Proving Correctness of the Basic TESLA Multicast Stream Authentication Protocol with TAME*

Myla Archer

Code 5546, Naval Research Laboratory, Washington, DC 20375
E-mail: archer@itd.nrl.navy.mil

The TESLA multicast stream authentication protocol is distinguished from other types of cryptographic protocols in both its key management scheme and its use of timing. It takes advantage of the stream being broadcast to periodically commit to and later reveal keys used by a receiver to verify that packets are authentic, and it uses both inductive reasoning and time arithmetic to allow the receiver to determine that an adversary cannot have prior knowledge of a key that has just been revealed. While an informal argument for the correctness of TESLA has been published, no mechanized proof appears to have previously been done for TESLA or any other protocol of the same variety. This paper reports on a mechanized correctness proof of the basic TESLA protocol based on establishing a sequence of invariants for the protocol using the tool TAME, an interface to PVS specialized for proving properties of automata. It discusses the organization and process used in the proof, and the possibilities for reusing these techniques in correctness proofs of similar protocols, starting with more sophisticated versions of TESLA.

1. Introduction

Much attention is paid to the design of protocols; in fact, the majority of papers encountered when searching through the protocol literature describe designs for new protocols. Belief in the correctness of these protocols is usually based on informal arguments. However, formal analysis of protocols has often revealed problems: see, e.g., [8]. Thus, formal analysis of protocols is important in providing better assurance of their correctness.

Several types of tools have been applied to the automated analysis of protocols. The NRL Protocol Analyzer [10] and the HOL-based AAPA2 [4] are special purpose tools for protocol analysis. The analysis done in [8] used FDR, a model checker for CSP, which has been successfully used in the analysis of several protocols. In addition, methods have been developed for applying several general purpose theorem provers to proving correctness properties of protocols: e.g., NuPrl in [7], PVS in [5] and [11], and Isabelle in [12]. These tools and methods are generally designed to make the analysis as automatic as possible, though some user guidance can be required.

In the TESLA protocol for multicast stream authentication [13], authentication is based on the absolute timing of the publication of keys and the indirect relation of each new n-th key to an original key commitment. This kind of de-

* This work is funded by the Office of Naval Research.

pendence on time and indexing distinguishes TESLA from other cryptographic protocols. Protocols that publicly reveal keys at strategic times can be expected to have analogous correctness arguments, and hence a formal and mechanically checked proof of one such protocol is expected to be useful for guiding the development of correctness proofs for similar protocols. To our knowledge, no one has previously produced a mechanized proof for a protocol of this class. Given the complexity of this class of protocols, it is unlikely that any tool can prove the correctness of such a protocol totally automatically (i.e., without user guidance). Model checking this kind of protocol is not feasible because an infinite state system is required to represent the inductive relationship between an arbitrary n-th packet and the initial packet. Mechanical theorem proving without user guidance is also problematic because the timing computations involve nonlinear real arithmetic, for which there are no decision procedures.

Reference [13] describes the basic TESLA protocol and several increasingly sophisticated variants. This paper reports on a mechanically checked proof of the correctness of the basic TESLA protocol using the tool TAME [3,2]. TAME is an interface to PVS that simplifies specifying and proving properties of automata. The basic TESLA correctness proof is based on the method demonstrated in [6]: 1) model the system being studied as a Lynch-Vaandrager (LV) timed automaton [9], 2) express any desired system property as a state invariant, and 3) establish the validity of the state invariant by developing auxiliary invariants that support its proof. (The method also includes refinement or simulation proofs between automata, but these were not needed for TESLA.) The mechanized proof for basic TESLA bears a clear relationship to the informal proof in [13] but fills in missing details.

The paper is organized as follows. Section 2 describes the basic TESLA protocol. Section 3 describes how TAME supports specifying and proving properties of LV timed automata. Section 4 describes how the basic TESLA protocol was modeled as an LV timed automaton in TAME. Section 5 describes how the basic TESLA correctness property was formulated as a state invariant and proved using a set of auxiliary invariants. Section 6 describes the use of TAME in establishing the correctness of basic TESLA. Finally, Section 7 discusses the manner in which the specification and proof of basic TESLA can likely be recycled to produce correctness proofs of more sophisticated TESLA versions.

2. The Basic TESLA Protocol

Every variant of the TESLA multicast stream authentication protocol assumes a single sender who is broadcasting a more or less continuous stream of packets. Because new packets are continually arriving, a receiver can use information in later packets to authenticate earlier packets. Each packet contains a `MAC` (message authentication code) used by the receiver to authenticate the packet.

The `MAC` is a value computed by applying a MAC function known to the receiver to the remainder of the packet content and some encryption key. Thus, when the receiver learns the key, he can compute the proper value of the `MAC` from the packet content and determine whether it matches the `MAC` value sent with the packet. In the case of a match, the receiver can consider the packet authentic provided there is assurance that an adversary could not have used the proper key to create a forged packet.

The receiver obtains the key `k` needed to compute the `MAC` of a packet as part of the content of some later packet. To ensure that the packet being authenticated has not been forged by an adversary using the key `k`, `k` is required to be a fresh key revealed only after the receiver is expected to have received the packet being authenticated. Further, by requiring that the sender must also have included a commitment to use `k` in a packet previous to the packet to be authenticated, the adversary is prevented from fooling the receiver by simply choosing a fresh key $\mathtt{k}'$, sending a forged packet, and then revealing $\mathtt{k}'$ in a later forged packet. The commitment to `k` is computed by applying a pseudo-random function, known to the receiver, to `k`; thus, the receiver can check when it receives `k` that `k` is the key committed to.

Since a pseudo-random function is effectively impossible to invert, the key `k` cannot be computed from the commitment to `k`. The MAC function is likewise designed to be effectively impossible to invert. Therefore, neither broadcasting the commitment nor broadcasting the `MAC` value reveals `k`. The informal justification for the correctness of the protocol is that the key to the `MAC` of a packet `p` from the sender is never known to an adversary at the time `p` is received by a receiver, and therefore cannot have been used by the adversary to create an acceptable `MAC` to include in a forged substitute for `p`.

Of course, every key commitment needs to be in an authenticated packet not forged by an adversary. Thus, the whole authentication scheme in TESLA must be bootstrapped by guaranteeing that the initial packet is authentic. This is assumed to be done by the sender using the more expensive method of digitally signing the first packet.

Basic TESLA is the simplest version of the protocol. In basic TESLA, the encryption key for the `MAC` of the i-th packet is committed to in the $(i-1)$-st packet and revealed in the $(i+1)$-st packet. To allow the receiver to determine whether the i-th packet arrives before the $(i+1)$-st packet has been sent, the packets are sent at regular intervals of length `I`. A typical packet from the sender in basic TESLA contains:

1. the message to be delivered,
2. a commitment to the key to be used to encode the `MAC` of the next packet,
3. the key that was used to encode the `MAC` of the previous packet from the sender, and
4. the `MAC` of the current packet.

Two exceptional packets are the initial packet, which is assumed to be digitally signed and serves only to commit to the key for the second packet, and the second packet, which need not reveal a key. Thus, all packets carry a key commitment, and every packet from the third packet on reveal a key.

Under basic TESLA, a receiver can authenticate the i-th packet `p` when:

- `p` has been received;
- either `p` is the digitally signed initial packet, or else the preceding packet `p1` and succeeding packet `p2` have been received, and the following hold:
  - applying the MAC function to the key revealed in `p2` and the content (other than the `MAC`) of packet `p` yields a value equal to the `MAC` component of `p`,
  - the key revealed in `p2` is the key committed to in `p1`,
  - `p1` can be authenticated, and
  - $ArrT_i < T_{i+1}$, where $ArrT_i$ is the receive time of the i-th packet `p` and T_{i+1} is the (earliest possible) send time of the $(i+1)$-st packet `p2`, as measured on the receiver's clock.[1]

3. Reasoning about LV Timed Automata in TAME

TAME [3,2] is an interface to PVS [14] that simplifies specifying and proving properties of automata. To make it simpler to specify automata, TAME provides specification templates for various classes of automata, including LV timed automata. To make it simpler to prove properties of automata, TAME provides a set of proof steps, implemented as PVS strategies, that mimic typical steps used in hand proofs of automata properties. The TAME proof steps go a long way towards relieving the user from low-level reasoning in verifying state invariants. However, the rich type system of PVS, which permits predicate subtypes, can result in *type correctness conditions* (TCCs) being produced both when a specification is type checked and when instantiations of quantified variables are performed during proofs. In the case of TESLA, proving these type correctness conditions can require the user to guide the prover through basic reasoning about non-emptiness of lists and about nonlinear arithmetic. This will be discussed further in Section 6.

Lynch-Vaandrager timed automata [9] are transition systems defined by a set of states, a distinguished subset of start states, and a set of actions that cause state transitions. The basic part of each state is represented by the values of a set of state variables. Each state s also has associated timing information: a current time $\texttt{now}(s)$, and time bounds $\texttt{first}(s)(a)$ and $\texttt{last}(s)(a)$ on each action a which describe the next time interval in which a must execute. If there is no particular schedule for a, these time bounds are 0 and ∞. The time bounds for any a can be reset by other actions. Associated with each action is a precondition describing when it is enabled, and an effect describing the resulting change in state. There is always a special time-passage action $\nu(\Delta t)$ which causes time to advance by

[1] This condition is formulated slightly differently than in [13] but is equivalent; see Section 4.

amount Δt. The TAME template for LV timed automata contains declarations of all of the above automaton features, together with any standard parts of their definitions.

4. Modeling Basic TESLA in TAME

Most protocols, including TESLA, can be modeled by an automaton whose initial state is modified by the actions of the participants. Because the TESLA protocol depends on measuring time passage, it is natural to model it as a timed automaton. The participants in a multicast stream authentication protocol include the sender, one or more (intended) receivers, and one or more possible adversaries. Adversaries in TESLA are assumed to have full power over the network [13]: they can allow packets to go through as sent and in a timely fashion, but they can also delay or block packets, modify or replace packets, or flood the network with packets. Thus, TESLA provides no guarantees against denial of service; it is only intended to provide a means for guaranteeing authenticity of packets.

This section discusses the assumptions made about the power of the adversary and other issues in the TAME model of basic TESLA and how the basic TESLA actions, states, and timing constraints are represented in the model. The actual TAME specification of TESLA can be found in [1].

Assumptions made in the model. In modeling TESLA in TAME, several simplifying assumptions were made. First, because collusion among adversaries does not lead to additional power, and because receivers act independently of one another, only one adversary and one receiver are modeled. Second, rather than computing packet receive time on the receiver's clock and send time on the sender's clock as in [13], both receive time and send time are computed on the receiver's clock, with the receiver always assuming the worst case (earliest) possible send time consistent with synchronization information. Third, we omit use of the second pseudo-random function used in [13] to compute a "secret key" for the `MAC` from the revealed key, as we fail to see how this provides any additional security.

Because the receiver must know the indices of the various packets received to authenticate a packet, another assumption is that the index of the packet is part of its content. The initial packet, digitally signed by the sender, is assigned index 0. The value 0 is not considered a valid index, but is used only to identify the initial packet for purposes of proof. The initial 0-th packet is assumed to be sent at time $T_0 = 0$.

Further assumptions clarify the power of the adversary. The adversary is assumed to send well-formed packets containing a valid message index (i.e., not 0), a message body, a key commitment, a key, and a MAC computed from some key and the message body. The sender and the adversary are assumed to start off with non-overlapping sets of keys which they know and can use. The keys available to the adversary are assumed to be only the adversary's initial set of keys plus any keys that have been revealed by the sender, and any key commitment available to the adversary is either a commitment already sent by the sender or a commitment computed from some key available to the adversary. As noted in Section 2, key commitments are created by applying a pseudo-random function to the key being committed to; like any receiver, the adversary is assumed to have knowledge of this pseudo-random function.

Finally, the usual assumptions are made that events of very low probability are actually impossible. These assumptions are represented in TAME in two ways. First, the precondition of every adversary action prevents the adversary from using a key committed to but not sent by the sender, meaning the adversary is unable to invert the pseudo-random function. Second, axioms state that both the pseudo-random function used to compute key commitments and the MAC function are uniquely invertible, which implies that the adversary cannot use substitute keys to create matches to encrypted authentication information used by the sender.

TESLA actions in TAME. Each participant in TESLA performs actions characteristic to the role of that participant. The sender sends packets at regular intervals, the adversary sends packets at will, and the receiver may receive any packet sent by either the sender or the adversary. In TAME, these actions are represented as parameterized automaton actions in which the parameters are used to construct or designate packets sent or received. The precondition of the sender action is used to enforce the protocol by guaranteeing that the components of the packet to be sent by the sender bear the correct relationship to the previous sender packets and that the key commitment of the new packet is for a fresh key. The precondition of the adversary action captures the restrictions on the adversary's power to use keys and key commitments. The effects of actions of the participants on the state are described below.

In addition to the three types of action performed by the participants, there is also the standard time-passage action `nu`, parameterized by the amount of time that has passed. This amount is constrained to be at most the amount of time remaining until the sender will send the next packet.

TESLA states in TAME. Abstractly, a state of TESLA embodies the current history of events: the packets sent, who sent them, when they were sent, and when (if ever) they were received. This information can all be represented as a set of annotated packets, each being a record consisting of a sender, a send time, a receive time (which may be ∞ for a packet never received), and a packet (or its contents). This set is represented in TAME by the state variable `SentPacket_part`. However, to simplify retrieval of information about the current state, some of this information is represented redundantly in separate state variables. A small set of special state invariants was proved to confirm that the redundant state variables contain the expected information with respect to the abstract state.

The sender and adversary actions both cause `SentPacket_part` to be updated by adding an appropriate annotated packet with receive time ∞. The sender action also updates the redundant state variables appropriately. The receiver action can "receive" any packet that appears as an annotated packet in `SentPacket_part` with receive time ∞. Receipt of `sp` causes a new packet $\widehat{\texttt{sp}}$ to be added to `SentPacket_part` that is identical to `sp` except that its receive time is set to the time of the receive action.

Representing TESLA timing properties in TAME. The basic TESLA scheme has only one special timing property: that the sender sends packets at regular intervals. This property is guaranteed in TAME by using the `first` and `last` components of the state, which map each action to the earliest and latest times of the time interval in which it must occur. The sender action begins with `first` and `last` set to 0, and each execution of the sender action advances both of these times by the fixed amount of time `I`. The adversary and receiver actions always have `first` and `last` set to 0 and ∞—that is, they are unrestricted as to when they may occur.

The timing condition $ArrT_i < T_{i+1}$ is captured in the formal definition of the authentication condition from Section 2.

5. Overview of the Correctness Proof

Correctness for (any version of) TESLA means that if the receiver can verify the authentication condition of TESLA for a packet, then the packet was indeed sent by the sender. The basic TESLA authentication condition in Section 2 is formalized in TAME as a predicate `Authenticated(sp:SentPacket,s:states)`, where `SentPacket` is the type of annotated packets. Because in basic TESLA the authenticity of the $(i+1)$-st packet depends on the authenticity of the i-th packet, the definition of `Authenticated` is recursive. Because `Authenticated` is formalized in terms of annotated packets, it uses `sp1`, `sp`, and `sp2` in place of `p1`, `p`, and `p2` from the informal definition in Section 2.

Given the predicate `Authenticated`, the correctness property for basic TESLA can be formulated as the following state invariant, `Inv_A(s:states)`:

```
(FORALL (sp:SentPacket):  Authenticated(sp,s) => Sender(sp) = S);
```

This formulation uses the notation `S` for the sender from the TAME model of TESLA (the adversary is `A`). The proof that `Inv_A` is an invariant is based on the following informal argument: If the sender of `sp` is `A`, then the key used to encode the `MAC` of `sp` must have been a sent key or one of `A`'s keys at the time `sp` was received. However, since `sp` was received before the sender revealed this key, the key was *not* a sent key. Moreover, the key is not one of `A`'s keys, because it was a fresh key committed to by `S`. Hence, the sender must have been `S`.

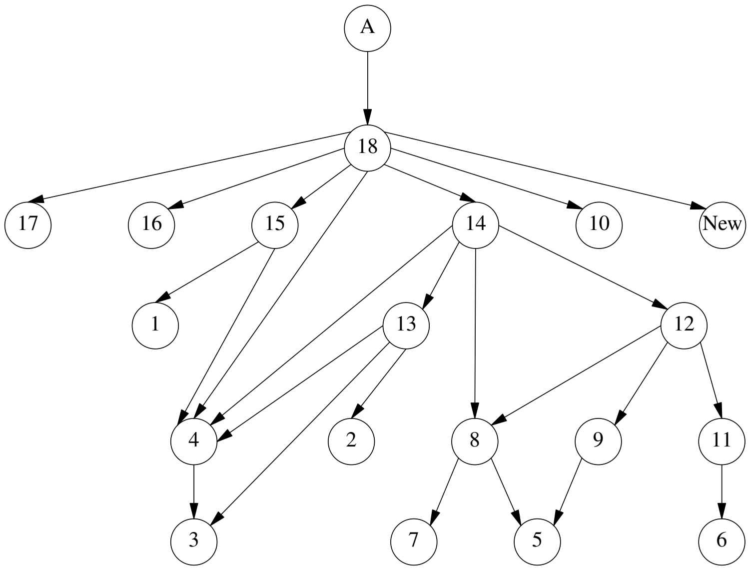

Figure 1. Dependencies among the TESLA invariants.

In all, nineteen auxiliary invariants were used in proving the invariance of `Inv_A`. The dependency relationships among these invariants are shown in Figure 1. The labels in the graph abbreviate the invariant names; thus, "A" stands for the invariant `Inv_A`. Because the definition of `Authenticated` is recursive on the index of the packet being authenticated, the proof first had to be cast as a proof by mathematical induction over this index. This was done by slightly reformulating the invariant `Inv_A` as `Inv_18`:

```
(FORALL (n:nat):  (FORALL (sp:SentPacket):
  Id(sp) = n & Authenticated(sp,s) => Sender(sp) = S));
```

`Inv_18` holds trivially when `n` is `0`. In the case `n > 0`, one may assume that the sender of the packet `sp1` in the definition of `Authenticated` is `S` in proving that the sender of the packet `sp` being authenticated is also `S`. `Inv_A` clearly follows immediately from `Inv_18`.

The informal argument that `Inv_18` is an invariant is the same as that for `Inv_A`. In formalizing this argument, one must first notice that the argument is implicitly reasoning about some state $\hat{\texttt{s}}$ in which `sp` has been received and in which the key `Key(sp2)` used to compute its `MAC` has been committed to by S but has not yet been revealed by S. But when `Authenticated(sp,s)` holds, a packet `sp2` revealing `Key(sp2)` has been received (and may have been sent by S). Hence, one cannot take $\hat{\texttt{s}}$ to be `s`. That $\hat{\texttt{s}}$ exists follows from `Inv_15`, which states that if the annotated packets `sp` and `sp1` have been sent in state `s`, the sender of `sp1` is `S`, and `sp` was received before the key `k` committed to in `sp1` was to be revealed by S, then there exists a reachable state ŝ whose current time `now(ŝ)` is before `k` is to be revealed, in which `sp` is has been received, and in which `sp1` has been sent by S. The reachability of ŝ is important to establish, since it allows other auxiliary state invariants to be applied to ŝ.

The three major invariants that support applying the informal argument outlined above to state ŝ are `Inv_10`, which says that a packet sent by `A` must use a sent key or a key not belonging to `S` to encode the `MAC`, `Inv_NEW`, which says that a key committed to in a packet sent by `S` must be one of `S`'s keys, and `Inv_14`, which implies that the key committed to in packet `sp1` was not a sent key at the time `sp` was received. For the precise formulation of all the invariants, see [1].

6. Mechanizing the Correctness Proof in TAME

Constructing the TAME proof for basic TESLA required a substantial amount of thought regarding both the model of TESLA and the set of auxiliary invariants sufficient to support the proof that `Inv_A` is an invariant. Although the description of basic TESLA in [13] did not explicitly mention including the packet index in a packet, this detail was added in the TAME model because it was hard to see how a receiver could reason effectively about authenticity without this additional information (given that the adversary can insert arbitrary numbers of intermediate packets). Looking ahead at other kinds of information one would probably need in the reasoning led to including some redundant variables in the state. For example, the state variable `SenderPacketList_part`, which accumulates the list of packets sent by S, was included to make retrieval of the most recently sent packets direct. The simplest method for retrieving the most recent packet with `SentPacket_part` alone would be to deduce the index of the latest packet from the current time and prove and use an invariant to the effect that for each index `n`, if the time is at least `n*I`, then there is a unique annotated packet in `SentPacket_part` whose index is `n`.

Two invariants were especially important to discover because they express details too obvious to include in the informal correctness argument. One of these invariants, `Inv_15`, was described in Section 5; it partially captures the "obvious" fact that any given state `s` has a predecessor state with an earlier time stamp in which events earlier than that time stamp have occurred. Another such invariant is `Inv_13`, which states that if packet `sp` was sent by `S` at time `n*I` and the current time is less than `(n+2)*I`, then `sp` is one of the two most recent packets sent by `S`. These invariants provided the needed "glue" to connect previously established invariants into the proof for `Inv_A`.

The proof of `Inv_13` is interesting in that it is the only one requiring some help from the user with respect to reasoning about nonlinear arithmetic: the user must directly apply the cancellation law to derive a needed equality of the form

TAME Strategy	Purpose
AUTO_INDUCT	Set up a structural induction proof
DIRECT_PROOF	Set up a non-induction proof
DIRECT_INDUCTION	Set up a mathematical induction proof
APPLY_SPECIFIC_PRECOND	Introduce the specified precondition
APPLY_GENERAL_PRECOND	Introduce the timing constraints
APPLY_IND_HYP	Apply the inductive hypothesis
APPLY_INV_LEMMA	Apply an invariant lemma
APPLY_LEMMA	Apply any general lemma
SUPPOSE	Do a case split and label the cases
COMPUTE_POSTSTATE	Compute the poststate of the current transition
SKOLEM_IN	Skolemize an imbedded quantified formula
INST_IN	Instantiate an imbedded quantified formula
TRY_SIMP	Try to complete the proof automatically

Figure 2. TAME strategies needed in the basic TESLA proof.

`n` = `i` from a hypothesis of the form `n*I` = `i*I`. Further study of the invariants and proof show that this need goes away if `Inv_13` is restated to replace the hypothesis that `sp` was sent by `S` at time `n*I` by the hypothesis that the index of `sp` is `n`. The only invariant whose proof relies on `Inv_13` is `Inv_14`, whose invariance proof can be slightly modified to use this new formulation of `Inv_13`.

The other places in which nonlinear real arithmetic complicates the complete TESLA proof are TCCs, both for the TESLA specification and in the proofs of some invariants. These TCCs arise when time is defined to be a nonnegative real number or ∞, and their proofs typically require the user to supply the information that the product of two nonnegative numbers—e.g., `n` and `I`—is nonnegative. Changing the definition of time to allow it to be an arbitrary real number or ∞ eliminated all of these TCCs. Thus, a combination of two changes in the specification of basic TESLA and its invariants allows complete avoidance of any special guidance from the user about nonlinear real arithmetic.

Figure 2 summarizes the TAME steps that were required in the proof. Most of these proof steps require appropriate arguments. Some of their effectiveness comes from the fact that they both maintain and use labels that correspond to the semantics of formulae in subgoals. For example, **INST_IN** and **SKOLEM_IN** can be directed to instantiate or skolemize with respect to an appropriate embedded quantifier in the inductive hypothesis. All of the proof steps in Figure 2 have been used in earlier TAME applications. Most have been used many times, but this is only the second application where **DIRECT_INDUCTION** has been needed: it is used to perform the combination of mathematical induction followed by direct reasoning about the automaton that is needed in the proof of `Inv_18`.

In addition to the TAME steps, the PVS step EXPAND (for expanding definitions) is also needed in the TESLA proof. When the specification and proof are not modified as described above to eliminate user guidance for reasoning about nonlinear arithmetic, a few additional direct PVS steps are required for this reasoning.

7. Conclusions

One expected benefit from the development of a correctness proof for an example protocol of the class of TESLA is a model that can provide guidance in the construction of correctness proofs for analogous protocols. The guidance that can be expected from the example proof described in this paper is in the form of the high-level features of the specification of the protocol and the nature of the set of auxiliary invariants that will be needed in the proof.

A very simple example of proof adaptation occurred during the development of the TESLA proof described in Section 5. The original formulation of basic TESLA in TAME allowed the adversary `A` less power: instead of having an initial set of known keys disjoint from the keys known to the sender `S`, `A` was assumed not to know any keys, and to have to use only keys revealed by the sender. The proof of correctness constructed for basic TESLA under these stronger restrictions on the adversary used almost the same structure of auxiliary invariants shown in Figure 1: only `Inv_New` was missing. `Inv_10` was stronger, saying that a packet sent by `A` had to use a sent key. Modifying the set of auxiliary invariants needed for the case when `A` has more power required only the weakening of `Inv_10`, the addition of `Inv_New` to compensate for this weakening, and minor modifications in the proof of `Inv_18`. Modifying the specification of basic TESLA in TAME required only adding a declaration of the set of keys known to `S` and the appropriate weakening of the precondition on the adversary action.

Of course, the degree of similarity of the proof of an analogous protocol to the proof of basic TESLA will depend on the degree of difference of this protocol from basic TESLA. However, one can expect it to remain appropriate to model the state of the protocol using a set of annotated packets and some form of list of the packets sent by the sender, to capture the details of the protocol in the precondition of a sender action, and to capture assumed restrictions on the power of the adversary in the precondition of an adversary action. One can also predict that certain invariants or their analogues will be needed for particular roles in the proof. For example, the "informal correctness argument" will be similar, and will be formalized, as in the basic TESLA proof, using analogues of `Inv_10` and `Inv_14` in the manner described in Section 5. There will almost certainly have to be an analogue of `Inv_15` establishing the existence of a predecessor state to which the analogues of `Inv_14` and `Inv_10` can be applied.

For other variants of TESLA described in [13], one can make more precise predictions. For example, the simplest of these variants differs from basic TESLA only in using each i-th key as the commitment to the $(i+1)$-st key. The timing properties are unchanged, and thus `Inv_4`, which says that the sender sends the

n-th packet at time `n*I` will remain the same. A more complex variant ties packets and the keys to their `MACs` to *time intervals* rather than to packet indices. For this variant, some more sophisticated analogue of `Inv_4` referring to the time at which the n-th key (rather than the n-th packet) is sent is likely to be needed.

The above preditions will be tested in future work to adapt the basic TESLA proof to these more sophisticated variants.

Acknowledgments

I thank Catherine Meadows for introducing me to this problem. I also thank Catherine, E. Leonard, R. Bharadwaj, and C. Heitmeyer for helpful comments.

References

[1] M. Archer. Proving correctness of the TESLA multicast stream authentication protocol with TAME. Draft report.

[2] M. Archer. TAME: Using PVS strategies for special-purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, 29(1-4), 2000. Published February, 2001.

[3] M. Archer, C. Heitmeyer, and E. Riccobene. Using TAME to prove invariants of automata models: Case studies. In *Proc. 2000 ACM SIGSOFT Workshop on Formal Methods in Software Practice (FMSP'00)*, August 2000.

[4] S. H. Brackin. Using checkable types in automatic protocol analysis. In *Proc. 15th Annual Computer Security Applications Conference (ACSAC '99)*, pages 99–108. IEEE Comp. Soc. Press, December 1999.

[5] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems.* To appear.

[6] C. Heitmeyer and N. Lynch. The Generalized Railroad Crossing: A case study in formal verification of real-time systems. In *Proc., Real-Time Systems Symp.*, San Juan, Puerto Rico, Dec. 1994.

[7] J. Hickey, N. Lynch, and R. V. Renesse. Specifications and proofs for ensemble layers. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, volume 1579 of *Lect. Notes in Comp. Sci.*, pages 119–133. Springer-Verlag, December 1999.

[8] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, volume 1055 of *Lect. Notes in Comp. Sci.*, pages 147–166. Springer-Verlag, December 1996.

[9] N. Lynch and F. Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[10] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[11] J. Millen and H. Rueß. Protocol-independent secrecy. In *2000 IEEE Security and Privacy Symposium*, pages 110–119, Oakland, CA, May 2000. IEEE Computer Society.

[12] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[13] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of IEEE Security and Privacy Symposium (S&P2000)*, pages 56–73, May 2000.

[14] N. Shankar, S. Owre, J. M. Rushby, and D. W. J. Stringer-Calvert. The PVS prover guide. Technical report, Computer Science Lab., SRI Intl., Menlo Park, CA, 1998.