of the statements will result in a different semantic evaluation.

We have not yet seen a logical construct that can effectively resolve this sequencing problem. One logic [GNY90] addresses this notion with weak temporal operators such as a “M not originated here” and a “P once said M” operator on messages, and by assigning precondition status to beliefs. A common critical view of using temporal constructs is that they complicate the language of the logic significantly. This results in specifications that are harder to implement and proofs that

are more complex. Accordingly, the logics are hardly usable. A key to the acceptance of BAN logic is that it is simple, with few postulates or constructs.

A third advantage to the weakest precondition method is that we anticipate that the logic of reasoning we use will be simple. We project that the logic that we will use as the target for the formal semantic representation of CPAL will incorporate only the classical logical operators: and, or, not, and quantification.

Finally, the use of weakest preconditions for program verification is well understood from its use in the software field. There is much literature describing this approach [DIJK76], [WULF81], and several implementations available for study [GOOD78], [HOA73] as a foundation for our work. Many of the definitions have already been given, such as the meaning of concatenation of statements, the IFTHENELSE structure, and statement concatenation definitions which are described in [WULF81]. One important operator to be defined is NEW. NEW plays a large part in establishing security of keys and determining jurisdiction of data items. There is a definition of the NEW statement in Hoare and Wirth's description of the Pascal language [Hoar83], in terms of the implementation of a pointer. Our concept of the NEW operator is more practical, reflecting the "non-existence" of a value prior to its creation with NEW. Other definitions such as for the encrypt, decrypt, send, receive, accept, and reject are yet to be determined.

An important consideration of the formal semantics of each statement is the ability to represent the accountability for actions described above. By using "dot" notation, we will represent the data items stored in the user's address space. Hence we can represent key "kab" in A's address space as A.kab, or supposedly the same key in B's address space as B.kab. This information can easily be gathered by the syntax checker and later utilized by the verification condition generator.

3.3 The Verification Condition Generator

Once the syntax and semantics of CPAL are formalized, the next logical step is to build a Verification Condition Generator (VCG). The VCG will take the protocol specification as input, and from it construct the weakest precondition that must hold for an arbitrary postcondition. The process will begin by expanding each protocol step using the weakest precondition definition for concatenation. Subsequent "passes" will utilize the remaining statement definitions to expand or reduce the output of previous passes until the definition reaches its final form, containing only comparisons and operations on arbitrary predicates. The definition is not circular since CPAL is not a recursive language, nor is there any looping structure. Each reduction is simpler than the statement being reduced, so the process will eventually terminate and the protocol will reduce to statements such as variable replacement of the assignment statement. Consider the protocol

S == {M1 := M2; if C then M3 := M4;}

Suppose we define the meaning of the assignment statement to be:

$$wp(M1:=M2, Q(M1)) \equiv Q(M2) \qquad (1.2)$$

Based on definitions 1.0, 1.1, and 1.2, the VCG will expand protocol S as follows for the arbitrary

postcondition Q dependent upon variables M1 and M3:

$$\begin{aligned}
&wp(S,Q) \equiv wp(M1 := M2,\ wp(\text{if } C \text{ then } M3 := M4, Q(M1,M3))) && (1.1)\\
&\quad \equiv wp(M1 := M2,\ (C \supset wp(M3 := M4, Q(M1,M3))) \Lambda (\sim C \supset Q(M1,M3)) && (1.0)\\
&\quad \equiv wp(M1 := M2,\ (C \supset Q(M1,M4) \Lambda (\sim C \supset Q(M1,M3))) && (1.2)\\
&\quad \equiv (C \supset Q(M2,M4) \Lambda (\sim C \supset Q(M2,M3))) && (1.2)
\end{aligned}$$

The simplified result, $P \equiv wp(S,Q)$, of the VCG represents the semantic definition of the specified protocol. For any arbitrary predicate Q, if P holds before execution of the protocol, Q will hold after the execution is complete. Due to the definition of concatenation, P will be expressed in terms of Q. In an actual proof of the security of a protocol we will replace Q with the goals of the protocol, creating P', and then prove that the assumptions imply P'. "Constructing these proofs is a matter of logic and mathematics" [WULF81]. In the previous example, the verification condition can be easily examined by inspection. Clearly, more complex protocols may produce verification conditions that are difficult to evaluate. Much research has been devoted to establishing techniques for conducting and simplifying such proofs. Theorem provers are one example of the results of this research. It may be possible to utilize an existing mechanism to mechanically simplify the final proof step of our protocol verification process. It is envisioned that this tool would mechanically reduce identities and identify any contradictions that may exist. Though such a mechanical process is essential to use of weakest precondition reasoning for program verification, it is not considered to be an essential step to the success of this protocol verification method since cryptographic protocols are much shorter than programs, rarely involving more than ten messages.

Another important aspect of the final proof step is the selection of a representation for the goals and assumptions of the protocol. Though the logic we use may be similar to an existing logic, such as BAN logic, we may be able to reason sufficiently using only the classic logical operators and, or, and quantification, and without such extensive constructs. The essence of the BAN Logic language is the function of the belief predicate which allows users to infer beliefs based upon messages transmitted from other users. BAN Logic is a monotonic logic of belief. It does not preclude a principal from believing a predicate that may in fact be false, and once a belief is attained, it may not change. It may be necessary or advantageous to select a logic of knowledge for our evaluation, such as Syverson's Formal Semantics [SYV91], which precludes erroneous beliefs. It also does not allow predicates to change their truth values. We may also desire to use a nonmonotonic logic of belief such as Moser's [MOS89], which allows new information to change the truth values of predicates and thus modify previously held beliefs. A strength of our system, is that any of these logics may be used once the semantic representation of the protocol is established by the VCG derivation.

It should be pointed out that this evaluation is based exclusively on the actions that the principals take during the protocol run. The address spaces of the principals are compared to determine if each has enough information to claim the desired knowledge. The use of dot notation allows for reasoning about data items in different address spaces, and information in address spaces is changed only based on the actions that occur due to execution of the protocol. There is no requirement for non-mechanical interaction to determine the meaning of the protocol since no idealization is required.

One potential drawback to “weakest precondition” reasoning, is that it can become rapidly intractable in the size and complexity of proofs it generates. We believe that evaluation of the classic protocols will confirm that the proofs generated from our system will be manageable from the size and complexity viewpoints.

4.0 Summary

In this paper, we have proposed a cryptographic protocol specification language and a methodology for establishing a formal semantic of the language. We believe this semantic will allow us to more effectively apply existing evaluation tools to learn more about protocols. Based on the sequencing capabilities of our method, we believe our system will be able to effectively deal with the BAN problem pointed out by Nessett [NESS90]. When sequencing of the steps is forced, the actions taken in the protocol will determine if the assumption of a good key is valid [BAN90b].

Other interesting and useful tools may emerge from this research effort. Clearly, it would be beneficial to have a pseudocode interpreter that would automatically translate protocols into CPAL. This interpreter could accept a wide variety of constructs common to several pseudocodes or there may be multiple interpreters for different pseudocode “versions”.

Another highly practical tool would be a translator that takes the specification of the protocol and produces the C (or other common language) code to implement the protocol specified. This would provide the protocol designer with a complete protocol implementation system that would take their protocol from a pseudocode version, evaluate it with regard to its environment and goals, and mechanically produce executable code that exactly implements his design.

Finally, it may be useful to provide an interface from this tool to other existing tools for protocol evaluation. Potentially, this tool could one day be part of a protocol evaluation workbench which includes our specification/verification system, a testing system (such as Interrogator [MIL87]), a BAN implementation (such as [CHEN90]), etc. Another sub-component of such a system would be a language translator to take a protocol specified in CPAL and convert it to the language used by Interrogator to provide a fully integrated environment.

Acknowledgments

Many thanks to Katie Oliver, Ramesh Peri, and Sally McKee for their insightful comments on the content and structure of this paper.

Bibliography

[AT91] Martin Abadi and Mark R. Tuttle, “A Semantics for a Logic of Authentication”, Tenth Annual ACM Symp on Princ of Dist Computing, Montreal, Canada, August, 1991

[BAN90] Burrows, M., Abadi, M., and Needham, R. M. “A Logic of Authentication”, ACM Transactions on Computer Systems, Vol. 8, No. 1, Feb 1990, pp. 18-36.

[BAN90b] Burrows, M., Abadi, M., and Needham, R. M., “Rejoinder to Nessett”, ACM Operating Systems Review, vol. 24, no. 2, April 1990, pp. 39-40

[BIRD92] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. “Systematic Design of Two-Party Authentication Protocols.” In Joan Fegenbaum, editor, Advances in Cryptology - CRYPTO ‘91, volume 576 of Lecture Notes in Comuter Science. Springer Verlag, Berlin, 1992

[CHEN90] Cheng, Pau-Chen and Gligor, Virgil D. “On the formal specification and verification of a Multiparty Session Protocol”. From 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 216-233

[DENN81] D. E. Denning and G. M. Sacco, “Timestamps in key distribution protocols,” Communications of the ACM, vol. 24, no. 8, Aug 1981, pp. 533-536

[DIJK76] Edsger W. Dijkstra, “A Discipline of Programming”, Prentice Hall Series in Automatic Computation, Prentice-Hall Inc. Englewood Cliffs, NJ, 1976

[DOL83] Dolev, D., and Yao, A.C. “On the security of public key protocols”. IEEE Trans. Inf. Theory IT-29, 2(Mar. 1983), pp. 198-208

[GLAS88] Janice I. Glasgow and Glenn H. MacEwen, “Reasoning About Knowledge in Multilevel secure Distributed Systems”, in Proceedings of the 1988 IEEE Syposium on Security and Privacy, Washington (IEEE), 1988, pp. 122-128

[GMW86] O. Goldreich, S. Micali, and A. Wigderson, “Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design”, Proceedings of the 27th IEEE Symposium on foundations of Computer Science, 1986, pp. 174-187

[GOLD85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, “The Knowledge Complexity of Interactive Proof Systems,” Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 291-304

[GOLD89] Goldwasser, Shafi, Micali, Silvio, and Rackoff, Charles, “The Knowledge Complexity of Interactive Proof Systems,” Siam Jrnl of Comp, Vol 18, No 1, Feb 1989, pp. 186-208.

[GNY90] Gong, L., Needham, R., and Yahalom, R. “Reasoning about Belief in Cryptographic protocols”. From 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 234-248

[GOOD78] D I. Good et. al. Report on the language Gypsy - version 2.0, Univ. of Texas at Austin, Certifiable Minicomputer Project, Report ICSCA-CMP-10, 1978

[HAL88] Halpern, J.Y., and Moses, Y.O. “A Knowledge-based analysis of zero knowledge” (preliminary report). In Proceedings of the 20th ACM symposium on Theory of Computing (CHicago, Ill, May 1988), ACM, New York, 1988, pp. 132-147

[HOAR69] C.A.R. Hoare, “An Axiomatic Basis for Computer Programming”, Communications of the ACM, Vol 12, Number 10, Oct 1969

[HOAR73] Hoare, C. A. R., and N. Wirth: “An Axiomatic Definition of the Programming Lan-

guage Pascal," Acta Informatica, 2, 1973

[HOAR78] C. A. R. Hoare, "Communicating Sequential Processes", Communications of the ACM, Vol 21, Number 8, Aug 1978, pp 666-677

[HOAR85] C. A. R. Hoare, "Communicating Sequential Processes", Prentice Hall, 1985

[KEM89] R. A. Kemmerer, "Using Formal Methods to Analyze Encryption Protocols," IEEE Journal on Selected Areas in Communications, vol. 7, mo. 4, pp. 448-457, May 1989

[LAMP91] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice", ASM OS Review, Vol 25, No. 5, Special Issue, Proceedings of the 13th Symposium on Operating System Principles, 13-16 Oct 1991, pp 165-182

[MEAD89] Meadows, C., "Using Narrowing in the Analysis of Key Management Protocols". From 1989 IEEE Symposium on Research in Security and Privacy, pp. 138-147.

[MEAD91] Meadows, C., "A System for the Specification and Analysis of Key Management Protocols". From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 182-195.

[MIL87] Millen, J.K., Clark, S. C., and Freedman, S. B. "The interrogator: Protocol security analysis". IEEE Trans. Sofw. eng. SE-13, 2(Feb. 1987), pp. 274-288

[MOS89] L. Moser, "A Logic of Knowledge and Belief for Reasoning about Computer Security" in Proceedings of the Computer Security Foundations Workshop II, Washington (IEEE), 1989, pp. 57-63

[NEED78] Needham, R. M., and Schroeder, M. D. "Using encryption for authentication in large networks of computers". Commun. ACM 21, 12 (Dec. 1978), pp. 993-999

[NEED87] Needham, R.M. & Schroeder, M.D., "Authentication Revisited", ACM Operating Systems Review, Vol. 21, No. 1, January 1987.

[NESS87] D. M. Nessett, "Factors Affecting Distributed System Security", IEEE Trans On SWE, Vol. SE-13, No 2, Feb 1987, pp. 204-222

[NESS89] D. M. Nessett, "Layering Central Authentication on Existing Distributed System Terminal Services", From 1989 IEEE Symposium on Security and Privacy, pp. 290-299.

[NESS90] D. Nessett, "A Critique of the Burrows, Abadi, and Needham Logic", ACM Operating Systems Review, vol. 24, no. 2, April 1990, pp. 35-38

[OTWY87] Otwy, D., and Rees, O. "Efficient and timely mutual authentication". Operating Systems Review 21, 1(Jan. 1987), pp. 8-10

[SNEK91] Snekkenes, E., "Exploring the BAN Approach to Protocol Analysis". >From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 171-181.

[SYV91] Syverson, P., "The Use of Logic in the Analysis of Cryptographic Protocol"., From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 156-170.

[SYV91] Syverson, P., “The Use of Logic in the Analysis of Cryptographic Protocol”., Journal of Computer Security, 1 (1992), 317-334, IOS Press.

[RANG88] P. Venkat Rangan, “An Axiomatic Basis for Trust in Distributed Systems”, in Proceedings of the 1988 IEEE Symposium on Security and Privacy, pp. 204-211, IEEE Computer Society Press, Washington, DC, 1988

[VOY83] Victor L. Voydock and Stephen T. Kent, “Security Mechanisms in High-level Network Protocols”, Computing Surveys, Vol 15, No. 2, June 1983, pp. 135-171

[WULF81] Wm. A. Wulf, Mary Shaw, Paul N. Hilfinger, and Lawrence Flon, “Fundamental Structures of Computer Science” Addison-Wesley, 1981

[WULF93] Wm. A. Wulf, Alec Yasinsac, Katie S. Oliver, and Ramesh Peri, ”Remote Authentication Without Prior Shared Knowledge”, UVA Dept of Computer Science, Technical Report CS-93-37.

APPENDIX A - Description of CPAL

The purpose of this language is to provide protocol developers a language that is sufficiently rich to express cryptographic protocols, yet small enough to allow creation of an easily understood, unambiguous formal semantic for the language. To make the language expressive, we allow arbitrary selection of identifier names for principals, messages, nonces, and keys. The operators are designed to allow expression of the actions of principals involved in a protocol run. The operators in CPAL provide for sending, receiving, encrypting, and decrypting of messages; generating keys, nonces, and timestamps; computing functions; and making decisions using an IFTHENELSE structure. This language allows explicit expression of assumptions by providing send and receive off-line operators so *no* prior knowledge by participants need be assumed.

Since we have not encountered a protocol as yet that requires iteration, we do not provide any mechanism to express it in this language. As a brief description of the language, the following notes are offered:

> Since all interactions between participants must be explicit. Each protocol step will be labeled with the ID of the user taking the action.
>
> Messages are sent/received via the snd/rcv operators. The intended destination for a sent message is separated from the messages by a vertical line. No identifier is allowed to represent network provided identification of the sender for received messages. The identification of the sender must be determined by the cryptographic method.
>
> The format for encryption/decryption is an ad hoc standard of preceding curly braces with ‘e’ or ‘d’ with the message identifier within the braces. We added the backslash to designate that the following identifier is the key for the previous encryption/decryption.
>
> Keys can be represented by an arbitrary identifier. For purposes of convention, we allow the characters ‘+’ and ‘-’ to suffix identifiers which could be used to represent public and

private keys. For example, user A's public/private key pair might be ka+ and ka-.

Literals are not used in this language.
Concatenation of messages is signified by an optional comma.
Statements in the protocol are separated by semicolons.
':=' is the assignment operator, '==' is the comparison operator.

```
A: snd(B| M1);                          [User A sends message M1 addressed to user B]
B: rcv(M1);                             [User B receives message M1 on the net]
A: sndoff(B| M2);                       [User A sends message M2 to user B off the net]
B: rcvoff(M2);                          [User B receives message M2 off the net]
A: M4 := e{M3}\kab;                     [A encrypts M3 under A & B's private key k into M4]
A: If (V1 == V2) then stmt else stmt    [If-then-else statement]
A: new(N1);                             [N1 is a new nonce, ie. A has jurisdiction over N1]
A: Y :=  f(X);                          [Compute a function of X, store in Y]
```

APPENDIX B - Examples of Protocols Expressed in CPAL

Needham, R. M., and Schroeder Private Key Protocol [NEED78]

A->B: A,B,I_{A1}

AS->A: $\{I_{A1},B,K_{ab},\{K_{ab},A\}^{KB}\}^{KA}$

A->B: $\{K_{ab},A\}^{KB}$

B->A: $\{I_B\}^{Kab}$

A->B: $\{I_B\text{-}1\}^{Kab}$

Fully described protocol

```
S: new(f);      S: sndoff(A,B| f);  A,B: rcvoff(f);
S: new(kas);    S: sndoff(A| kas);  A: rcvoff(kas);
S: new(kbs);    S: sndoff(B| kbs);  B: rcvoff(kbs);
A: N1 := new();
A: snd(S| A,B,N1);              -- A->AS:  A,B,I_A1
S: rcv(Srcid,Destid,N1);
S: kab := new();
S: snd(Srcid| e{N1,Destid,kab,e{kab,Srcid}\kbs}\kas);
                        -- AS->A: {I_A1,B,K_ab,{K_ab,A}^KB}^KA
A: rcv(Msg1);
A: (N1',Destid,kab,ticket) := d{Msg1}\kas);
```

```
A: if (N1' == N1) then
          accept(kab);
          snd(B| ticket);     -- A->B: {K_ab,A}^KB
   else reject;
B: rcv(ticket);
B: (kab,Srcid) := d{ticket}\ksb;
B: N2 := new();
B: snd(Srcid| e{N2)}\kab);   -- B->A:{I_B}^Kab
A: rcv(Msg2);
A: N2 := d{Msg2}\kab);
A: snd(B| e{f(N2)}\kab;       -- A->B:  {I_B-1}^Kab
B: rcv(Msg3)
B: N3 := d{Msg3}\kab;
B: if (N3 == f(N2)) then accept; else reject;
```

DENNING & SACCO PRIVATE KEY PROTOCOL [DENN81]

A->AS: A,B

AS->A: $\{B,K_{ab},T1,\{A,K_{ab},T1\}^{KB}\}^{KA}$

A->B: $\{A,K_{ab},T1\}^{KB}$

B->A: $\{I_B\}^{Kab}$

A->B: $\{I_B\text{-}1\}^{Kab}$

Fully described protocol

```
S: new(f);     S: sndoff(A,B| f);  A,B: rcvoff(f);
S: new(kas);   S: sndoff(A| kas);  A: rcvoff(kas);
S: new(kbs);   S: sndoff(B| kbs);  B: rcvoff(kbs);
A: snd(S| A, B);                   -- A->AS: A,B
S: rcv(Srcid,Destid);
S: new(T1);
S: new(kab);
S: snd(Srcid| e{Destid,kab,T1,e{T1,kab,Srcid}\kbs}\kas)
                              -- AS->A:{B,K_ab,{T,K_ab,A}^KB}^KA
A: rcv(Msg1);
A: (Destid,kab,T1,Ticket) := d{Msg1}\kas
```

```
A: if (T1 == f(time) Λ (Destid == B)) then
       accept(kab);
      snd(Destid| Ticket);   --A->B:{A,K_ab,T1}^KB
   else reject;
B: rcv(Ticket);
B: (T1,kab,Srcid) := d{Ticket}\kbs;
B: if (T1 == f(time)) then
       new(N1); snd(Srcid| N1);  -- B->A:{I_B}^Kab
   else reject;
A: rcv(Msg2);
A: N1 := d{Msg2}\kab
A: snd(B| e{f(N1)}\kab);
B: rcv(Msg3);
B: N2 := d{Msg3}\kab
if (N1 == f(N2)) then accept(kab); else reject;
```

OTWAY AND REES PRIVATE KEY PROTOCOL [OTWY87]

```
A->B:  C,A,B,{R_1CAB}^K1
B->AS: C,A,B,{R_1,C,A,B}^K1,{R_2,C,A,B}^K2
AS->B: C,{R_1,K_C}^K1,{R_2,K_C}^K2
B->A:  C,{R_1,K_C},^K1
```

Fully described protocol

```
S: kas := new(); S: sndoff(A| kas); A: rcvoff(kas);
S: kbs := new();S: sndoff(B| kbs);  B: rcvoff(kbs);
A: N1 := new();
A: C := new();
A: snd(B| C,'A','B',e{N1,'C','A','B'}\kas);
                                --A->B:C,A,B,{R_1,C,A,B}^K1
B: rcv(C,Src,Dst,Ticket1);
B: N2 := new();
B: Ticket2 := e{N2,C,A,B}\kbs;
B: snd(S| C,Src,Dst,Ticket1,Ticket2);
S: rcv(C,Src,Dst,Ticket1,Ticket2);
S: (NA,CA,SrcA,DstA) := d{Ticket1}\kas;
```

```
S: (NB,CB,SrcB,DstB) := d{Ticket2}\kas;
S: if (C <> CA) or (C <> CB) or (Src <> SrcA) or
       (Src <> SrcB) or (Dst <> DstA) or (Dst <> DstA)
       reject;
S: kab := new();
S: PassA := e{NA,kab}\kas;
S: PassB := e{NB,kab}\kbs;
S: (B| C, PassA, PassB);
B: rcv(C1,PassA, PassB);
if (C1 <> C) reject;
B: (N21,kab) := d{PassB}\kbs
B: if (N21 <> N2) reject; else accept;
B: snd(A| C, PassA);
A: rcv(C1, PassA);
A: (N11,kab) := d{PassA}\kas
A: if (N11 <> N1) reject; else accept;
```