Formally Verified Credentials Management for Industrial Control Systems

Tomas Kulik
Aarhus Univerity
Aarhus, Denmark

Jalil Boudjadar
Aarhus University
Aarhus, Denmark

Diego F. Aranha
Aarhus University
Aarhus, Denmark

***Abstract*—The field of industrial automation is experiencing growth in interconnectivity and digital interaction. This growth is slower than in a consumer segment due to often critical nature of industrial control systems. Security of such systems is an important aspect as malicious behaviors could lead to potential system malfunction, injuries or financial losses. As control networks are becoming more complex, having a robust credential management for system operators and users that could interact with the system components is an essential need. One way of assuring the robustness of the credential management is by using formal methods. In this paper we present a formally verified credential management system for use within industrial control systems. We demonstrate that the credential management can use centralized credential storage with secret passwords available only to system administrators. We use UPPAAL to formally analyze security properties based on requirements defined by our industrial partner and present the viability of formal verification to a real-world industrial case study.**

***Index Terms*—Credential management, Formal verification, Model checking, UPPAAL.**

I. Introduction

Modern industrial control systems offer an automation of application functionality that is interactive to an open environment, such as remote access to secure real-time monitoring and configuration by engineers, maintenance personnel and vendors. Such systems rely heavily on automated security protocols to secure access and integration of new devices given the risks related to the system interconnectivity [11].

Key and credential management protocols have been widely adopted and successfully demonstrated to secure different applications [15], [19] following the ubiquitous automation of control systems due to cheap computation resources. While traditional industrial networks are typically protected by introducing bump-in-the-wire devices to encrypt serial communications or tunnel legacy protocols through encrypted connections, modern approaches support encryption or authentication closer to the protocol itself [1], [4]. OAuth2 [22] and RADIUS [6] are industry standards widely deployed in practice to provide user and device authentication in many settings. Recent research has proposed advanced protocols where manufacturer-provided secrets allow for assigning and validating credentials on the fly with authorization information encoded directly in the keys [20], [25].

Given the high interoperability and connectivity of such systems for potential reconfiguration or to replace existing components and integrate new modules, maintaining the functionality and security level expected by the proprietary industrial control system becomes challenging [10], [18]. This is aggravated by the need to support different authentication mechanisms potentially coming from different vendors such as certificates, passwords, cryptographic keys and other credential types. Current solutions for this problem deployed in production rely on a trusted, possibly redundant, centralized system. A possible implementation for key and certificate management consists in a tamper-proof device embedded into the system, for example the *keyvault* proposed in [12].

Vulnerability of authentication protocols stems usually from faulty design, inconsistent implementations and uncertainties related to integration of new components and potentially malicious users behavior [16]. Examining the reliability and functionality of credential management protocols online is a complicated task due to real-time functionality. Formal methods to verify security and integrity of credential management protocols have been thoroughly used as a solid ground in the literature [21], [23]. This provides assurances not obtainable by use of post deployment testing.

This paper presents a formally verified credential management protocol for a dynamic industrial control system where the network security depends on the access to network switches. The credential management uses a cascade of authentication processes to access different services with secret passwords available only to system administrators. We use UPPAAL to formally analyze security properties based on requirements defined by our industrial partner. The rest of the paper is organized as follows: Section II presents the architecture of the industrial control systems and the key management infrastructure. Section III introduces the protocols and cryptographic elements used within this work, security requirements for the credential management system and briefly describes UPPAAL. Section IV is a formal modeling of both system behavior and credential management protocol. Section V presents and discusses results from the analysis of the security properties conducted using UPPAAL. Section VI presents related work and finally Section VII concludes the paper and discusses future plans.

II. System Architecture

This section presents the architecture of the industrial control system we consider and the credential management

infrastructure. The control system architecture consists of (1) the operations terminal (OT) providing local access to the control subsystems for control engineers, (2) network switches providing the network access to system controllers and (3) the *keyvault* providing credential management services to the OT and the switches within the control network. The control network further consists of controllers executing the industrial control, however in order to focus on components requiring credential management, these are omitted. The architecture is based on a control system used by our industrial partner, where we add the keyvault component as a new subsystem. The full architecture is shown on Figure 1.

OT is a local engineering terminal used by operators to interact with the system. Within the scope of this paper, it is considered a single point of access to the control system. OT is used as a client to login to the system and, once logged in, user can attempt to access and modify behavior of the network switches, for example by updating different network settings. Apart from providing access to the system, OT also displays different messages generated by the system.

The switch is a networking device used to interconnect different industrial controllers. As the industrial control process is distributed among these controllers, the switch is considered a critical device since in case of malicious behavior it can cause disruptions to the control process itself. In order for the user to be able to interact with the switch, the user needs to be logged in towards OT and further provide valid switch credential. The credential is not known to the user before hand and could be only obtained from the keyvault once the user is authenticated and has a correct role.

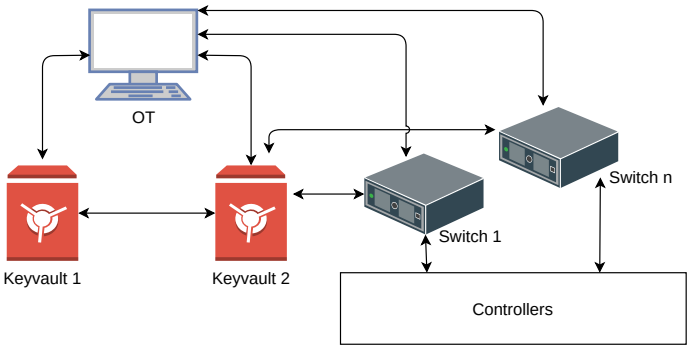

Fig. 1. Credential management system architecture

The keyvault is a centralized system for credential and key management.

In our previous work we have proposed similar system for cryptographic key management [12], it however omits user credential management. The credential management differs from key management since it requires user interaction. In order to ensure the system availability for user interaction, the keyvault is deployed in a redundant setup where the primary keyvault component is active and, in case of disruption of this component, the secondary component takes over operational tasks transparently. In this system the keyvault acts as a credential server, i.e. the users login towards the keyvault and are issued a token, which shall be included in subsequent requests. Furthermore, the keyvault is the only component with knowledge of the current passwords for different switches within the system, hence operators need to obtain this password from the keyvault in order to login towards a switch.

In this paper we consider that the system has passed the commissioning phase, i.e. the random credentials for different switches have been generated as well as a user database stored within the keyvault.

It is important to note that the entire system resides on a mobile platform with limited physical access, hence we consider physical attacks such as removing or disconnecting a switch unlikely to be carried out by unauthorized personnel.

III. Background

This section provides the background of our work, mainly cryptography principles and protocols, shared services for the control network, the verification tool and the security properties for key management.

A. The UPPAAL verification tool

UPPAAL is a tool-chain for analyzing concurrent real-time systems that can be described as a network of non-deterministic processes with finite control structure and real-valued clocks [14]. Processes are described as timed automata which is an extension of classical finite-state automata that introduces clocks to describe real-time behavior. Such automata use predicates, Boolean, integer variables and user defined types to model execution states and based on this it is able to carry out model checking. Communication between processes occurs via channels or shared variables.

B. Authentication protocol

We consider the authentication workflow described in Figure 2. The protocol implements single sign-on for a user authenticated with the OT to access any switch, without having to know specific credentials. Authentication starts with a user equipped with login credentials accessing the OT. The OT forwards the login credentials to the keyvault for validation. In case they are considered valid, the keyvault returns a *token* for identifying the authentication attempt and requests information about what switch the user would like to further access. The workflow continues with the user selecting a switch and receiving a matching *ticket*, which is forwarded to the switch for validation. If positive, a session with identifier *id* is opened.

The protocol allows flexibility in the implementation of the ticket validation process. Basically, the ticket can store any information that the switch can validate itself as authentic, or possibly forward to the keyvault for validation through a backend connection. This allows multiple design choices with performance-functionality trade-offs. Implementation choices can thus range from simple and efficient random strings as one-time passwords generated by the keyvault, to more computationally expensive public-key cryptography. An advantage of the former is not storing any sensitive information in the ticket

itself. This option does not require any care when handling it within the OT, which is why we consider it from here on. As a side note, an example of the latter could be a digital signature computed by the keyvault to attest a successful authentication, in case the user must prove in a verifiable way to a third party that such an authentication occurred.

In order to allow reuse of existing standards, the protocol attempts to capture the main features of solutions widely adopted by industry standards, so an implementation can be built using off-the-shelf components. We considered several solutions to instantiate our proposed protocol and explore the design space in a practical deployment, in particular the RADIUS and OAuth2 protocols. It is our expectation that the combination of protocols for purposes as presented in this paper can go beyond use by our industrial partner and can be widely adopted in within the industrial control system domain. This is where formal verification can provide answers in regards to feasibility of the presented approach of combining protocols and hence allow other industrial entities determine if the subsystem presented in this paper is deployable within their system.

1) RADIUS: The Remote Authentication Dial In User Service is a *de facto* standard for authentication in wireless networks and other networked infrastructures, widely deployed in devices manufactured by multiple companies. In RADIUS, a *Network Access Server* (NAS) receives requests from a client to access a network resource using a set of credentials. The NAS in turn contacts the RADIUS server to validate the set of credentials through a backend connection, which can result in access being granted or rejected. A challenge-response authentication mechanism is also supported, in which the RADIUS server returns a challenge to the client in a way to implement more sophisticated authentication workflows where the NAS does not receive the authentication credentials in plaintext.

In terms of security, previous analyses have found the RADIUS protocol to not be cryptographically sound, as it requires using an insecure hash function (MD5) as a self-synchronizing stream cipher in a customized construction [7] to encrypt the credentials in transit. However, these concerns are easily resolved by adopting the RadSec extension [26], which basically eliminates the reliance on MD5 by transmitting the protocol messages through a SSL/TLS connection, which could be supported in our use case with a certificate infrastructure [12]. Another security concern left unspecified is how to protect the backend connection between NAS and RADIUS server, but this could again be resolved by using SSL/TLS with certificates in place issued by the keyvault.

2) OAuth2: It is an industry standard for delegated and federated authentication in single sign-on scenarios. The protocol is highly complex with many different workflows tailored for multiple scenarios. It assumes multiple entities: a resource owner capable of granting access to a protected resource, a resource server hosting the protected resources, and capable of accepting and responding to protected resource requests using access tokens; a client application masking resource requests on behalf of the resource owner, and an authorization server issues access tokens to the client after successful authentication.

A first obstacle with an OAuth2 instantiation is mapping the entities in our scenario to the prescribed roles. While the OT is clearly represented by the client, the switch could both represent the resource to be accessed *or* the resource server hosting resources. The keyvault accumulates the roles of a resource owner and authentication server. Semantically, the switch is not a resource owned by the client inside the network, and there is no separation between the entity owning the resource and the authentication server. Practically this means that an implementation of our proposed protocol would have OAuth2 and a RADIUS phase. The OAuth2 phase will be used to let the user login to the system itself (i.e. the interaction application running within the OT) in order to become authorized to request a switch login. The action of user logging onto the switch is then handled using RADIUS.

IV. Behavior Modeling

We define the behavior of the system using a labeled transition system (LTS) [2] with a set of states S, initial state s_0 and a transition relation $\rightarrow$, given by a tuple $\langle S, s_0, \rightarrow \rangle$. The transitions are labeled with actions from an alphabet of actions of the respective system components. Specific transitions define guards specifying the conditions to enable the respective transition. We further introduce the following definitions:

- S refers to a set of states while s denotes a single state.
- S^{comp}, $comp \in \{kv, ot, sw\}$ represents the set of states of keyvault kv, OT ot and a switch sw respectively.
- A refers to a set of actions while a denotes a single action.
- A_{comp} refers to an alphabet of actions of a component as $A_{comp} \subset A$.
- V refers to a set of variables while v denotes a single variable.
- Z refers to a set of constants while z denotes a single constant.
- G refers to a set of guards while g denotes a single guard defined as:
  $g \triangleq v < z \mid v > z \mid v = z \mid g \vee g' \mid g \wedge g' \mid true \mid false$.
- $\rightarrow$ is a transition relation such as $\rightarrow \in S^{comp} \times G \times A_{comp} \times S^{comp}$.
- Component type $Comp_T$ is given as a tuple of two elements, its behavior β and parameters $\mathcal{P}$ such as $\langle \beta, \mathcal{P} \rangle$.

Further, we define each system component template with the parameter $\mathcal{P} : \mathcal{P} \subset 2^V$ and behavior β as:

$$Comp_T \triangleq \langle \mathcal{P}, \beta \rangle : \beta = \langle S^{comp}, s_0^{comp}, \rightarrow \rangle$$
$$\text{and } \rightarrow \in S^{comp} \times G \times A_{comp} \times S^{comp}$$

The behavior is first defined in the LTS notation as introduces, in order to allow for generally specify the system, and then this specification is translated to a specific tool, in case of this paper UPPAAL. The LTS notification is part of a

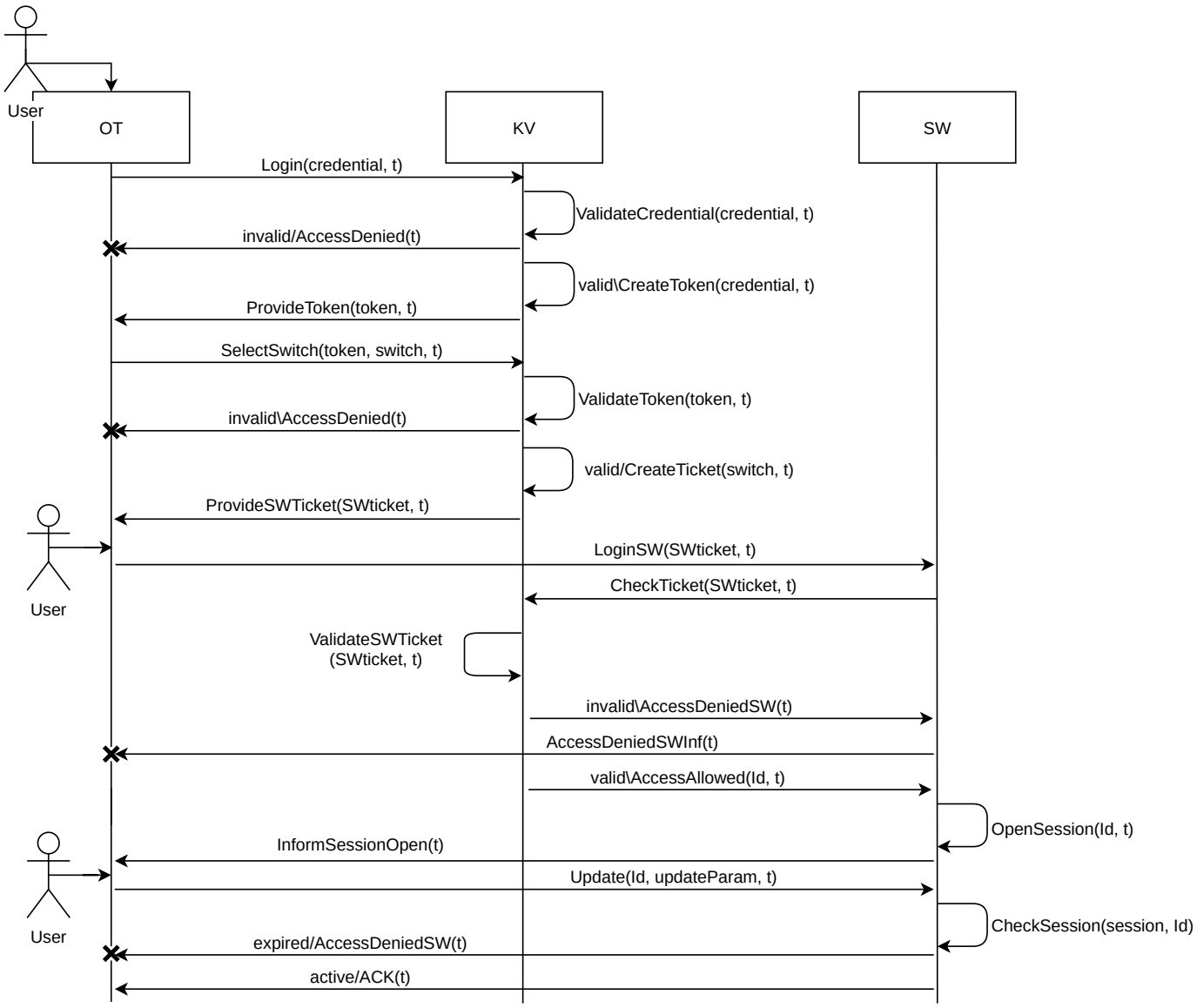

Fig. 2. System and Switch access protocol

larger system definition with intention to potentially utilize different tools and their formalisms, hence need to have a general model that can be translated to these formalisms. The behavior β of different components is understood as timed automata templates of the different components as for example a timed automata model of the switch as shown in Figure 5.

A. Keyvault behavior

The keyvault is the central part of the credential management system. It can perform several actions given by an alphabet A_{kv} shown in Table I. Actions can be parametrized, for example *ValidateCredential(cred, t)* where parameter *cred* represents the credential received by the keyvault from OT and *t* represents the time when the action occurs. This can be expressed as transition:

$$s \xrightarrow{\textit{ValidateCredential(cred, t)}} s'$$

We further define guards and variables of the keyvault as $G^{kv} = \{$*ValCred, ValToken, ValTicket*$\}$ and $V^{kv} = \{$*credDB, tok_exp, tic_exp, act*$\}$ respectively. The keyvault template is instantiated as: KV = KV$_T\{$*credDB, roleDB, tok_exp, tic_exp, act*$\}$.

credDB is a set representing a database of credentials loaded onto the keyvault at commissioning. We consider this database static and do not model potential updates. The database contains credentials of form: *cred:* $\langle Id, password\rangle$. *roleDB* represents a database of roles that the credentials can be associated to, the considered roles are *role* $\in \{admin, user\}$. The credentials map to the roles in a one-to-one mapping as *credDB*$\mapsto$*roleDB*. The login requests against the keyvault are validated by the guard *ValCred* as:

$$\textit{ValCred(cred, t)} \triangleq \textit{cred} \in \textit{credDB}$$

tok_exp is a predetermined offset indicating the temporal validity of access tokens issued by the keyvault. These tokens are of form *token:* $\langle$ *Id, role, issue_time, expiry_time*$\rangle$. The keyvault validates each request that contains a token to ensure that the token is not expired using a predicate *ValToken* as:

$$\textit{ValToken(token, t)} \triangleq \textit{token.expiry\_time} > t$$

Since the keyvault's main task is to provide a switch access ticket to successfully authenticated users with administrative role *admin*, the guard is extended as:

$$\textit{ValToken(token, t)} \triangleq \textit{token.expiry\_time} > t \wedge token.role = admin$$

Once the user has a valid token and requests an one-time access ticket, the keyvault generates a switch access ticket as *ticket* $\in$ *SwTickets* where *ticket*:$\langle$ *uId, swId, expiry_time* $\rangle$ with the *uId* representing the user identity as read from the token, the *swId* is the identity of the switch that the user requested access to and the *expiry_time* is the time when the access ticket expires. Furthermore the *SwTickets* represents the set of issued switch access tickets contained within the keyvault. The switch access ticket could be sent to the switch by the user, where the switch would again request the keyvault to validate it. The full interaction protocol is illustrated in Figure 2. The user login to the system and the consequent login to the switch are based on protocols introduced in Section III. The keyvault validates the switch access ticket as follows:

$$\textit{ValTicket(ticket, t)} \triangleq \textit{ticket.expiry\_time} > t \wedge ticket \in SwTickets$$

In case that the ticket is determined as not valid, the access to the switch is denied and the user has to request new access ticket from the keyvault in order to gain access to the switch.

In order to provide redundancy within the system for reliability concerns, it is considered that two keyvaults are present within the system at any time, following a hot standby procedure. In this procedure, in case that the primary keyvault kv_p experiences issues the system switches to the secondary keyvault kv_s. To provide transparent move from the primary to the secondary keyvault, the two keyvaults keep consistent state among themselves. In scope of this paper, this means that the keyvaults synchronize their *SwTickets* database. We express the *Consistent* predicate as:

$$\textit{Consistent}(kv_p, kv_s, t) \triangleq kv_p\textit{.SwTickets} = kv_s\textit{.SwTickets}$$

In case that the change from primary to the secondary happens, the secondary keyvault becomes a primary and the primary keyvault becomes secondary. This is represented by the keyvault variable *act*, where *act* $\in \{true, false\}$ as keyvault with *act=true* is the keyvault currently used operationally within the system. The keyvault that is broken down requires a human interaction in order to be repaired, hence this procedure is out of scope of this paper.

B. Operations Terminal behavior

OT is a local point of access terminal to the system. It can perform several actions given by an alphabet A_{ot} as shown in Table I. As with the keyvault several of the OT actions can be parametrized, for example *Login(cred, t)* where this action synchronizes with the *ValidateCredential(cred, t)* of the keyvault. OT does not define guards or variables, hence the OT template is instantiated as:

$$OT = OT_T\langle\rangle$$

Since OT is the primary human-machine interface for the system, we consider the following assumptions:

1) OT is present in an area with restricted physical access.
2) OT connects towards the keyvault on a secure local network.
3) OT provides an application to the user to access the system.

C. Switch behavior

The switch is a network device deployed within the control network providing interconnectivity for various system controllers. The switch can perform several actions given by the alphabet A_{sw} as defined in Table I. We further define guards and variables of the switch as $G^{sw} = \{ValSession\}$ and $V^{sw} = \{swId, \textit{session\_exp}\}$ where Id represents the unique identifier of the switch and *session_exp* represents the preset offset for session expiry time. The switch template is instantiated as:

$$SW = SW_T\langle Id, \textit{session\_exp}\rangle$$

The user can login to the switch by providing a keyvault-generated one-time access ticket, which the switch sends further for validation to the keyvault. Receipt of the ticket is represented by the action *LoginSW(swId, ticket, t)*. Since the switches within the system need to be addressable individually, each of the switches has a unique identifier, expressed as:

$$\forall i, j, sw | i \neq j \implies sw_i.swId \neq sw_j.swId$$

Once the switch successfully validates the access ticket against the keyvault, it opens a session. The session is expressed as *session*:$\langle$*swId, Id, expiry_time*$\rangle$, where *Id* is the identity of the ticket opening the session and *expiry_time* is the expiry time of the session. In order for the user to interact with the switch, the session has to be valid, i.e. not expired, this is represented as guard:

$$\textit{ValSession(session, t)} \triangleq \textit{session.expiry\_time} > t$$

The switch is considered a single user system, hence only one session could be open at any time point. Once the session expires, the user has to request a new one-time ticket in order to interact with the switch further.

D. System behavior

We define the system behavior *Sys* as a parallel composition of the component instances synchronizing on a defined set of events (Table. I) [3]. The system comprises two keyvault units and a possibility for multiple switches:

$$\begin{aligned} \mathit{Sys} \triangleq \langle \; \|_i \; & \mathit{SW}\langle Id, \mathit{session\_exp}\rangle \\ \| \; & \mathit{KV}_p\langle \mathit{credDB},\ \mathit{roleDB},\ \mathit{tok\_exp},\ \mathit{tic\_exp},\ \mathit{act}\rangle \\ \| \; & \mathit{KV}_s\langle \mathit{credDB},\ \mathit{roleDB},\ \mathit{tok\_exp},\ \mathit{tic\_exp},\ \mathit{act}\rangle \\ \| \; & \mathit{OT}\langle\rangle\rangle \end{aligned}$$

During execution, the system can either advance due to the transition of a single component instance or due to a synchronization of compatible transitions among multiple component instances. The overview of all actions including their synchronization is shown in Table I.

V. Formal Security Analysis

This section describes the formal security analysis of the credential management system using UPPAAL model checker. First, we formally express the security properties of interest, then we show how the properties have been verified within UPPAAL. Finally, we discuss the results of the verification and its impact on the system design.

A. Confidentiality properties

One of the important properties of the credential management system is the assurance that the user, including the administrator, never learns the credential for switch access. Within the system this is handled by issuance of one time tickets, however it is important to ensure that all generated tickets are unique and cannot be reused. We express this as:

$$\forall i, j | i \neq j \implies \mathit{ticket}_i \neq \mathit{ticket}_j$$

B. Authentication properties

Another important property of the credential management system is the assurance that only administrators could obtain a login ticket for the switch. This is important since the switch reconfigurations shall be only carried out by system administrators. We express this property formally as:

$$\begin{aligned} \forall t, token | & \mathit{token.role} \neq \mathit{admin} \\ & \wedge (t > \mathit{token.issue\_time} \wedge t < \mathit{token.expiry\_time}) \\ & \implies \neg \exists \mathit{ticket} | \mathit{ticket.uId} = \mathit{token.Id} \end{aligned}$$

C. Freshness properties

Freshness is an integral property of the credential management system. This property guarantees that the logins only last for a reasonable amount of time. This lowers the attack surface against replay attacks based on token or ticket theft, as they would have to be applied in a short time frame. The same freshness must hold for the switch sessions, where a long-lived login could persist even though the administrator is no longer using the system, and any user could take advantage of this open session. We express the freshness property formally as:

$$\begin{aligned} & \forall \mathit{token}\ \exists t | \mathit{token.expiry\_time} < t \\ & \wedge \\ & \forall \mathit{ticket}\ \exists t | \mathit{ticket.expiry\_time} < t \\ & \wedge \\ & \forall \mathit{session}\ \exists t | \mathit{session.expiry\_time} < t \end{aligned}$$

Once the expiry times of different entities (token, ticket, session) are reached a corresponding login mechanism must be used again to grant the entity with a new expiry time.

D. Availability properties

The availability property represents the expectation that at any point in time, the system shall have an available primary keyvault that contains all the necessary login information. This means that the keyvault must satisfy the *Consistent* predicate as well as being operationally active. We express this formally as:

$$\forall t\ \exists i |\ kv_i.act = true \wedge \mathit{Consistent}(kv_i, kv_s, t)$$

This property of the system is important since if no keyvault is available it is not possible to login to system components protected by the keyvault. For an industrial control system this could result in a loss of control and potential damage to the system. The availability property has not been analyzed in this paper due to the decision of the industrial partner to focus on a system with single keyvault only. We have however prepared this property as proposed next step will extend the system with the second keyvault.

E. Formal verification in UPPAAL

Following the authentication protocol described in section V-B, we have created a model in UPPAAL capturing the behavior of this protocol. In this section, we provide an overview over parts of the model and describe how the different security properties are expressed. The full model is available via [13]. The model has been built manually based on the system behavior provided in section IV.

The UPPAAL template modeling the keyvault process, illustrated in Figure 3, describes the synchronizations of the keyvault with the OT, shown in Figure 4 and the switch, illustrated in Figure 5. This includes handling of login requests from the OT, validation of issued tokens, generation of one-time-access tickets for the switch and validation of these one time access tickets. The protocol is initiated by the OT login request using the `Login[Cred]!` and `Login[Id]?` synchronizations, sending the selected credential to the keyvault. This synchronization uses the `chan Login` channel with a parameter determining which credential from the array of possible credentials is to be used. Once the credential is received, it is validated and the system progresses based on the result of this validation. This is expressed by the guard `validCredential`. In case that the validation is successful, the keyvault issues a token that is then provided to OT. The token

TABLE I
OVERVIEW OF SYSTEM ACTIONS AND THEIR SYNCHRONIZATIONS

A_{ot}	A_{kv}	A_{sw}
Login(cred, t)	*GetLogin(cred, t)*	-
-	*ValidateCred(cred, t)*	-
AccessDenied(t)	*SendAccessDenied(t)*	-
-	*CreateToken(cred, t)*	-
GetToken(token, t)	*ProvideToken(token, t)*	-
SelectSwitch(token, SWId, t)	*SwitchReq(token, SWId, t)*	-
-	*ValidateToken(token, t)*	-
-	*CreateTicket(SWId, t)*	-
GetSWTicket(ticket, t)	*ProvideTicket(ticket, t)*	-
LoginSW(ticket, Id, t)	-	*GetLogin(ticket, Id, t)*
-	*GetTicket(ticket, t)*	*CheckTicket(ticket, t)*
-	*ValidateTicket(ticket, t)*	-
-	*AccessDeniedSW(t)*	*GetAccessDenied(t)*
AccessDeniedSW(t)	-	*SendAccessDenied(t)*
-	*AccessAllowed(Id, t)*	*GetAccessAllowed(Id, t)*
-	-	*OpenSession(Id, t)*
GetSessionOpen(t)	-	*SendSessionOpen(t)*
UpdateParam(Id, param, t)	-	*GetParam(Id, param, t)*
-	-	*CheckSession(session, Id, t)*
ACKSW(t)	-	*SendACK(t)*

generation is modeled via an update `CreateToken()`, where the keyvault assigns the role, and its corresponding issue and expiry times. The token generation is shown in Listing 1. Since the token validation is an action that is not instantaneous, the model captures passing of time during invocation of this action. The flow of time is captured using the `kv_clk <= 1` invariant in conjunction with the `kv_clk >=1`. This expression ensures that the action uses one time unit by utilizing the internal clock of the keyvault process `clock kv_clk`. This clock is reset to zero at each invocation of the `kv_clk=0` update. In order to provide timestamps such as the expiry and issue times for tokens, tickets and sessions we introduce a time discretization process, reading the UPPAAL clock and providing integer discrete time. The time discretization process is shown in Figure 6.

```
bool ValidateCredential(){
    int i;
    for(i=0;i<users;i++){
        if(credentialRoles[i].Id ==
    credentials[credential_Id].Id ){
            if(credentialRoles[i].pass ==
    credentials[credential_Id].pass){
                return true;
            }
        }
    }
    return false;
}

void CreateToken(){
    tokens[credential_Id].Id =
    credentials[credential_Id].Id;
    tokens[credential_Id].issue_time = current_time;
    tokens[credential_Id].expiry_time = current_time +
    expiry_offset;
    tokens[credential_Id].role = getRole();
    token_issued = true;
}
```

Listing 1. Token generation for valid credential

Once the OT initiates switch access request, the keyvault validates the user token and determines if the access shall be granted for the selected switch. The token validation is shown in Listing 2. In case that the token is valid the ticket is issued and transferred to the OT, where the user could utilize it within a specific amount of time. Once utilized, the keyvault validates the ticket using the `ValidateTicket()` guard and based on the result hands over control to the switch.

```
bool ValidateToken(int sw){
    int tokenId = switchRequests[sw].tokenId;
    if(tokens[tokenId].expiry_time > current_time &&
    tokens[tokenId].role == admin){
        return true;
    }
        return false;
}
```

Listing 2. Token validation by the keyvault

In order to formally verify the security aspects of the protocol, we express the selected security properties in CTL. In this paper we mainly present the confidentiality, the authentication and the freshness property. The confidentiality property is expressed as `A[]ticketsHaveUniqueIdentities()` as shown in Listing 3, the authentication property is expressed as `A[](canGenerateTicket()==false) imply !kv.gen_ticket` with the detail shown in Listing 4 and the freshness property is expressed as `A<>(issuedTokensExpire() &&`

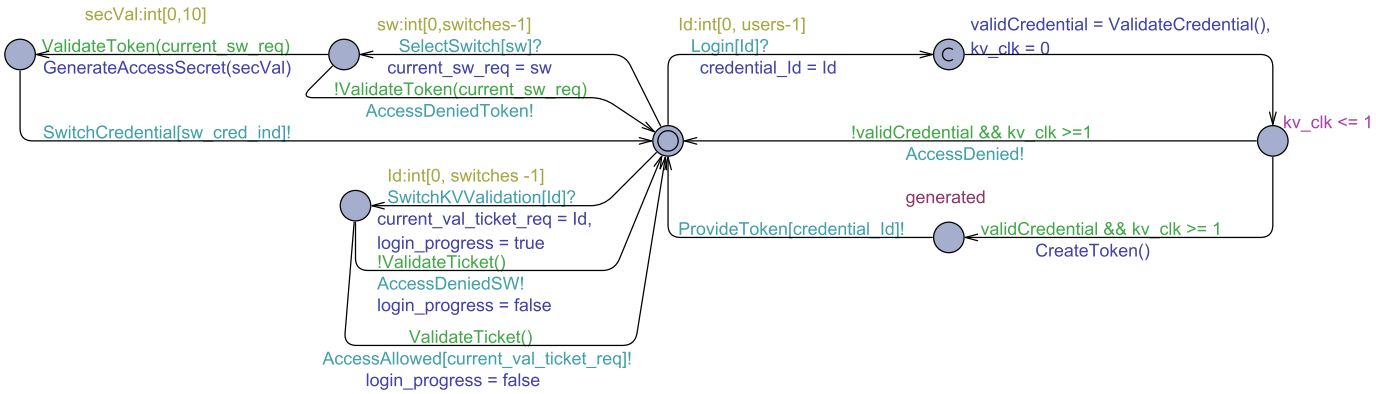

Fig. 3. UPPAAL template of the keyvault process

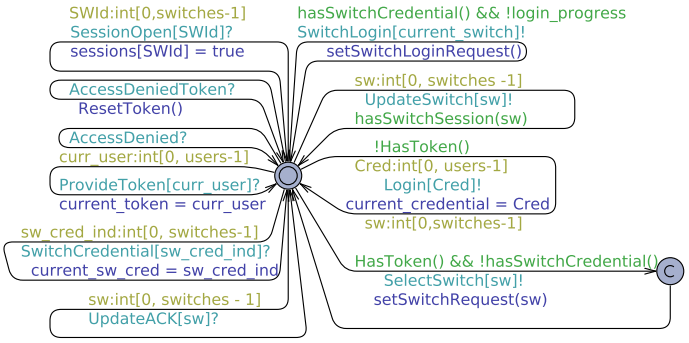

Fig. 4. UPPAAL template of the OT process

issuedTicketsExpire() && issuedSessionsExpire()), shown in Listing 5.

```
bool ticketsHaveUniqueIdentities(){
    int i;
    for(i = 0;i<switches;i++){
        int j;
        for(j = 1;j <switches;j++){
            if(switchCredentials[i].Id != -1){
                if(switchCredentials[i].Id ==
    switchCredentials[j].Id)
                    return false;
                }
            }
    }
    return true;}
```

Listing 3. The confidentiality property

```
bool canGenerateTicket(){
    int i;
    bool can = false;
    for(i=0;i<users;i++){
        if(tokens[i].role == admin &&
    tokens[i].issue_time <= current_time &&
    tokens[i].expiry_time >= current_time){
            can = can | true;
        }else{
            can = can | false;
        }
    }
    return can;
}
```

Listing 4. The authentication property

```
//Token freshness
bool issuedTokensExpire(){
    bool expiry = true;
    int i;
    for(i=0;i<users;i++){
        if(tokens[i].expiry_time>0){
            if (tokens[i].expiry_time <= current_time){
                //is expired
                expiry = expiry & true;
            }else{
                expiry = expiry & false;
            }
        }
    }
    return expiry;
}
//Ticket freshness
bool issuedTicketsExpire(){
    bool expiry = true;
    int i;
    for(i=0;i<switches;i++){
        if(switchCredentials[i].expiry_time>0 &&
    switchCredentials[i].expiry_time <= MAXTIME){
            if (switchCredentials[i].expiry_time <=
    current_time){
                expiry = expiry & true;
            }else{
                expiry = expiry & false;
            }
        }
    }
    return expiry;
```

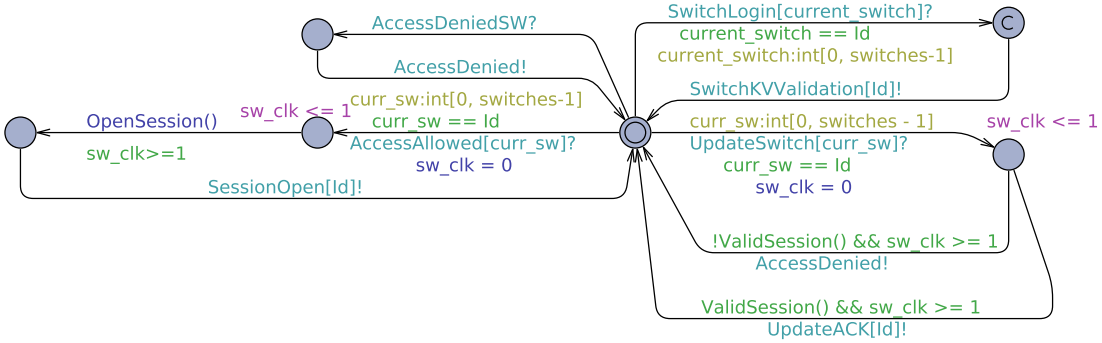

Fig. 5. UPPAAL template of the switch process

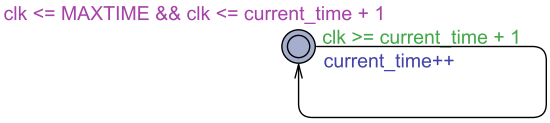

Fig. 6. Time discretization process

```
}
//Session freshness
bool issuedSessionsExpire(){
    bool expiry = true;
    int i;
    for(i=0;i<switches;i++){
        if(sessions[i].expiry_time>0){
            if (sessions[i].expiry_time <=
    current_time){
                expiry = expiry & true;
            }else{
                expiry = expiry & false;
            }
        }
    }
    return expiry;
}
```

Listing 5. The freshness property

F. Results and discussion

The analysis has been carried out on a desktop computer equipped with a 3GHz eight core CPU and 16 GB of DDR4 RAM running a 64bit version of UPPAAL 4.1.24 on a Linux operating system. The system considered for verification consisted of one OT, two switches, one keyvault and the time discretization process. The system has been instantiated with three credentials, one valid with administrative role, one valid without the administrative role and one invalid. This configuration has been selected based on initial requirements provided by our industrial partner, as the initial system that is expected to be deployed by the industrial partner will be of this minimalist nature, with expectations to add more nodes to the system such as an extra keyvault for redundancy purposes in the future. The maximum allowed execution time that the system could reach has been set to 16 time units. This has been done in order to limit the state space, hence the time expiration offsets for tokens, tickets and sessions have necessarily been set to shorter amount of time to be able to capture the intended scenarios. The setting of time expiration periods was such that the full run of the protocol could be carried out within the allotted time window. The verification of the confidentiality property has taken 57 seconds and consumed 1,4GB RAM, the verification of the authentication property has taken 65 seconds and consumed 1,6GB RAM and the verification of the freshness property has taken 6 seconds and consumed 0,1GB RAM. Verification of the rest of the properties has taken less than a minute with proportional memory consumption. The verification provided necessary answers for our industrial partner in terms of determining the selection of specific protocols for the system, in order to ensure that a specific combination of protocols could be used within the given scenario. This specifically answered the questions of feasibility of combining the two protocols and provided understanding of keyvault interaction is such environment. Based on the analysis the industrial partner has decide to create a prototype and utilize formal analysis further during the prototype implementation process.

VI. Related Work

Credential management and key management protocols are a critical functionality to maintain the security of modern control systems. Different studies have examined the robustness and reliability of such protocols to secure industrial control systems using formal methods [5], [8], [9], [15], [17], [19], [24].

Li et al. [15] extend the state of the art key management protocol MAKA with a prompt user revocation we propose a provable dynamic revocable three-factor and provide a formal security proof in the random oracle. However, the analysis does not explore the entire state space, thus no absolute guarantee about the reliability can be delivered. Kahya et al. [9] run a formal security analysis of the security sublayer of IEEE802.16 standard. Vulnerabilities with respect to DoS and Man-in-the-middle attacks have been identified, and the authors come up with mitigation processes following the

vulnerability traces. We aligned with this work when designing our models by using UPPALL to improve the ICS security.

Rocchetto et al. [24] introduced a formal framework to examine the security of cyber physical systems using a tool-assisted CL-AtSe verification. The system model including agents, communications and attack models are semantically translated to a formal transition for verification purposes. However, representing the system as a set of concurrent agents my lead to state space explosion.

Huang et al. [8] introduced a formal verification framework for security related timing constraints for communicating automotive systems. The system model is described using PrCcsl (probabilistic extension of clock constraint specification language) then translated to UPPAAL models with stochastic semantics. We align with this work, however we adopt deterministic model behavior.

Using a high-level description language to model control systems, as in [8], [24] leads to cheap modeling but complicates the verification process as such languages need to be translated to verifiable models.

Dojen et al. [5] presented a formal verification of a cluster based key management protocol for Wireless Sensor Networks. Mitigation processes have been proposed against identified attacks and improve the reliability of such a protocol. A formal verification of the updated protocol, incorporating the mitigation strategies, against such threats has been conducted. We model a dynamic credential management protocol and analyze its robustness against security properties elicited during a risk assessment carried out with our industrial partner.

VII. Conclusion

This paper introduced a formal setup to model and verify several security properties of a credential management for industrial control systems. The considered system architecture has been modeled based on an actual case study, provided by our industrial partner. The credential management protocol adopted and verified in this paper is based upon two well known authentication and authorization protocols, OAuth2 and RADIUS. We have used UPPAAL timed automata to model the behavior of the credential management protocol and used symbolic model checking to verify security properties for different aspects of the protocol. We demonstrate that the formal verification could be used practically with short verification time, while providing input for the future system design. Furthermore the industrial partner has decided to utilize formal modeling and analysis when considering extensions to their system.

As a future work, we plan to combine the presented keyvault system with works on a cryptographic key management system, in order to propose a formally verified subsystem capable of both key and credential management in an industrial control system setting.

Acknowledgments

This work is supported by the Manufacturing Academy of Denmark (MADE). For more information see http://www.made.dk/.

References

[1] I. S. Association et al. P1815 – ieee standard for electric power systems communications-distributed network protocol(dnp3). https://standards.ieee.org/standard/1815-2012.html, 2012.

[2] J.-P. Bodeveix, A. Boudjadar, and M. Filali. An Alternative Definition for Timed Automata Composition. In T. Bultan and P.-A. Hsiung, editors, *Automated Technology for Verification and Analysis*, pages 105–119, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] A. Boudjadar, J. H. Kim, K. G. Larsen, and U. Nyman. Compositional schedulability analysis of an avionics system using UPPAAL. In *Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering, ICAASE 2014*, pages 140 – 147, 2014.

[4] G. Brown and L.-P. Lamoureux. Mqtt and the nist cybersecurity framework version 1.0. https://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html, 2014.

[5] R. Dojen, F. Zhang, and T. Coffey. On the formal verification of a cluster based key management protocol for wireless sensor networks. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 499–506, 2008.

[6] J. Hassell. *Radius*. O'Reilly & Associates, Inc., USA, 2002.

[7] J. Hill. An analysis of the radius authentication protocol. Technical report, 2001.

[8] L. Huang and E.-Y. Kang. Formal verification of safety & security related timing constraints for a cooperative automotive system. In R. Hähnle and W. van der Aalst, editors, *Fundamental Approaches to Software Engineering*, pages 210–227, Cham, 2019. Springer International Publishing.

[9] N. Kahya, N. Ghoualmi, and P. Lafourcade. Key management protocol in wimax revisited. In D. C. Wyld, J. Zizka, and D. Nagamalai, editors, *Advances in Computer Science, Engineering & Applications*, pages 853–862, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] S. Karnouskos and A. W. Colombo. Architecting the next generation of service-based scada/dcs system of systems. In *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, pages 359–364, Nov 2011.

[11] W. Knowles, D. Prince, D. Hutchison, J. F. P. Disso, and K. Jones. A survey of cyber security management in industrial control systems. *International Journal of Critical Infrastructure Protection*, 9:52 – 80, 2015.

[12] T. Kulik, J. Boudjadar, and D. F. Aranha. *Towards Formally Verified Key Management for Industrial Control Systems*, page 119–129. Association for Computing Machinery, New York, NY, USA, 2020.

[13] Kulik, Tomas and Boudjadar, Jalil and Aranha, Diego F. UPPALL model of credential management subsystem for ICSs. https://tinyurl.com/ycwt7pcj, 2020.

[14] K. G. "Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, Dec 1997.

[15] W. Li, L. Xuelian, J. Gao, and H. Y. Wang. Design of secure authenticated key management protocol for cloud computing environments. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[16] M. Malekzadeh, A. A. A. Ghani, J. Desa, and S. Subramaniam. Vulnerability analysis of extensible authentication protocol (eap) dos attack over wireless networks. *ICGST International Journal on Computer Network and Internet Research CNIR*, 9(1):39–46, 2009.

[17] C. Meadows. Applying formal methods to the analysis of a key management protocol. *J. Comput. Secur.*, 1(1):5–35, Jan. 1992.

[18] P. J. "Mosterman and J. Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(1):5–16, Feb 2016.

[19] S. Naoui, M. E. Elhdhili, and L. A. Saidane. Security analysis of existing iot key management protocols. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–7, 2016.

[20] A. L. M. Neto, A. L. F. Souza, I. Cunha, M. Nogueira, I. O. Nunes, L. Cotta, N. Gentille, A. A. F. Loureiro, D. F. Aranha, H. K. Patil, and et al. Aot: Authentication and access control for the entire iot device life-cycle. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, page 1–15, New York, NY, USA, 2016. Association for Computing Machinery.

[21] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh. Formal verification of oauth 2.0 using alloy framework. In *2011 International Conference*

on Communication Systems and Network Technologies, pages 655–659, 2011.

[22] A. Parecki. *OAuth 2.0 Simplified*. Lulu.com, 2017.

[23] V. Ravi, N. R. Sunitha, R. Pradeep, and S. Verma. Formal methods to verify authentication in tacacs+ protocol. In *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT)*, pages 1–4, 2017.

[24] M. Rocchetto and N. O. Tippenhauer. Towards formal security analysis of industrial control systems. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 114–126, New York, NY, USA, 2017. Association for Computing Machinery.

[25] S. R. Tate and R. Vishwanathan. Expiration and revocation of keys for attribute-based signatures. In P. Samarati, editor, *Data and Applications Security and Privacy XXIX*, pages 153–169, Cham, 2015. Springer International Publishing.

[26] S. Winter, M. McCauley, and S. Venaas. Radsec version 2-a secure and reliable transport for the radius protocol, 2008.