Low Cost Multicast Authentication via Validity Voting in Time-Triggered Embedded Control Networks

Chris Szilagyi
ECE Department
Carnegie Mellon University
szilagyi@cmu.edu

Philip Koopman
ECE Department
Carnegie Mellon University
koopman@cmu.edu

Abstract

Wired embedded networks must include multicast authentication to prevent masquerade attacks within the network. However, unique constraints for these networks make most existing multicast authentication techniques impractical. Our previous work provides multicast authentication for time-triggered applications on embedded networks by validating truncated message authentication codes across multiple packets. In this work, we improve overall bandwidth efficiency and reduce authentication latency by using unanimous voting on message value and validity amongst a group of nodes. This technique decreases the probability of successful per-packet forgery by using one extra bit per additional voter, regardless of the number of total receivers. This can permit using fewer authentication bits per receiver. We derive an upper bound on the probability of successful forgery and experimentally verify it using simulated attacks. For example, we show that with two authentication bits per receiver, adding four additional bits per message to vote amongst four nodes reduces the probability of per-packet forgery by a factor of more than 100. When integrated with our prior work on time-triggered authentication, this technique reduces the number of authentication message rounds required for this example by a factor of three. Model-checking with AVISPA confirms data integrity and data origin authenticity for this approach.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and Protection

General Terms

Design, Reliability, Security.

Keywords

Embedded, Networks, Security, Voting, Multicast, Authentication, Controller Area Network, CAN, FlexRay, Time-Triggered Protocol, TTP, In-vehicle, Real-time.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WESS'10, October 24, 2010, Scottsdale, AZ,USA
Copyright 2010 ACM 978-1-4503-0078-0 ...$10.00.

1. Introduction

Embedded control networks are increasingly connected to external media and networks [16] (e.g., wireless and Internet), but do not support secure authentication to prevent manipulation of internal message traffic. In the event that an attacker accesses the internal embedded control network, whether through physical manipulation or via a compromised network connection, they can trivially inject messages to disrupt system operation and subsequently violate safety requirements. Koscher et. al. demonstrated that the brake controller in a modern automobile can be remotely manipulated using a wireless-enabled MP3 player connected to the embedded control network [18]. An attacker might access the embedded control network through such a connection to engage an emergency brake in a car while it is traveling on a highway, unlock doors and start the engine, or shut off headlights while traveling at night.

Common embedded network protocols, such as Controller Area Network (CAN) [2], FlexRAY [8], and Time-triggered Protocol (TTP) [24], do not support secure cryptographic mechanisms to prove that a message originated from the node that claims to have sent it. Thus, these protocols are vulnerable to masquerade and replay attacks. Masquerade attacks occur when a node sends a message in which it claims to be a node other than itself. This attack can be performed by broadcasting during another node's Time-Division Multiple Access (TDMA) slot or by changing a message identifier. Similarly, replay attacks occur when a previously sent message is recorded and retransmitted by an attacker.

Values are typically multicast over a broadcast bus, requiring simultaneous authentication of messages to multiple receivers. While many methods for multicast authentication exist, embedded network constraints make them impractical. Resource-limited nodes must authenticate short periodic messages within tight real-time deadlines and tolerate potentially high packet losses. A reasonable size for an authenticator in an eight byte packet might be two to four bytes (similar in size to current error detection codes). However, existing multicast authentication schemes require tens to hundreds of bytes per packet.

This paper builds on our previous work for authentication in statically scheduled time-triggered applications. Our previous work amortizes authentication bandwidth costs over multiple time-triggered packets, using truncated Message Authentication Codes (MACs). This approach allowed us to provide secure authentication for time-triggered applications. However, using one MAC tag per receiver introduces unused redundancy because each receiver only benefits from a single MAC tag. If limited to a few bytes per-packet to authenticate to a large number of receivers, a sender may have to amortize authentication over too many packets to meet real-time deadlines.

In this work, we integrate voting to improve the bandwidth efficiency and subsequently reduce the latency of our approach for time-triggered authentication. We take advantage of the redundant MACs in our previous approach to allow a group of nodes to cross-check the validity of a packet. Using the redundant MACs as secondary confirmation channels to validate a packet requires an attacker to forge multiple authenticators rather than just one per packet. We provide a conservative upper bound on the probability of successful per-packet forgery and verify this bound experimentally using simulated attacks. Using this upper bound, we also demonstrate that this combined approach can reduce application level latency to just a few message rounds using a few bits per receiver. We discuss tradeoffs among probability of per-packet forgery success, packet loss, and tolerance to node compromise. Additionally, we use the model-checker AVISPA [1] to verify that this approach provides data integrity and data origin authenticity to prevent masquerade and replay attacks.

In this paper, Section 2 describes our assumptions for embedded control networks. Section 3 surveys related work. Section 4 presents our attacker model. Section 5 reviews our baseline multicast authentication and time-triggered authentication approaches. Section 6 introduces our voting technique. Section 7 describes our verification of the security of this approach using AVISPA. Section 8 provides an upper bound on the probability of successful forgery and experimental verification of this bound. Section 9 discusses some limitations of our approach. Finally, section 10 states our conclusions and future work.

2. Embedded network assumptions

We focus on embedded control networks composed of a set of hardware Electronic Control Units (ECUs) connected to a single-hop broadcast bus. These ECUs communicate via a network using a protocol such as CAN, FlexRay, or TTP to accomplish time-triggered communications. This work assumes an embedded control network has the following characteristics:

Time-triggered - We consider only time-triggered applications where communications are defined by a *static TDMA schedule*. A real-time system is time-triggered if all communications and processing activities are initiated at predetermined points in time from an a priori designated clock tick [17]. Each node periodically broadcasts current values for a set of predefined message types during designated time slots. While protocols like CAN are not TDMA, our techniques can be applied to statically scheduled variations such as Time-Triggered CAN (TTCAN) [10].

Multicast communications - Most distributed embedded networks are inherently multicast. All nodes connected to the network can receive every packet. Each packet includes the sender's identity, often implicitly through a message identifier (CAN; FlexRay) or time slot (TTP), but usually no explicit destination information.

Static network configuration - We assume the network configuration is fixed at design time, with no run-time reconfiguration. Usually only a few nodes are attached to any network (commonly 32 or fewer), as opposed to enterprise networks which may have hundreds or thousands of potential receivers.

Resource limited nodes - Processing and storage capabilities of nodes are often limited due to cost considerations. Authentication mechanisms which require large amounts of processing power or storage in RAM may not be feasible. More powerful ECUs are impractical for most nodes in the system, and many nodes are 8-bit ECUs with significantly smaller memories due to cost and power considerations.

Small packet sizes - Packet sizes are small in embedded network protocols when compared to those in enterprise networks. Packets have maximum data payload sizes as small as eight bytes in the case of CAN, while payloads for FlexRay and TTP can be 254 bytes and 236 bytes respectively. Authentication should incur minimal bandwidth overhead. Our goal is to produce authenticators similar in size to current error detection codes (two to four bytes in size).

Tolerance to packet loss - Embedded networks are subject to message blackouts from environmental disturbances such as interference from large electric motors. As such, authentication schemes must tolerate packet losses as part of normal system operation.

Real-time deadlines - In real-time systems, processes must complete within specified deadlines. Authentication of nodes must occur within a known time bound, with that bound being fast enough to match the physical time constants of the system being controlled (as fast as tens of milliseconds).

3. Related work

Several authors have shown the vulnerability of embedded networks protocols to masquerade attacks. Wolf et al. [25] provide an overview of the security vulnerabilities of in-vehicle network protocols, and identify the need for multicast authentication to prevent masquerade attacks. Koscher et. al. [18] demonstrate the ease with which spoofed messages allow an attacker to control safety critical actuators in a live automobile. This section surveys related work in multicast authentication, embedded network security, and fault tolerance methods similar to our voting approach.

3.1 Existing multicast authentication

The multicast nature of embedded network protocols makes authentication particularly challenging. A single point-to-point cryptographic authenticator does not provide adequate security in a multicast setting. Multicast authentication requires some form of key asymmetry, so that no node can masquerade as a different node.

Many approaches for multicast authentication exist, however most cannot be applied with our target bandwidth overhead of two to four bytes per packet and resource constrained nodes. Most require tens or hundreds of bytes per packet or rely on computationally expensive digital signatures.

Digital signatures provide strong source authentication using public and private keys, but the processing overhead makes it impractical for a resource constrained device to compute digital signatures for each message for real-time control. For example, pagers and Palm Pilots can take several seconds to compute a 512 bit RSA signature in resource constrained nodes [3]. Amortizing a digital signature (e.g., Wong and Lam [26]) reduces per-packet bandwidth overhead, but waiting to authenticate tens or hundreds of time-triggered samples at once would make these approaches unsuitable for real-time operations.

One-time digital signature schemes [12] allow senders to sign messages faster than traditional digital signatures, but can incur several kilobytes of authentication data per message. Amortizing these costs makes them impractical for the same reasons as traditional digital signatures.

Canetti et al. [4] suggest a multi-MAC scheme which appends k one-bit MACs to each message, computed using k different keys. The keys are distributed amongst receivers such that at least w receivers must collude to forge a message. However, mitigating collusion can require hundreds or thousands of authentication bits per message. Our work also uses truncated MACs and assumes a fixed number of compromised nodes, but still fits within two to four bytes of per-packet bandwidth overhead.

TESLA [22] uses time-delayed release of keys to provide asymmetry. By releasing keys at a pre-specified interval after a MAC is released, receivers can confirm the authenticity of the data from a sender. The released keys are computed using one-way hash chains. Storing the entire chain of keys is prohibitive, so techniques are used to reduce memory overhead at the expense of a small recomputation cost. While TESLA sends a single MAC per interval, it also requires the sender to include a key for each interval of messages to be authenticated. The authenticator could be truncated using our approach for time-triggered authentication [23], but the key cannot. Truncating the key exponentially reduces the security of this approach. Hu et. al. propose a variation on one-way hash chains, called sandwich chains, which allows smaller keys to be released per message by regularly initializing new key chains [13]. However, this technique assumes the attacker does not have the computational resources to break the current key before the next is released. This technique does not seem feasible with a target per-packet bandwidth overhead of two to four bytes.

Chan and Perrig propose a multi-MAC approach using secure aggregation [5] over multiple rounds. Each node initally authenticates their value to a base station node. The base station then releases a combined authenticator for the group. Finally, each node releases its portion of the combined authenticator. Once all nodes have transmitted, all nodes can "unlock" the authenticator. However, this approach assumes reliable point-to-point communications, which may not be available in most embedded networks. This approach also requires a base station, which might act as a single point of failure.

3.2 Embedded network security

This work builds upon our proposed technique for time-triggered multicast authentication [23]. Our technique takes advantage of the existing temporal redundancy of message values and system inertia in embedded control networks to authenticate messages over multiple packets. We improve our time-triggered authentication technique in this work by integrating voting techniques.

Other approaches such as SPINS [21] and TinySec [14] apply security to resource constrained wireless sensor networks. However, those approaches are specifically designed for use in wireless networks, which do not typically have real-time deadlines.

Morris and Koopman [19] identify the potential for masquerade failures to cause accidental or malicious failures, via non-critical nodes masquerading as higher criticality nodes. They propose counter-measures of varying strengths to prevent masquerading failures between nodes of varying criticality. Their approach assumes non-malicious software faults or attacks from a cryptologically unsophisticated attacker. Fault tolerance mechanisms are not necessarily secure against malicious masquerade or replay attacks. Masquerade prevention for safety-based systems typically uses bus guardians or a symmetric key shared among all trusted nodes. Compromise of a single node would permit an attacker to masquerade as any system node.

3.3 Fault tolerance

Voting techniques and redundancy are a classic approach to improve system reliability [20]. These techniques enable fault detection and handling to prevent fault propagation in a system. Typically system designers assume each input to a voter or comparator mechanism fails randomly and independently of others. In our approach, nodes detect differing views of message authenticity by voting on the validity of MAC tags from other nodes. We assume the outputs of each MAC function can only be successfully forged randomly and independently of other MAC functions.

Our voting approach also has similarities to the TTP group membership service [24]. This service provides agreement on current operating mode and set of nodes believed to be correct and alive. In TTP, nodes encode membership information into packet error detection codes. Disagreeing error codes indicate either the sender or receiver failed, and nodes take appropriate action to segregate out the failed node. We use a similar technique, computing a MAC function over a previous set of values seen from the network and a bit vector indicating each value's validity. In our approach, disagreeing authenticators indicate that an attacker may have fooled one or more receivers. Nodes then reject potential forgeries.

4. Attacker model

This work uses an active attacker model that controls the network. An attacker may modify, inject, drop, or eavesdrop upon network traffic. However, an attacker cannot successfully forge packets unless they have access to the appropriate key or can randomly guess the authenticator correctly.

An attacker must not be able to masquerade as any critical node they do not already control to induce a system failure, except with some acceptably low probability. We do not address how an attacker gains access to a network, but rather how to prevent masquerade and replay attacks from succeeding in the event that they do gain access (e.g., through a compromised node or physical access). Attackers accessing the network through compromised nodes will have access to the key material in those nodes and can send messages from those nodes.

This work assumes an attacker is aware of existing error detection mechanisms along with the network schedule, and is capable of injecting well-formed packets in valid time slots. We constrain the attacker to one forgery attempt per valid time slot in a TDMA network such as TTP or FlexRay, since transmitters are only permitted to transmit a single packet per time slot in a time-triggered application.

5. Background

This section describes our baseline approach for multicast authentication and summarizes our previous work on time-triggered authentication [23].

5.1 Baseline - one MAC per receiver

Our techniques use one MAC per receiver as a baseline multicast authentication mechanism. When transmitting a packet, the sender computes one MAC per receiver and appends the outputs to the data payload. We assume each pair of communicating nodes securely establishes symmetric keys during node installation or replacement (e.g., Diffie-Helman key exchange [6]). This prevents masquerade attacks because no more than two nodes share the same key. Further, to prevent replay attacks and limit the rate at which attackers may attempt forgeries we assume nodes have securely synchronized to a common time base (current time or TDMA round number) at system start up (e.g., Secure Pairwise Synchronization [11]).

Bandwidth constraints do not permit the full output of a MAC function for each receiver in a packet (potentially hundreds of authentication bits per packet). Thus, we modify this baseline mechanism to amortize the authentication bandwidth costs over multiple time-triggered packets.

5.2 Time-triggered authentication overview

Our previous work on time-triggered authentication uses the *temporal redundancy* present in most time-triggered system designs to amortize authentication bandwidth overhead across multiple time-triggered packets [23]. In time-triggered applications, nodes periodically broadcast current values of state variables and sensor inputs to the rest of the network. Receivers then update outputs and actuators based on the most current system state. This information is typically sampled faster than the time constraints of control stability requirements. As a rule of thumb, ten or more samples are sent within the rise time of a control system or prior to a system deadline [9][17]. Choosing such a sample rate reduces the delay between a command and the system response, smoothes outputs to steps in control input, and tolerates lost messages.

5.2.1 Time-triggered message generation

When transmitting, the sender generates one MAC tag for each distinct receiver of the packet. The sender computes each MAC over the packet header, message value, shared secret key, and synchronized time. The sender truncates each tag to just a few bits and appends the tags to the message value. We assume there are fewer receivers for any particular message than available bits in the data payload, allowing one truncated MAC tag per receiver to be placed into each packet.

5.2.2 Time-triggered message verification

To reduce the rate at which masquerade attacks induce system failures to occur no more often than acceptable failure rates, nodes verify a message over multiple time-triggered packets.

For state-changing messages, which cause state machine transitions or discrete actuations (e.g., locking car doors if vehicle speed exceeds some threshold), nodes wait until a sufficient fraction of the most recently received packets for that message type are valid and have consistent values. The receiver then commits to the transition triggered by those values.

For reactive control message types, which update continuous or ordered values in nodes running feedback control loops (e.g., updating vehicle speed in automatic cruise control), we rely on system inertia to damp the response to individual packets. The damped response to messages requires an adversary to successfully forge multiple packets within some period of time to compromise system operation.

6. Voting on message authenticity

In this section, we introduce a new technique in which nodes cross-check message values to increase bandwidth efficiency when using one MAC per receiver. This reduces the probability of successful per-packet forgery while increasing bandwidth consumption by one bit per voting node. A group of nodes exchanges indications of the received value and validity of each packet. During each time slot, nodes update the validity of their most recently received messages, rejecting any value the group disagrees upon or indicates as invalid. Any disagreement indicates a masquerade attempt, whereas unanimous agreement indicates no such attempt.

6.1 Enabling properties

We use secure MAC functions to enable voting on message validity and detect disagreement on message values. Without knowledge of the key, an attacker can at best guess the MAC tags for any message value it injects or modifies. Further, because nodes compute each tag with different keys, successfully forging one MAC tag does not assist the attacker in forging another tag.

In our approach, a group of nodes attests to the validity of each other's message values. A sender transmits its value and directly authenticates it to each group member. In subsequent time slots, each member attests to the validity of a previous sender's value by computing their MACs over that sender's value in addition to its own transmitted value. Group members accept a previous sender's value if the sender's packet contained a valid MAC tag, all packets attesting to that value also had valid MAC tags, and all attesting packets indicated the previous sender's value was valid.

This attestation process creates a series of indirect secondary confirmation channels from the sender to each receiver, and from each receiver to all other receivers. Because an attacker can only forge each authenticator randomly and independently of each other, each receiver in the group can vote on the results of these channels to reduce the probability of successful forgery.

We also take advantage of the collision resistance of secure MAC functions so that nodes do not have to explicitly retransmit values being compared using these secondary indirect channels. By computing the MAC function over the current value, previous values, and their validity, the MAC tags should only be valid if the sender and receiver agree on the values and validity of all packets.

6.2 State variables and functions

To check for discrepancies in packet value or validity, each node n maintains three state vectors: a *value vector* R_n, *validity vector* V_n, and *confirmation vector* C_n. We use a subscript to denote the identity of the node that produces a variable (e.g., R_n is the value vector produced by node n). Nodes initialize all vectors to zeros.

The value vector R_n stores the most recently received value (valid or not) for each message type defined in the TDMA schedule that node n consumes or participates in voting on. Receivers record lost packets as a predefined error code 'lost' if they detect a transmission error (indicated by an incorrect error checking code or no packet broadcast in a scheduled time-slot).

The validity vector V_n contains the authentication results of each entry in the value vector. A node stores a '1' value if the most recent value for the corresponding message type was valid and a '0' value if invalid.

Finally, the confirmation vector C_n contains a counter of positive secondary confirmations of validity for each message type in R_n.

We define a function *getMostRecent*(z, R_n, V_n, C_n) to obtain a subset of received values, their validity, and confirmations. This function produces a triple <r_n, v_n, c_n> of vectors; where r_n is a chronologically ordered subset of R_n containing z values recently received by node n, v_n is a subset of V_n containing the validity bits for each element in r_n, and c_n is a subset of C_n containing confirmation counters for each element in r_n. We require that z be sufficiently small, such that r_n does not contain more than one sample of any message type broadcast by any node (e.g., if node X broadcasts most frequently, sending any of its message types at most once every eight time slots, then z can be no greater than seven). This allows each time-triggered sample of a message type to be authenticated independently of another sample of the same message type. Two nodes executing *getMostRecent* during the same time slot will obtain the same ordering of message types, because they share the same TDMA schedule.

The function *setNewest*(*msg*, *validity*, R_n, V_n, C_n) replaces the element of R_n for the message type broadcast in the current time slot with value *msg*. The corresponding element in V_n is set to '1' if *validity* is 'valid', and '0' if *validity* is 'invalid'. The corresponding element in C_n is set to zero.

The functions *updateValidity*(z, v_n, V_n) and *updateConfirmations*(z, c_n, C_n) overwrite the z elements in V_n or C_n with the elements of v_n or c_n respectively, using the inverse of the mapping used in function *getMostRecent*.

6.3 Message generation for voting

We modify the sending process for time-triggered packets (Section 5.2.1) to allow senders to attest to the validity of z recently received message values in addition to authenticating the current message value (Figure 1).

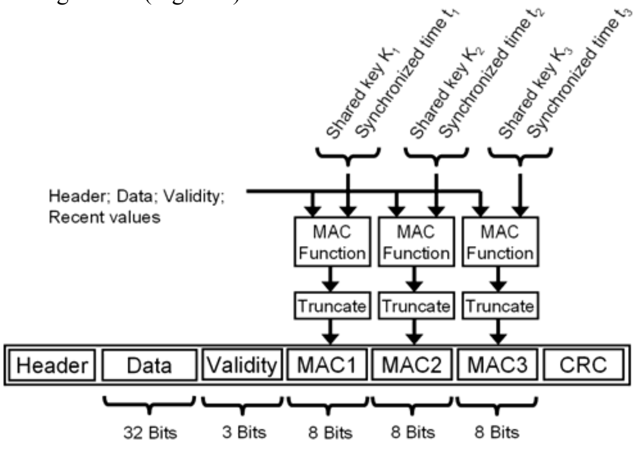

Figure 1 - Example of message generation process for 32 bits of data and three 8-bit MACs, using unique shared keys and synchronized times for three receivers. This packet includes three validity bits, attesting to three prior message values.

For each receiver i, a sender S computes the MAC function over the current header and message value, shared secret key k_i, synchronized time t, and vectors r_S and v_S produced by *getMostRecent*(z, R_S, V_S, C_S). Before computing the MAC functions, the sender replaces any element of r_S with an 'invalid' value if the validity vector v_S indicates the that value's packet contained an invalid authenticator. We use MMAC as a short hand notation for a function that computes an array of MAC tags (one per receiver) and truncates each MAC tag to just a few bits.

The sender includes the array of truncated MAC tags in the data payload as before, but also includes the validity vector v_S. This allows receivers to recompute the MAC function over the same values as the sender, replacing values with 'invalid' for those indicated by v_S. After broadcasting their packet, the sender optimistically sets its own validity vector assuming its packet is received correctly with a valid authenticator. Figure 2 provides pseudocode for the send process.

6.4 Message verification and voting

Receivers use each authenticator to confirm that the current packet and the most recently seen packets are valid. We break down the message verification into two processes. During each time slot, each receiver executes the Receive process, followed by the Final Verification process (Figure 2).

6.4.1 *Receive process*

If a transmission error occurs, the receiver i records a 'lost' value for the received message type, marks it as valid, and exits the receive process without incrementing any confirmation counters. Otherwise, the receiver executes *getMostRecent* to obtain the most recent set of message values r_i received from the network, corresponding validity vector v_i, and confirmation vector c_i. The receiver replaces any element of r_i with an 'invalid' value if the sender's transmitted validity vector v_S indicates the sender believes that value's packet contained an invalid authenticator. The receiver recomputes the MAC function, and compares the MAC tags.

The MAC tags will only be equal if the sender and receiver agree on the current and prior values (with the infrequent exception of MAC collisions). If they match, the receiver accepts the current value as valid. If the tags do not match, the receiver rejects the current value and all prior values that the sender is attesting to. Because the attested values are sent implicitly as inputs to the MAC function, the receiver cannot determine which value caused the disagreement and conservatively rejects all attested values. For a valid packet, receivers execute a vote on the authenticity of attested values. Receivers reject an attested value as invalid if either the sender's valid packet indicated it was invalid or the receiver originally saw that value as invalid. To perform the vote, we perform a bitwise logical And operation on the v_i and v_S vectors. For any value in r_i that is still considered valid in v_i after the vote, the receiver increments the corresponding counter in the confirmation vector c_i.

Once this process is complete, the results are committed to the complete vectors R_i, V_i, and C_i.

6.4.2 *Final Verification process*

Once the Receive process is completed during a time slot, the receiver checks any packets for which all secondary confirmations should have been received. There are three possible outcomes for a value: invalid, lost, and valid. First, if the bit in the validity vector V_i is '0', then the receiver rejects the value as invalid, because at least one voting node claimed that the packet was a

```
Send process, performed by node $S$:
• Ready to send message value $m_S$ to all nodes
• $<r_S, v_S, c_S> \leftarrow$ getMostRecent(z, $R_S$, $V_S$, $C_S$)
• For any element of $v_S$ that is '0', replace the corresponding ele-
  ment of $r_S$ with 'invalid'
• tag_array$_S \leftarrow$ MMAC($m_S$ | $t$ | $r_S$ | $v_S$)
• Broadcast {$m_S$ | $v_S$ | tag_array$_S$}
• setNewest($m_S$, 'valid', $R_S$, $V_S$, $C_S$)

Receive process, performed by node $i$:
• Receive {$m_S$ | $v_S$ | tag_array$_S$}
• If transmission error occurs
      • setNewest('lost', 'valid', $R_i$, $Vi$, $Ci$)
      • Return from receive process
• $<r_i, v_i, c_i> \leftarrow$ getMostRecent(z, $R_i$, $V_i$, $C_i$)
• For any element of sender's $v_S$ that is '0', replace the corre-
  sponding element of receiver's $r_i$ with 'invalid'
• tag$_i \leftarrow$ MAC$_{ki}$(m$_S$ | $t$ | $r_i$ | $v_S$)
• If (tag$_i$ = tag_array$_S$[i])
    Accept new value as valid
      • setNewest($m_S$, 'valid', $R_i$, $V_i$, $C_i$)
      • $v_i \leftarrow$ bitwiseAnd($v_i$, $v_S$)
      • updateValidity(z, $v_i$, $V_i$)
      • For each element in $v_i$ that is '1', increment $c_i$ counters
      • updateConfirmations($z$, $c_i$, $C_i$)
• Else,
  Reject previous values the current MAC tag included
      • setNewest($m_S$, 'invalid', $R_i$, $V_i$, $C_i$)
      • Set all elements in $v_i$ to '0'
      • updateValidity(z, $v_i$, $V_i$)

Final Verification process, performed by receiver $i$:
After Receive process is completed, perform final verification
step for each message type that node $i$ has received all $z$ secon-
dary confirmations:
• Reject value as masquerade attempt if bit in $V_i$ is '0'
• Accept value as lost if bit in $V_i$ is '1' and (value from $R_i$ is "lost"
  or confirmations in $C_i < z$-1)
• Accept value (valid and not lost) if the corresponding bit from
  $V_i$ is '1' and number of confirmations in $C_i$ equals $z$-1.
```

Figure 2 - Pseudo-code for message generation and verification processes using voting, during time slot *t*.

masquerade attempt. Second, if the bit in V_i is '1', and the value is 'lost', then the receiver accepts that the packet suffered a transmission error and no other receivers claimed it to be a masquerade attempt. Similarly, receivers accept a value as lost if it is valid, but an insufficient number of positive confirmations were received. Finally, if V_i indicates the value is valid, the value is not 'lost', and the counter in the confirmation vector C_i indicates a sufficient number of positive confirmations from other voting nodes, then the value is accepted as valid.

For a received packet to be accepted as valid, there must be a unanimous vote among the z voting nodes that the packet contained a valid authenticator. To fool a single receiver into accepting an injected value, an attacker must successfully forge not only the MAC tag for that receiver, but must also successfully forge the z-1 other tags to or from the rest of the voting nodes.

We emphasize that successfully forging one or two packets, then provoking receivers to drop the attestation packets does not increase an attacker's chance of forging a message. By dropping any attesting packets, the packets targeted for forgery will also be dropped by receivers.

6.5 Integrating time-triggered authentication

To amortize the bandwidth cost of authentication, we integrate our voting technique with our prior work on time-triggered authentication. Time-triggered authentication validates a message or actuation command over a set of independent samples.

To accomplish this, our voting technique must validate each time-triggered sample of a message type independently of other samples of the same message type. In our voting process, each packet can only attest to prior packets, preventing interference with future packets of the same type. To prevent interference with prior packets of the same type we limit the number of confirmations such that a packet does not attest to more than one sample of any message type broadcast from a single node, nor does the current packet attest to any previous message broadcast from the same sender. Thus, by the time the current value of a message type is broadcast, all nodes have completed the final verification process for the previous value of that message type. For example, in Figure 3, three nodes each broadcast message types m_1, m_2, and m_3 respectively. Packets of message types m_2 and m_3 attest to those of type m_1, but by the time the current sample of type m_1 is broadcast, the two confirmations of the previous sample of m_1 have already been broadcast and processed.

This independent verification of each sample also enables quick recovery from transient faults or masquerade attacks. As soon as the source of transmission interference or attack ceases, receivers simply resume authenticating over new values. Old corrupted values cannot interfere with authentication of future values. However, a single loss will affect a few previous packets.

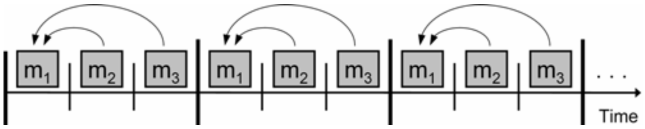

Figure 3 - Example TDMA schedule with non-overlapping attestations. Receivers complete verification of m_1 values using m_2 and m_3 by the time the next value of type m_1 is sent.

6.6 Potential complications and tradeoffs

6.6.1 Packet loss

This approach introduces a design tradeoff between loss tolerance and probability of successful packet forgery. By requiring more secondary confirmations, we reduce the probability that an attacker successfully forges individual packets. However, this also increases the number of packets lost by a single transmission error. If a packet is lost by all nodes due to a symmetric fault, the number of positive confirmations for the values attested to by the lost packet will not be high enough for those values to be accepted. Nodes will drop all packets attested to by the lost packet.

One minor issue with our approach is that an asymmetric packet loss (some receivers see a well-formed packet, while others drop the packet) will be interpreted as invalid. MAC tags will disagree because two nodes observed and recorded a different set of values. To resolve this, an additional bit vector (similar to the validity vector) can be transmitted to allow voting nodes to indicate which packets were lost. While this modification is beyond the scope of this paper, we plan to incorporate this in future work.

Lastly, while our approach recovers once transient faults cease, permanent node failure might cause the same set of packets to be repeatedly lost. We also plan to address this in future work.

6.6.2 Tolerating compromised nodes

Relying on secondary confirmations from other nodes introduces a tradeoff between tolerance to compromised nodes and probability of successful per-packet forgery. Compromised nodes could assist in forgery attempts, attesting that a forged packet from an attacker is valid. The probability that this secondary confirmation is successfully forged is equal to one. To tolerate a fixed number of compromised nodes *w*, a node must receive a total of *z*+*w*-1 total positive confirmations before finally accepting a value. System designers may trade tolerance to node compromise for increased probability of successful forgery. We assume the number of compromised nodes is limited to one or two nodes. If an attacker controls multiple critical nodes participating in voting, then the attacker can likely cause the system to fail without resorting to masquerade attacks.

7. Model-checking

To confirm that this voting technique for authentication is secure, we implemented and model-checked this technique using the Automated Validation of Internet Security Protocols and Applications (AVISPA) framework [1]. Model-checking is a formal method based technique for verifying properties of concurrent finite-state systems. Model-checking security protocols allows designers to identify flaws which allow an attacker to circumvent the protocol. Our goal is to use model-checking to ensure an attacker cannot successfully forge a packet despite full control over the network, and control over some nodes. This requires verification that our protocol provides data origin authenticity and data integrity. In AVISPA, when testing for data origin authenticity, data integrity is implicitly verified as well.

AVISPA uses a Dolev Yao attacker model [7], giving the attacker full control over the network. This is similar to our attacker model in Section 4. However, the Dolev Yao model assumes that all cryptographic primitives are unforgeable unless the attacker obtains the correct key material. We address the probability the attacker successfully guesses authenticators in Section 8.

7.1 Model description

We implement a simple network (Figure 4) consisting of three nodes N_1, N_2, and N_3, broadcasting message types m_1, m_2, and m_3 respectively. Each node is modeled as an independent process, broadcasting and receiving according to a fixed schedule. We model the broadcast bus using point-to-point channels, sending a copy of every message simultaneously on each channel. However, all messages in AVISPA are passed through the attacker [1] regardless of channel definitions, resulting in a bus-like topology.

Nodes communicate according to a round-robin TDMA schedule, in which each node takes a turn broadcasting, then the cycle repeats (as per Figure 3). We split the model of each node into five time slots, allowing each node to complete our protocol on one value of each message type (Figure 5). In each slot, one node sends while the other two receive and vote. In this model, nodes transmit the current value of their message type, and attest to the validity of the most recent value of the other two (as per Section 6) Nodes compute MAC functions over the current value of their message type, the two previous values transmitted by the other nodes, and the validity of those two other message types. Each node receives a direct authenticator and one indirect secondary confirmation of validity for each message type.

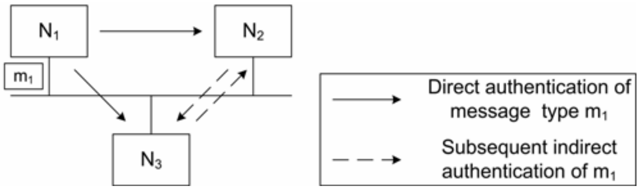
N_1
N_2
m_1
N_3
Direct authentication of message type m_1
Subsequent indirect authentication of m_1

Figure 4. Model of three nodes authenticating message type m_1. Node N_1 directly authenticates m_1 to N_2 and N_3. In subsequent time slots, N_2 and N_3 exchange indirect confirmations of m_1's validity and vote on the results.

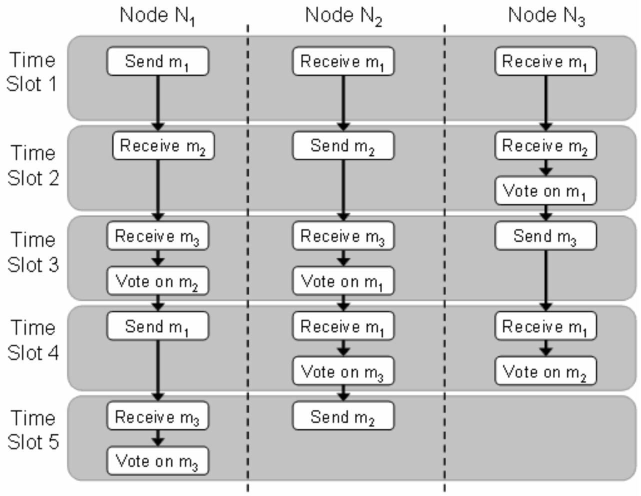

Figure 5. Our model executes over five time slots, allowing each node to cross-check each of three message types.

Assume valid m_2 and m_3 values have been previously transmitted without attacker interference (for simplicity, nodes in our model do not vote on these previous values). During time slot one, N_1 sends m_1 with authenticators for N_2 and N_3, attesting to the validity of prior values of m_2 and m_3. Nodes N_2 and N_3 receive m_1 and check its authenticity. If m_1 is valid, N_2 updates its value and validity vectors for m_1 and m_3, while N_3 updates its own vectors for m_1 and m_2. If m_1 is invalid, N_2 and N_3 reject m_1 and the previous values of m_2 and m_3 as invalid. In time slot two, N_2 broadcasts m_2 and attests to whether m_1 and m_3 were valid. N_1 and N_3 update their vectors accordingly. At the conclusion of time slot two, N_3 has received both its direct authenticator for m_1 and the secondary confirmation from N_2. N_3 performs a unanimous vote on its validity vector entry for m_1 and the validity included in N_2's transmission. N_3 accepts the value of m_1 if both the direct authenticator was valid, the packet containing the secondary confirmation was valid and indicated m_1 was valid, and the value of m_1 was not received as 'lost.' This process continues over the next three time slots, each node voting once it has received the direct authenticator and secondary confirmation for each message type.

7.2 Properties and results

We verified the data origin authenticity property for each message type for all receivers using OFMC and Cl-Atse, backend components of AVISPA that check this property [1]. To test a transmitted variable for data origin authenticity, AVISPA uses a pair of functions: *witness* and *request*. These functions also implicitly test for data integrity. For each transmitted message, the sender executes the *witness* function. This indicates to the model-checker a node with a specific identity transmitted that value. Upon voting and accepting a message as valid, a receiver executes the *request* function. This function tests that the identity of the supposed sender and the value itself are the same as the ones specified in the corresponding *witness* function. If not, then the attacker has managed to successfully forge a packet.

AVISPA detected one trivial attack using parallel sessions starting in the same message round. This attack requires nodes the execute the same protocol twice simultaneously, accepting two values in each time slot. However, existing protocols do not allow transmission of multiple packets over a bus within a time slot.

After modifying the model to disallow multiple parallel sessions, AVISPA reported that the protocol was safe. AVISPA was not able to find any masquerade attacks, including tests where the attacker controlled one of the three nodes. While this model executes among only three nodes over one message round, it demonstrates that adding an indirect secondary confirmation from another receiver does not permit an attacker successfully forge values. This confirms our expectations, as we assume an attacker must successfully forge each MAC tag independently of others and a receiver only accepts a value if all direct and indirect authenticators agree on the value of a valid packet. However, because AVISPA assumes MAC tags are unforgeable unless an attacker holds the key, AVISPA cannot analyze the probability that an attacker successfully guesses truncated authenticators. We analyze our technique against simulated attack in Section 8.

8. Probability analysis

To spoof an individual packet to a single receiver, an attacker must successfully forge the authenticator designated for that receiver in the packet and all subsequent confirmations of validity. The probability of successfully forging a single secure MAC tag of *b* bits in length is 2^{-b}. When attempting to forge a subsequent confirmation, the attacker has two opportunities to succeed. First, the attacker may succeed in forging the *z* other MAC tags in the initial packet. For each initial attempt that fails (indicated by validity vectors in packets), the attacker must attempt to forge each subsequent confirmation. The first confirmation can be forged with probability $2^{-b} + 2^{-b}\,(1\text{-}2^{-b})$. The probability of successfully forging each confirmation beyond the first decreases slightly with each confirmation, because each attesting node performs a unanimous vote on its validity vector and each previous attester's validity vector. We do not attempt to assign an exact probability based on these tertiary interactions; instead we use $2^{-b} + 2^{-b}\,(1\text{-}2^{-b})$ as a conservative upper bound for each confirmation.

The probability P_p of successfully forging an individual packet with *z* subsequent confirmations and at most *w* compromised nodes is bounded by:

$$P_p \le 2^{-b}\left(2^{-b} + 2^{-b}\left(1 - 2^{-b}\right)\right)^{z\,-\,w\,-\,1} \qquad (1)$$

Using time-triggered authentication, receivers validate state-changing and reactive control messages over multiple packets for each message type they consume. In prior work, we have shown that the upper bound on the probability P_A of successful masquerade attack requiring *k* out of *n* valid time-triggered packets is [23]:

$$P_A = \sum_{i=k}^{n} \binom{n}{i} (P_P)^i (1 - P_P)^{n-i} \qquad (2)$$

8.1 Experimental results

We have experimentally confirmed the probability of successful forgery attacks against our approach using a embedded CAN network simulator written in Java [15]. We have modified the simulator to support TDMA scheduling and masquerade attacks. As per our attacker model, the simulated attacker may examine, modify, or replace any transmitted packet, so long as they obey the network schedule.

The simulated network consists of a set of nodes, broadcasting according to a round-robin schedule. Each node takes a turn sending, then the cycle repeats. The attacker selects one message type to forge, and attempts to fool a single receiver. After attempting to forge the initial packet, the attacker examines subsequent packets which attest to their forged packet. The attacker modifies any packets that indicate the initial forgery failed (visible to the attacker in the validity vector in packets). If the targeted receiver completes the Final Verification process and accepts the forged packet as valid and not lost, the simulator increments a counter for *successful packet forgeries*.

We measured the number of successful packet forgeries over a period of time long enough to record at least one hundred successful attack events per data point. We computed the *successful forgery rate* as average successful packet forgeries per message round and compared this rate to the probability of successful attack defined in equation 1.

Figure 6 shows the successful attack rate and the expected rate given by equation 1, varying the number of indirect secondary confirmations from zero to four and using two bits per receiver in each packet. Using only four confirmations decreases the probability of per-packet forgery by over two orders of magnitude, requiring four bits in the packet for the validity vector. To achieve a similar probability using only one MAC per receiver with zero confirmations, each MAC tag would need to be at least eleven bits. By using our voting mechanism, we only need three bits per receiver and four bits for the validity vector if we use four secondary confirmations, reducing authentication bandwidth costs by eight bits per receiver.

Figure 6 also shows the experimental results initially match the upper bound, then diverge from the upper bound as the number of confirmations increases. This is due to nodes performing unanimous votes with prior attesters and passing along the result, rather than simply sending whether they saw the initial authenticator as valid or not. We also carried out experiments using one to four bits per receiver, varying confirmations from zero to four, with results that similarly support equation 1. These experiments assumed zero compromised nodes.

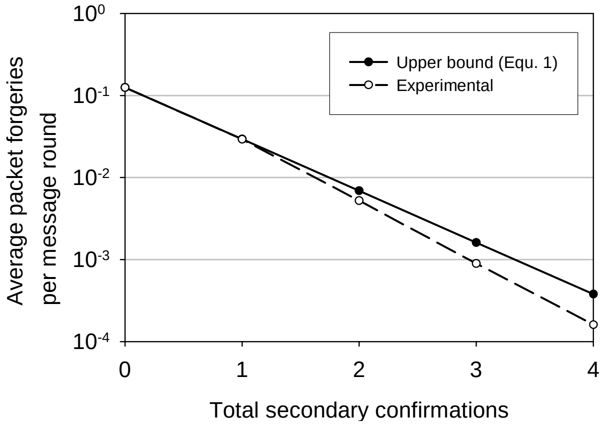

Figure 6. Simulated per-packet forgery rates using three bits per receiver, varying the number of secondary confirmations.

We also tested the effect of compromised nodes on the probability of successful forgery. Figure 7 shows the effect of increasing the number of compromised nodes on average attack events per message round. These experiments used three bits per receiver with a total of four secondary confirmations. The resulting successful packet forgery rates correspond to the same rates as those shown in Figure 6. Increasing the number of compromised nodes has the same effect on the probability of successful packet forgery as removing the same number of confirmations.

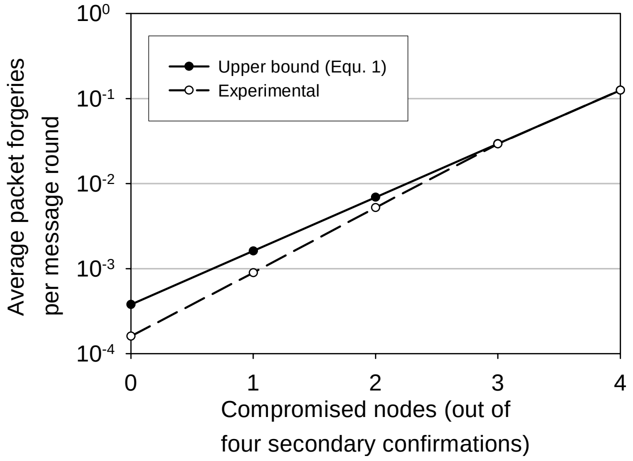

Figure 7. Simulated per-packet forgery rates using three bits per receiver and four total secondary confirmations, varying the number of compromised nodes participating in voting.

Figure 8 illustrates the effect of integrating our voting technique with our time-triggered authentication approach. Typical required failure rates for safety-critical systems might be defined at 10^{-3}/hr, 10^{-6}/hr, or 10^{-9}/hr. Figure 8 shows the number of authentication bits per packet and number of valid time-triggered packets to achieve a failure rate of 10^{-9}/hr using our time-triggered authentication approach alone (zero confirmations) and when combined with our voting technique (one, four, and eight confirmations). The number of packets and bits were obtained using the 10^{-9}/hr as an expected value for one forgery attempt per millisecond over the course of an hour, each succeeding with probability given by equations 1 and 2. For example, given four secondary confirmations, we can achieve an induced failure rate of 10^{-9}/hr using 3 bits per receiver over five time-triggered packets.

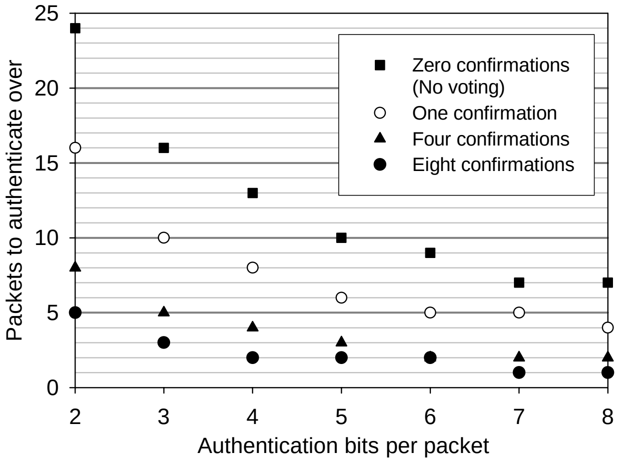

Figure 8. Authentication bits per packet and total packets to authenticate over required to achieve induced failure rate of 10^{-9}/hr on one message type broadcast once per millisecond.

9. Limitations

While this approach allows more efficient use of authentication bandwidth, it does have several limitations.

First, like our approach for time-triggered authentication, the per-packet bandwidth overhead scales nearly linearly with the number of receivers, limiting the maximum number of receivers in practice. With limited bandwidth for authentication, this approach cannot scale to hundreds or thousands of receivers. However, embedded networks typically have only tens of receivers.

Also, for simplicity this work assumes a statically scheduled TDMA network. Nodes must all have received the same set of message values by a particular time. This allows nodes to recompute authenticators over the same set of message values without explicitly retransmitting all values. Further, we rely on the periodic broadcasts of message types in time-triggered networks. Our voting technique partially alleviates this issue by significantly reducing the number of time-triggered samples required, even if only one secondary confirmation is used, as shown in Figure 8.

We also limit the number of secondary confirmations. First, the number of confirmations must be less than the number of initial receivers of a value. If a node did not receive a direct authenticator, they cannot attest to whether the value in a packet was valid or not. Second, for simplicity this work assumes that the confirmation packets of one sample of any message type does not overlap with the confirmation packets for any other sample for any message type broadcast by any node.

This approach also assumes a fixed number of compromised nodes to tolerate when determining the number of authentication bits, history buffer size, and secondary confirmations. If the number of compromised nodes exceeds this assumed number, no guarantees can be made about induced failure rates. However, in an embedded network containing critical nodes, if the attacker compromises more than one or two critical nodes they can likely cause the system to fail without resorting to masquerade attacks.

Lastly, this work does not address permanent faults (i.e., node failure) that permanently disrupt authentication of multiple message types. We plan to address this in future work. Also, we do not consider full DoS attacks intended to prevent delivery of all network traffic. Because as discussed by Wolfe et al. [25], there are numerous existing vulnerabilities in these networks to that type of attack (e.g., a node can prevent all traffic by simply broadcasting garbage values on the bus), and our scheme does not attempt to address DoS attacks.

10. Conclusions

In this paper, we present a new technique based on voting to improve overall bandwidth efficiency and reduce authentication latency. Specifically, we take advantage of the properties of secure MAC functions to vote on message value and validity amongst multiple nodes to reduce the probability of successful per-packet forgery, requiring only one extra bit per additional voter in each packet. We provide a conservative upper bound on per-packet forgery success and verify this bound through simulated attack. The model-checker AVISPA confirms data integrity and data origin authenticity of the voting mechanism. We leave a formal security analysis for future work. However, based on the results from model-checking, we do not anticipate this to reveal any vulnerabilities. Combining this voting mechanism with our prior work in time-triggered authentication allows system designers to reduce the per-packet bandwidth authentication costs or reduce application level latency while continuing to meet requirements for maliciously induced failure. While our scheme automatically recovers from transient faults with no additional overhead, in future work we plan to improve tolerance to packet losses due to permanent node failure.

11. Acknowledgements

This work is supported in part by the General Motors Collaborative Research Laboratory at Carnegie Mellon University.

12. References

[1] The AVISPA Project. Retrieved February 2010 from http://avispa-project.org/.

[2] R. Bosch GmbH, CAN Specification, Version 2, Sept. 1991.

[3] M. Brown, D. Cheung, D. Hankerson, J. L. Hernandez, M. Kirkup, and A. Menezes. PGP in constrained wireless devices. In *SSYM'00: Proc. of the 9th Conf. on USENIX Security Symposium*, pp. 19–34, 2000.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *INFOCOM '99: Proc. 18th Annual Joint Conf. of the IEEE Computer and Communications Societies*, vol. 2, pp. 708–716. IEEE, 1999.

[5] H. Chan and A. Perrig, "Efficient security primitives derived from a secure aggregation algorithm," in *Proc. ACM Conf. on Computer and Communications Security*, pp. 521–534, 2008.

[6] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, vol. 22, pp. 644–654, 1976.

[7] D. Dolev and A. C. Yao. On the security of public key protocols. In *SFCS '81: Proc. of the 22nd Annual Symp. on Foundations of Computer Science*, pp. 350–357. IEEE, 1981.

[8] FlexRay Consortium. FlexRay Communications System Protocol Specification, Version 2.1, Revision A, December 2005.

[9] G. Franklin, J. Powell, and A. Emami-Naeini. Feedback Control of Dynamic Systems. Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[10] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel and M.Walther. Time Triggered Communication on CAN (Time Triggered CAN - TTCAN). *7th International CAN Conference (ICC)*, 2000.

[11] S. Ganeriwal, S Capkun, C.-C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *WiSe '05: Proc. of the 4th ACM workshop on Wireless security*, pp. 97–106. ACM, 2005.

[12] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. In *CRYPTO '97: Proc. of the 17th Annual Int'l Cryptology Conf. on Advances in Cryptology*, pp. 180–197. Springer-Verlag, 1997.

[13] Y. Hu, M. Jakobsson, and A. Perrig. Efficient constructions for one-way hash chains. In *Applied Cryptography and Network Security*, pp. 423–441, 2003.

[14] C. Karlof, N. Sastry, and D. Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proc. of the 2nd Int'l Conf. on Embedded Networked Sensor Systems*, pp. 162–175. ACM, 2004.

[15] P. Koopman. Carnegie Mellon University. 18-649 Distributed Embedded Systems. Retrieved February 2010 from http://www.ece.cmu.edu/ ece649/.

[16] P. Koopman, J. Morris, and P. Narasimhan. Challenges in Deeply Networked System Survivability. *NATO Advanced Research Workshop on Security and Embedded Systems*, pp. 57–64, 2005.

[17] H. Kopetz. Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[18] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage. Experimental Security Analysis of a Modern Automobile, In *Proc. of the IEEE Symposium on Security and Privacy*, pp.447-462, 2010.

[19] J. Morris and P. Koopman. Critical Message Integrity Over A Shared Network. *5th IFAC Int'l Conf. on Fieldbus Systems and their Applications*, pp. 145-151, 2003.

[20] J. von Neumann. Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components. In *Automata Studies (Annals of Mathematics Studies, no. 34)*, pp. 43-99. Princeton Univ. Press, Princeton NJ, USA, 1956.

[21] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless Networks*, vol. 8(no. 5):pp. 521–534, 2002.

[22] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *SP '00: Proc. of the 2000 IEEE Symposium on Security and Privacy*, pp. 56–73. IEEE, 2000.

[23] C. Szilagyi and P. Koopman. Flexible multicast authentication for time-triggered embedded control network applications. In *DSN '09: Proc. of the Int'l Conference on Dependable Systems and Networks*, pp. 165–174, 2009.

[24] TTTech. Time-Triggered Protocol Specification TTP/C, Version 1.1, November 2003.

[25] M. Wolf, A. Weimerskirch, and C. Paar. Security in Automotive Bus Systems. *Workshop on Embedded Security in Cars*, 2004.

[26] C. K. Wong and S. S. Lam. Digital Signatures for Flows and Multicasts. In *ICNP '98: Proc. of the 6th Int'l Conf. on Network Protocols*, pp. 198–209. IEEE, 1998.