
Armin Wasicek, Christian El–Salloum, and Hermann Kopetz
Vienna University of Technology
Institute for Computer Engineering
Vienna, Austria
Email: {armin, salloum, hk}@vmars.tuwien.ac.at

Abstract—This paper investigates on the security of time–triggered transmission channels, which are used to establish a predictable and timely message transfer in a distributed embedded system with potential safety constraints. Within such a system, safety and security are closely related, because malicious attacks can have an impact on a system's safety and thereby cause severe damage. An attacker could masquerade as an original sender and try to alter some system parameters by injecting malicious messages in the system. In the embedded real–time systems domain particularly the authenticity of data items is of interest, because a lack of integrity can lead to incorrect or erroneous system behavior. In addition, we address the open research question how a common notion of time can contribute to a system's security. Our solution encompasses an authentication protocol to secure time–triggered transmission channels. We illustrate two attack scenarios (insertion and substitution) that aim at injecting fake messages in such a channel thereby corrupting the internal system state of a receiver. We discuss the feasibility of several key management strategies for embedded systems and describe an authentication protocol using time–delayed release of symmetric keys for time–triggered systems. In a case study we implement the protocol for a prototype Time–Triggered Ethernet (TTE) system. The insight gained from the evaluation is that the computation of the cryptographic algorithms consumes most resources. Our solution shows that authentication can be transparently applied to a time–triggered system exploiting the available global time base and without violating its timeliness properties.

I. INTRODUCTION

Today, embedded systems are a well established class of computer systems. They line up to support the driver of a car, to ensure the safe flight of an airplane, and they play an important role in industrial automation. Shortly, life without embedded systems is unimaginable. New developments in this field like increasing connectivity, extensibility, and deployment in untrusted and less controlled environments enable new critical failure modes regarding security that have not been an issue in the past.

Koopman states in [1] that “Security for embedded systems is an open question and could prove a more difficult long–term problem than security does today for desktop and enterprise computing.” An important cause of these emerging security requirements is that traditional embedded systems were operated in physically secured environments like within a nuclear power plant. Ubiquitous and pervasive computing propagates the distribution of embedded systems and computers in environments that can be hardly controlled and where even the owners of a system can present a security risk. It is of utmost importance that in the class of safety–critical embedded systems no security failure can propagate to the safety domain, because an attacker could target the whole sphere of control of the embedded system that encompasses its physical environment. Successful attacks could lead to catastrophic events like mechanical damage on the equipment, environmental pollution, financial loss, or as a worst case scenario the loss of human lives.

The goal of a time–triggered system is to provide a consistent computing base for distributed real-time systems and thus to facilitate the implementation safety–critical systems. By making time explicit, the temporal relations within a real–time system can be mapped to an accordingly built computer system. This is achieved by enabling a global time base, i.e., all participating nodes share a common notion of time. Whereas time–triggered systems provide an adequate design choice for safety–critical systems, not much research has yet been dedicated to their security. For this paper, we picked two driving research questions in this area formulated by Ed Lee [2]:

• What are the implications of time synchronized systems for security?
• Can security techniques effectively exploit a shared notion of time to improve robustness?

In this paper, we propose an authentication technique for time–triggered transmission channels using the time–delayed release of keys. We show that the global time base present in these systems can be effectively used to implement an authentication protocol using this technique. Our contribution encompasses:

• An attack model for time–triggered transmission channels
• An authentication protocol using the time–delayed release of keys
• An implementation of the protocol on a prototype Time–Triggered Ethernet (TTE) system

The remainder of this paper is structured as follows: Section II outlines the design of a time–triggered system, and Section III gives the problem statement and lists requirements. The next Section IV analyzes potential attacks on a time–triggered transmission channel. In Section V we describe an
authentication protocol for time–triggered systems. This is followed by an evaluation and discussion in Section VI. After referencing related work in Section VII, we draw a conclusion in Section VIII.

II. TIME–TRIGGERED SYSTEMS

The time–triggered paradigm [3] facilitates a solution for the design and implementation of dependable embedded systems. Its major goal is to provide a consistent computing base in a distributed realtime systems. The Time-Triggered Architecture (TTA) [4] instantiates the time–triggered paradigm for safety-critical environments.

At its core, the TTA defines a deterministic and predictable communication system by establishing a common notion of time (referred to as the global time base) among the connected nodes. By making the progression of physical time explicit, this global time base enables the temporal coordination of actions executed in different parts of the system. Moreover, the availability of a global time enables the precise specification of the interfaces among subsystems, which significantly contributes to the fault tolerance as well as to the security capabilities as will be shown.

Examples for common triggered actions are:
- Transmission of a message
- Execution of a task
- Output to controlled (real-world) object

Time–triggered systems use a linear model of time. The time line is divided in equidistant intervals with a duration $d_{\text{slot}}$ referred to as time slots, thus enabling a Time Division Multiple Access (TDMA) schema. Each time slot can be allocated to perform one of the actions listed above. This allocation is done during the design phase of the system. During runtime, the system access this a priori generated knowledge. Hence, conflicting resource requests can be resolved during the system design. Therefore, a basic requirement is that all nodes have a consistent view of the current time. Clock synchronization is used to build a virtual clock (i.e., the global time base) which is distributed among all participating nodes.

Figure 1 depicts allocation of slots to nodes and relation of slots to each other. The sequence of time slots in which each node sends its messages forms a round. After a round is completed, the next round, with the same temporal access pattern but possibly different frames, is started. The number of different rounds determines the length of the cluster cycle. After a cluster cycle is finished, the transmission pattern starts over again with the start of the next cluster cycle.

The TTA safeguards the temporal domain by a basic architectural element called a bus guardian. A bus guardian ensures fail silent behavior of a node with respect to the temporal domain by cutting off the node from the network outside the preallocated transmission intervals. The bus guardian can be deployed either local to each node or as a central guardian (e.g., within a switch [5]). However, our attacker model assumes that an attacker can override this mechanism and access the communication medium at arbitrary instants.

Summarizing, the time–triggered paradigm facilitates the building of synchronous systems using a common communication schedule. Synchrony is established by a clock synchronization service. All participant have a consistent view on the progression of time. A priori knowledge about the events (triggers) within a system are used to bind message transmission, task execution and output to an explicit instant, therefore avoiding situations that might lead to an inconsistent system state (e.g., through concurrent events) by design.

III. PROBLEM STATEMENT

The TTA [4] provides a consistent computing base for implementing distributed embedded real-time systems. It is mostly deployed in the domain of safety-critical systems. In order to cope with the emerging security requirements in the safety domain, dedicated security mechanisms preventing the propagation from the security to the safety domain have to be addressed.

Currently, the TTA implements the identification of connected nodes via the time–triggered schedule. In a time–triggered system, senders and receivers access the communication medium at a priori defined instances in order to exchange messages. By associating at most one message per channel to an instant, messages can be identified by their respective receive instants. The result of identification is a claim of identity. Hence, the TTA enforces identification through the node’s temporal specification. Authentication is the verification of that claim. Currently, there is no means to confirm a node’s authenticity.

Therefore, we analyze three properties of time–triggered transmission channels that are relevant to prove that a message originates from an authentic source:

1) **Sender Authenticity**: The binding of a sender’s true identity to its emitted messages. The absence of this property is if this binding is not verified and messages are considered as authentic by default.

2) **Channel Integrity**: All receivers perceive a message’s image like produced at the sender. This implies that a message has not been changed during transit and all receivers have a consistent view on the information it carries. An example threat to channel integrity is a replay attack.

In addition to the described authenticity properties, the security mechanisms shall facilitate the seamless integration with the computing infrastructure of a time–triggered system. The security mechanisms should not decrease the overall
dependability of the embedded system as a result of their integration. Therefore, qualities of time–triggered systems like timeliness and determinism must be maintained when introducing authenticity as a new non–functional property to time–triggered systems.

Scalability is a system property referring to its capability to be enlarged. In this work, we consider a system as scalable, if the number of participants can be enlarged without any significant unfavorable effect.

In terms of performance, the periodic real–time transmission of an environmental parameter gives a benchmark what throughput should be achieved. In some typical SAE\textsuperscript{1} class C automotive applications, the period of a control loop can be as low as 5 ms [6].

Furthermore, the solution should contain no single point of failure. This a basic design rule for building dependable systems. Considering security, an adversary will try to spot the weakest link in the chain.

IV. THREAT MODEL

To illustrate potential attacks, we assume a system model encompassing a distributed system with three nodes. Two regular nodes are connected via an unidirectional time–triggered transmission channel and a third malicious node attached to this channel. In this attacker model the malicious node is able to access the communication system in a way that it can inject valid messages at any given instant to the channel. It can thereby overwrite messages which are in transit (e.g., currently stored in a switch’s output queue or in the memory of a receiver’s network adapter). Furthermore, it can make use of a clock which is synchronized to the global time of the network.

Under the assumption of the attacker model described above, we introduce two attacks with respect to the sender authenticity and channel integrity:

1) Insertion attack: A new message is injected in an event-based message channel producing a faked event.

2) Substitution attack: A state message is replaced by a malicious message inducing a view on the environment which must not correspond to reality.

Both attacks are instances of masquerading failures [7] which occur if an erroneous node assumes the identity of another node and causes harm to the system. Opposed to the original definition, which describes a masquerading failure as accidental bit flips in the source or destination field of a message, in the presented attacks this modification is due to a malicious interaction fault. If the receiver of a message has no further knowledge about the sender’s identity, it must trust in the communication system to deliver authentic messages.

V. AUTHENTICATION

Authentication is required to distinguish between the authentic service and the corrupt service of a component. A corrupt service is a service deviating from the system’s intended use that has been induced by a malicious user or a service delivered to an unintended group of users. Authentication facilitates checks which determine a user’s identity. A usual next step is to perform authorization which assigns access rights to an authenticated user for a requested resource or service.

Authentication protocols are security protocols that facilitate the authentication of entities by means of cryptographic ciphers. A security protocol embeds ciphers in applications. It describes how the cryptographic and non-cryptographic algorithms should be used in order to establish a certain security guarantee for a given system. In order to distinguish between the authentic service and the corrupt service of a node, each node requires some unique information (e.g., a cryptographic key) for the verification of its identity (i.e., to enable authentication). This piece of information can be stored in a tamper–proof memory to prevent unauthorized physical access.

Authentication in computer systems can be facilitated by cryptography. The knowledge of a cryptographic key binds an identity to an entity. Basically, two kinds of ciphers can be distinguished:

- **Symmetric ciphers** use the same key for decryption and encryption. The key is shared between the entities.
- **Asymmetric ciphers** use different keys for decryption and encryption. Keys are assigned according to the role an entity performs in the protocol (encrypt or decrypt).

Both support the computation of authentication protocols, e.g., through a Message Authentication Code (MAC) (symmetric cryptography) or a digital signature (asymmetric cryptography) [8]. Usually, symmetric ciphers can be implemented more resource–efficient on a modern processor than asymmetric ciphers (factor 100 – 1000).

It is a common design rule for cryptographic systems, that keys have to be exchanged after a certain period of usage, because a collection of ciphertexts using the same cryptographic key is an advantage for an attacker trying to break the cipher.

A. Key Management

The key management and key distribution strategies are a crucial aspect in security protocols. They provide the infrastructure needed to use cryptography. Key management is concerned with the generation, maintenance, and revocation of keys, whereas key distribution facilitates the exchange of keys. Authentication is concerned with the assignment of an identity to a principal and the verification thereof. In brief, key management takes all actions that are required to propagate a key to the right location at the right time. The link between authentication (that requires a cryptographic key as part of the protocol) and key management is the act of key distribution which is used to propagate that assignment information to other principals and systems.

Authentication schemes for multicast and broadcast communication can pursue several different key management strategies per single channel:

\textsuperscript{1}Society of Automotive Engineers (SAE)
### Strategy 1. One for all:
One symmetric key is used for each sender and all corresponding receivers. This setup effects a simple key management, but it has the drawback that each participant could maliciously act as sender and therefore introduces a single point of failure. Furthermore, this strategy does not protect against masquerading within the group, because it assumes that all members trust each other. This assumption does not hold in many setups.

### Strategy 2. One for each:
The sender shares a single symmetric key with each receiver. This approach removes the single point of failure, but it does not scale very well, because for each additional receiver, the sender has to perform a separate computation of the cryptographic algorithm. This increases the protocol's overhead regarding computation, memory, and bandwidth proportional to the number of receivers. However, if the number of receivers is not too large, authentication can be realized quite efficiently (see [9]).

### Strategy 3. Public keys:
The authentication uses asymmetric cryptography: The sender possesses a private key and all receivers hold the corresponding public key. This scheme is a powerful solution for source authentication, but is not suitable for a system with stringent resource requirements [10] like an embedded system. The computational effort for asymmetric ciphers is hundred to thousand times higher than for symmetric cryptography. Moreover, public key cryptography requires the deployment of a sophisticated Public Key Infrastructure (PKI) that might not be practical in a constrained environment.

### Strategy 4. Time–delayed release of keys:
The sender holds a symmetric key and shares a common notion of time with all receivers. This key is disclosed by after its usage time has elapsed [11]. With the newly gained knowledge about the key, participants can authenticate previously received messages using this key. Hence, this strategy requires that all participants agree on a common notion of time and the usage periods of a key. The drawback of this approach is that the authenticity of a message can only be verified with some delay.

All four strategies require that (initial) keys are distributed before the actual protocol (i.e., computation of cryptographic checksums and exchange of messages) starts. This is a common requirement for security protocols, for example the Transport Layer Security (TLS) protocol [12] uses a three-way handshake protocol (using asymmetric cryptography) to initialize a session and to exchange session keys which are then used for data encryption (using asymmetric cryptography).

### B. Comparison of key management strategies
Table I displays experimental data on the computation of authentication tags that can deliver the postulated authenticity properties. An authentication tag is usually appended to a message. This will be used as basis for the comparison.

As described in the beginning of this section the basic means to compute authentication tags are digital signatures and MACs. Both use different approaches for the computation. Table I lists in the upper half the digital signature scheme RSA–PSS/SHA1 [13] as an example for a public key scheme. The lower half lists some representatives for different MAC schemes. Although better performing schemes than RSA–PSS/SHA1 exist (e.g., using elliptic curve cryptography), the numbers show that MACs are predominant in terms of computational performance.

The paper in [14] gives a comprehensive cryptanalysis of the security properties of MACs regarding an attacker whose goal is to try to inject a fraudulent message and append a MAC value which will be accepted by the receiver. A MAC can deliver the postulated properties from Section III. Strategies using a MAC are Strategy 1, Strategy 3, and Strategy 4. In the following, we discuss different MAC schemes as a means to efficiently compute authentication tags as required for a secure time–triggered transmission channel. Table II compares the different strategies according to the requirements from Section III.

Strategy 1 is the most efficient solution in terms of performance, because the protocol overhead in addition to computing one authentication tag per message is negligible. An extension of the number of receivers is no problem, because the sender must not be aware of additional receivers. It uses the same key for every receiver. Unfortunately, this single key represents a single point of failure. Every receiver could maliciously act as sender. Strategy 2 improves this drawback from Strategy 1, but lacks scalability, therefore it is best for a small and fixed number of receivers.

Strategy 3 is quite optimal for the listed requirements. We discarded this strategy, because of the experimental results show that even for small keys, the performance requirement of a transmission period of 5 ms does not hold.

Strategy 4 computes one authentication tag for all receivers, therefore it is quite optimal. Here, the drawback is the protocol overhead. For this work, we selected Strategy 4 and analyze, if the protocol overhead is significant. However, in a special application context, other strategies like ‘one for all’ or ‘one for each’ might be more feasible. The next part of the paper details on implementation issues for the time–delayed authentication.

### TABLE I

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key size</th>
<th>Block size</th>
<th>Payload</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(bits)</td>
<td>(bytes)</td>
<td></td>
<td>(s)</td>
<td>(s)</td>
</tr>
<tr>
<td>RSA–SHA1</td>
<td>256</td>
<td>min ( k - 2 )</td>
<td>30</td>
<td>0.0320</td>
<td>0.0038</td>
</tr>
<tr>
<td>RSA–SHA1</td>
<td>512</td>
<td>min ( k - 2 )</td>
<td>62</td>
<td>0.1830</td>
<td>0.0084</td>
</tr>
<tr>
<td>RSA–SHA1</td>
<td>1024</td>
<td>min ( k - 2 )</td>
<td>126</td>
<td>1.2033</td>
<td>0.0239</td>
</tr>
<tr>
<td>HMAC–SHA1</td>
<td>160</td>
<td>n.f.</td>
<td>20</td>
<td>0.0029</td>
<td>0.0029</td>
</tr>
</tbody>
</table>

Abbreviations: n.f. . . . not feasible

### TABLE II

<table>
<thead>
<tr>
<th>Requirement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seamless integration</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Scalability</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Performance</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>No single point of failure</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>
C. Time–delayed Authentication

Our approach is based on a variant of the TESLA protocol [15], which is a Guy Fawkes type protocol [16], on top of a time–triggered communication system. The resulting authentication protocol implementation is very efficient, because it can be embedded into the time–triggered schedule.

The message authentication protocol uses these three building blocks:

- An authentication tag, e.g., computed by a MAC
- A key chain linking several symmetric keys
- An a priori specified Time Division Multiple Access (TDMA) message schedule.

A key chains’ goal is it to establish a secure association between single keys. It is constructed by applying a hash function to a cryptographic key and using the function’s result as input to the hash function [17]. The repeated application of a hash function on its result yields a pseudo–random sequence of numbers. A key chain comprehends these numbers as cryptographic keys. It is an important characteristic of the key chain that it can be traversed only in one direction. The elements of the key chain are worked off in reverse order and the keys are used to authenticate single messages. By frequently exchanging keys, the channel can be protected against replay attacks.

Before the protocol execution, each sending node builds up a key chain of $K_0 \rightarrow K_1 \rightarrow \cdots \rightarrow K_n$ keys, where $K_i = F(K_{i-1})$ under a hash function $F(.)$. The root key $K_0$ is chosen randomly. The chain’s initial key $K_n$ has to be distributed to every receiver in the network in a secure manner, e.g., by using a Certification Authority (CA). This process is called bootstrapping the protocol.

The key chain is worked off in reverse order starting with $K_n$. A sending node assigns every key $K_i$ to a set of $d_{key}$ subsequent timeslots ($1 < d_{key} < n$) of the TDMA schedule. This parameter determines how long a single key is valid and is chosen by the system designer. Figure 2 depicts the assignment of timeslots to a key chain with a chosen $d_{key} = 2$. The timeslots must be sending slots and 'subsequent' means that they are temporally ordered on a discrete timeline.

This allocation of a key $K_j$ to a timeslot $slot_i$ implies that $K_j$ is used to compute a $MAC^{K_j}_M$ for each message $M$. This MAC is appended to message $M$ as well as the key $K_{i-1}$ which was valid for the previous set of timeslots, hence forming the message $M' = \{M, MAC^{K_i}_M, K_{i-1}\}$. Figure 3 shows the chronology of the protocol for $d_{key} = 1$. In $slot_j$ message $M'_j$ is transmitted. During the next timeslot ($slot_{j+1}$), the message $M'_{j+1}$ is sent together with a proof of its own authenticity $(MAC^{K_{i+1}}_{M'_{j+1}})$ and the previously valid key $K_i$, which is now used by the receiver to authenticate message $M_j$ of $slot_j$.

Thus, after a key’s validity period has expired and all messages tagged with this key are permanent at their receivers, it can be securely disclosed. A validity period can expire as soon as all messages sent within $d_{key}$ are permanent at all receivers. A particular message becomes permanent, if the receiving node knows that all the messages that have been sent to it prior to the actual message have arrived (or will never arrive) [18]. Now, the receiving nodes use the disclosed key in order to authenticate previously received messages. The authenticity of the key is protected through the secure association established within the key chain, because the receiver knows $K_n$ and it can reconstruct the key chain from $K_i$ to $K_n$.

In order to prevent the repeated use of cryptographic material, the current key $K_i$ can be hashed to obtain a new key $K_i' = F(K_i)$. This procedure is described in [15] to avoid that the same key is used for the MAC computation and the computation of the successor in the key chain.

D. Authentication and release delays

The release delay $d_{release}$ is the time a sender has to wait before releasing a key. In order to execute the protocol securely, the sender may release a key $K_j$ not until the receivers do not accept any more messages authenticated with $K_j$. The release instant of a key is given by

$$t_{release} > t_{send} + d_{max} + 2g$$

with $d_{max}$ being the maximum propagation delay and $g$ the granularity of the global time base. Note that in a time–triggered system the reasonableness condition 2 $g \geq \Pi$ (II being the precision of the clock ensemble) must hold for the global time base [3].

A receiver can determine the instant when to stop accepting messages authenticated with $K_j$ by using the message schedule and its local clock. Knowing the maximum propagation delay $d_{max}$, the receiver has to add $g$, because the sender and receiver clocks may differ at most one tick. Changing the perspective to the sender, it has to add one more $g$, because it cannot know whether the receivers are advanced.

2This reasonableness condition ensures that the synchronization error is bounded to less than one macrogranule, i.e., the duration between two ticks...
or delayed for this one tick. Finally, it is trivial to compute 
\( d_{\text{release}} = t_{\text{release}} - t_{\text{send}} \).

A receivers can start to authenticate a message as soon as 
the corresponding key has been released and received. The 
authentication delay \( d_{\text{auth}} \) is the parameter that determines the 
period of time that will pass between the start of transmission 
of a given message \( t_{\text{send}} \) and the point in time the released key 
has been received. The worst case would be two messages \( A \) 
and \( B \) scheduled at opposing ends of two subsequent \( d_{\text{key}} \) 
intervals. Hence, the message \( A \) has been received in the 
beginning of \( d_{\text{key}} \) interval and the subsequent message \( B \) 
which releases the key for message \( A \) is scheduled at the end 
of the subsequent \( d_{\text{key}} \) interval.

In a time-triggered system it is meaningful to align the 
temporal durations of a time-triggered system with the \( d_{\text{key}} \) 
parameter of the authentication protocol. If a node emits each 
round one message then \( d_{\text{key}} = d_{\text{round}} \) is a proper choice. 
Hence, a message can be authenticated in each subsequent 
round. This yields an authentication delay of 
\[
d_{\text{auth}} = d_{\text{round}} + d_{\text{max}} + 2g \tag{2}
\]

If it is not viable for the application that the data is 
processed by the receiver after \( d_{\text{auth}} \) (e.g., because this might 
add too much dead time in a control loop), it might be feasible 
that unauthentic input can be used. This is due to oversampling 
and system inertia of the underlying control system as pointed 
out in [9]. Under these assumptions, the data can be used right 
after reception and a later failed authentication check must be 
guaranteed to lead over the application in a safe state.

E. Rekeying

With a key chain of length \( n \) and choosing the duration of 
a round for \( d_{\text{key}} \), the protocol can work for \( (n - 1) * d_{\text{key}} * 
\( d_{\text{slot}} \) \) seconds. Before the key chain is exhausted, the sender 
has to choose a new root key \( K_0 \), generate a new chain, 
and the initial key \( K_n \) has to be confidentially transmitted to 
all receivers. Remember that only the sender has to store the 
whole chain and that the receiver builds an identical chain 
with the disclosed keys as part of the authentication process.

Figure 4 depicts an exemplary key chain. It contains three 
keys that work with the protocol and that are released over 
time, and two keys that remain confidential. In addition, the 
trailing element \( g \) is used to generate the root key \( K_0 \) and 
subsequently the whole chain. This element must not be 
disclosed otherwise the whole chain becomes insecure. The 
receivers get their starting element \( K_3 \) through a secure key 
distribution channel. With the initial key they can determine 
whether a newly released key is part of the current chain by 
checking \( K_{i-1} = F(K_i) \).

After a key chain has been worked off (i.e., the root key 
has been reached) the new key chain’s initial key must be available 
at the receiver. This circumstance represents a constraint on 
the minimal duration of \( d_{\text{key}} \). The selection of the length of the 
key chain is a trade off between memory usage for the key 
chain, computation time of a new key chain, and the effort 
required to distribute a new initial key. Several optimization 
techniques for the computation of a key chain are suggested 
in [19].

The background to the separation of the trailing element \( g \) 
and the root key \( K_0 \) is the idea that \( g \) can remain confidential. 
The reason behind this is to conceal the quality of the random 
number generator.

F. Bootstrapping

Several methods are applicable to bootstrap the protocol, 
i.e., to generate and distribute a chain’s initial key \( K_n \), to every 
receiver in the network in a secure manner. In most cases, the 
selected method will depend on the application context of the 
protocol and reflect the trust relationships within the system. A 
new initial key has to be present at all receivers in time, before 
the sender starts to work off a new key chain. Otherwise, the 
authentication of single messages will be delayed, or even 
more – if receiving queues are full – messages have to be 
discarded. Furthermore, the generation of random numbers 
which is a prerequisite for many cryptographic key generation 
protocols is not a trivial task in embedded systems, because 
appropriate entropy sources like hardware random number 
generators or random events from the environment (e.g., key 
strokes) are scarce [20].

It is out of scope of this paper to present a detailed discus-
sion of how bootstrapping can be realized. However, the list 
below outlines several appropriate methods to bootstrapping 
the protocol:

- **Sender centric**: Each sender generates and distributes 
  new keys itself. It can mix the new keys either in the 
  existing information flow to its receivers or it can require 
  a separate channel. This approach reflects Strategy 2 and 
  is taken in the \( \mu \)TESLA variant of the protocol [21].
- **Trusted third party**: A separate node is trusted by all 
  participants. This node generates all the required keying 
  material and executes a key distribution protocol with 
  all nodes. The original TESLA paper [15] suggests this 
  approach together with a digital signature scheme (Strat-
  egy 3).

VI. EVALUATION AND DISCUSSION

We implemented the protocol as described above on the 
prototype Time-Triggered Ethernet system developed at our 
department [22]. The evaluation pursues following goals:

- show the feasibility of the presented threat model in a 
  real-time system
- describe a proof-of-concept implementation of a time-delayed release of keys strategy in a time-triggered system.

A. Time-Triggered Ethernet

TTE\(^{2}\) [22], [23], is a descendant from the TTA. It is the goal of TTE to unify real-time and non-real-time traffic into a single coherent communication architecture. Therefore TTE supports two message classes, an event-triggered message class that conforms to standard Ethernet [24] and a time-triggered message class. For this work, only the latter class is of interest. The time-triggered message class is furthermore divided into two types:

- **Time-Triggered periodic Messages (TTpM)** are typically used to transmit state information and the controller triggers periodically their send operation.
- **Time-Triggered sporadic Messages (TTsM)** reflect the event characteristics of information. They are allocate a TDMA slot that is only used in case if the application explicitly instructs the controller to send a message. Note that this type is not to be confused with standard Ethernet messages.

A TTE system consists of a special TTE switch and several connected TTE controllers which communicate using the Time-Triggered Ethernet Protocol. One controller acts as rate master and establishes a global time base [3], [4] by a master–slave clock rate correction algorithm. The temporal specification of each controller is stored in a data structure called a Message Descriptor List (MEDL). A MEDL entry embodies information like action (send or receive a message), message transmission period, phase offset, message type and a couple of configuration bits like interrupt enable. TTE supports different periods for messages, the smallest period having a duration of 61,0\(\mu\)s.

B. Attack Scenarios

This section discusses how the substitution and insertion attacks can be implemented in a time-triggered system. Most safety-critical time-triggered systems implement bus guardians to prevent a single node from monopolizing the communication system. If deployed as a central guardian, this mechanism can also effectively prevent the injection of malicious messages in a time-triggered transmission channel thus facilitating channel integrity. By knowing the schedules from all nodes, a malicious message will be blocked by the central guardian before it is propagated to the addressed receiver. However, the presented authentication protocol can be used to determine the authenticity of the whole node facilitating sender authenticity. Hence, it enforces the detection of a node’s physical replacement.

Figure 5 depicts three attack scenarios (referred to as a,b,c in the following), how a malicious node can connect to the network. To facilitate a substitution attack, a replaced node can entirely substitute an original node (b), thereby sending in its allocated time slots. In a TTE system, the malicious node will emit the configured TTpM messages instead of the original node. The insertion attack can be used to corrupt a time-triggered sporadic channel. For instance, the malicious node can be connected in parallel (a) to the original node injecting TTsM messages in time slots in which the original node remains silent. An example for both attack scenarios are power boxes that are tuned engine controllers that can be assembled into a car to override the original ones from the manufacturer [25]. Examples for time-triggered systems using bus guardians are TTEthernet [26], TTP [27], and FlexRay [28].

In systems without bus guardians, channel integrity is not guaranteed by the communication system. In such a setup, the protocol can even detect malicious messages that compete with original messages for the bus and overwrite them in the communication system (e.g., within the output queues of a switch). Figure 5 depicts this in (c) by connecting a malicious node to an empty port on the switch. Examples for systems that are do not use bus guardians and could potentially profit from the proposed protocol are Powerlink Ethernet [29] as well as time-triggered systems based on the IEEE 1588 clock synchronization protocol [30].

C. Protocol Implementation

The protocol implementation consists of two parts, the authentication of messages at the sender and their verification at the receivers. The sender builds a key chain, subsequently authenticates messages and performs the rekeying. The receiver has to store incoming messages, verify the integrity of released keys, and verify the authenticity of stored messages.

As has been showed in Table I the computational effort of the required cryptographic algorithms is feasible for embedded systems even for those with limited processing capabilities. Another critical resource is the memory consumption of the protocol. A sender requires enough memory storage for the key chain. A key chain will most likely have a fixed size and will not be dynamically allocated. For example, consider a time-triggered transmission channel with one message per time-triggered round having a period of 5ns and a rekeying interval of 4s, which gives enough time for an asymmetric cipher to execute within one \(d_{key}\). Using the HMAC–SHA1, this requires 20 * 800 = 16000 bytes. Taking for instance the AT91SAM7XC512 micro controller this boils down to 12,2\%.

---

\(^{2}\)Please note that this evaluation uses an academic TTE prototype that is substantially different from the TTEthernet product series available from TTEtech Computertechnik AG. The switch described in this section follows another principle of operation than a commercial TTEthernet switch.
of its available main memory (128K RAM). The paper in [31] provides the figure of 446ms for an elliptic curve operation on an ARM9 micro controller. Given an rekeying interval of 4s this translates to approximately 11.2% of processor utilization during this interval. A single HMAC–SHA1 computation lasts for 2.9ms. Therefore, computing the authentication tags will require 2,32s or 58%. Note that the micro controller used for this evaluation has very limited computational capabilities.

A receiver requires significantly less memory space. Memory for two keys is needed, one for the released key of the past interval K_last and one for the released key of the subsequent interval K_curr in order to perform an integrity check on the key chain. If K_last = F(K_curr), key K_curr is part of the key chain and therefore originates from an authentic source, because of the collision resistance property of a hash function F().

Other required resources are codesize and bandwidth. The codesize of the HMAC–SHA1 is 2039 bytes in the text segment. Table III shows the protocol overhead composed out of an exemplary size for the MAC (16 byte) plus the size of the disclosed key (16 byte) to the maximal message size as percent ratio for different time–triggered protocols.

Summarizing, we can give an estimate for the overhead of an authentication protocol using the time–delayed release of keys on top of a time–triggered system that will be approximately 15% considering memory, and bandwidth usage based on all the provided figures. The computation of the cryptographic ciphers is with 69, 2% still the largest part in this equation. For a real–world application a stronger processor or hardware support for cryptographic operations to relieve the main processor is necessary.

VII. RELATED WORK

In the paper in [9] a shared key for each receiver is used. The protocol computes a MAC for each receiver and truncates it MAC before appending one for each receiver of the time–triggered multicast message. This loss of information is compensated by adding up several received messages before an authentication failure is detected. However, this approach saves valuable bandwidth, but introduces a linear dependency on the number of receivers. The authors assume that the number of receivers is small (’tens of receivers’) and fixed in the target systems.

VIII. CONCLUSION

In this paper, we presented an approach to facilitate authentication in time–triggered systems. More precisely, we designed an authentication protocol using the time–delayed release of keys. From this work, we can conclude that the time synchronization that is inherent to time–triggered systems can contribute to implement efficient authentication in an embedded system where computational resources are scarce and asymmetric cryptography ciphers are not a choice. In an evaluation we identified the computational effort required to compute the cryptographic ciphers as the most costly part of the protocol. Furthermore, we illustrated how the key management plays a major role when designing a security protocol. Our protocol shows how authentication can be executed, but we were only able to give some hints on how the required cryptographic keys can be transported to the right location in time. This issue requires a more detailed research. Our main contribution is that we showed how the global time base can be used to efficiently increase the security of a time–triggered system.

ACKNOWLEDGMENT

This document is based on the ACROSS project in the framework of the ARTEMIS programme. The work has been funded in part by the ARTEMIS Joint Undertaking and National Funding Agencies of Austria, Germany, Italy and France under the funding ID ARTEMIS-2009-1-100208. The responsibility for the content rests with the authors.

REFERENCES

efficient stream loss-tolerant authentication (tesla): Multicast source
authentication transform introduction,” RFC 4082, Internet Engineering
[16] Ross J. Anderson and Francesco Bergadano and Bruno Crispo and Jong–
Hyeon Lee and Charalampos Manifavas and Roger M. Needham, “A
new family of authentication protocols,” ACMOSR: ACM Operating
[17] L. Lamport, “Password authentication with insecure communication.,”
Communications of the ACM, vol. 24, no. 11, pp. 770–772, November
1981.
[18] V. Paulo, “Ordering and Timeliness Requirements of Dependable Real-
Traversal,” Lecture Notes in Computer Science, vol. 2742, pp. 270–285,
2003.
pp. 1–12.
security protocols for sensor networks,” Wireless Networks, vol. 8, no. 5,
System,” Ph.D. dissertation, Technische Universität Wien, Institut für
[23] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, “The time-
triggered ethernet (tte) design,” in 8th IEEE International Symposium on
Object-oriented Real-time distributed Computing (ISORC), May 2005,
pp. 22–33.
Traktoren,” Francisco Josephinum Wieselburg, Rottenhauserstrasse 1,
A3250 Wieselburg, Bericht BLT053322, November 2006.
[26] TTTech, “TTEthernet - A Powerful Network Solution for All Purposes,”
[27] ——, “Time-Triggered Protocol TTP/C High-Level Specification Docu-
ment Protocol Version 1.1.” TTTech Computertechnik AG, Specification
V 1.5.2, January 2004.
[28] F. Consortium, “FlexRay Communications System Protocol Specifi-
[29] E. P. S. Group, “Ethernet Powerlink Brochure,” Online; Accessed 11-
Elliptic Curve over prime fields on an ARM946E,” in IEEE Military
Communications Conference (MILCOM), 2008.