IoTSEAR: A System for Enforcing Access Control Rules with the IoT

Andreas Put

imec-DistriNet, KU Leuven
Leuven, Belgium
andreas.put@kuleuven.be

Bart De Decker

imec-DistriNet, KU Leuven
Leuven, Belgium
bart.dedecker@kuleuven.be

***Abstract*—Internet of Things (IoT) environments are composed of heterogeneous sensors and devices that collect and share contextual information. This data can improve the accuracy and usability of access control systems, as authentication and authorization requirements can be specified more precisely. However, certain security requirements need to be enforced in order to use such data in access control decision processes. In short, the data must be authentic, recent, and unforgeable. In this paper, we present a generic model for context, which takes *data-security* into account along with properties about the device, or context-source. Security-objects, such as message signatures, are modeled as *proofs*, which are verifiable, while information about the context-source, communication channel, and the data itself is captured as *meta-data*. This model allows an access control system to verify the authenticity and trustworthiness of context-data by (1) checking the presence of a specific proof and verifying it, and (2) analyzing the associated meta-data. It covers not only data from IoT sources, but also authorization and identity tokens. In addition, we present IoTSEAR, a middleware for trustworthy context-aware access control, which uses this model internally. Finally, we show performance results of our IoTSEAR prototype, which show that the overhead is low and that the system is usable even on commodity hardware.**

Keywords–Access Control, Security, Internet of Things

I. INTRODUCTION

The rapid advancement of computing technologies has led to the paradigm shift from static device configurations to dynamic ubiquitous environments. Such a shift brings with it opportunities and challenges. On the one hand, users demand access to software services or information resources in an anytime, anywhere fashion. On the other hand, access to such services or resources needs to be carefully controlled, due to the additional security challenges and threats coming with dynamically changing environments. Consider a home-care setting in which IoT technologies enable the elderly and patients recovering from invasive treatments to stay in their own home instead of a healthcare facility. In such an environment, automation capabilities can facilitate the homeowner's day-to-day activities, while caregivers provide routine care to the inhabitant. The home is equipped with a smart lock, which automatically opens to the caregiver if (1) the patient is present in the home, (2) the health care provider authenticates the caregiver when she scans her NFC badge, and (3) the visit was scheduled. The home is equipped with an access control server, which accesses patient's presence status through a presence detector. Furthermore, the healthcare facility operates a federated access control service through which an (authenticated) identity is obtained using the output from an NFC terminal, which is integrated in the smart lock. Finally this identity is used to verify whether the visit is scheduled.

The access controller is required to combine different types of contextual information, whose properties and origin are completely different. The calendar service could authenticate itself with an SSL certificate, and the information it provides can be signed, while the presence detector sends its information over a Bluetooth channel without additional security controls.

Using context information empowers access control systems with extra capabilities and flexibility. However, it also opens up new attack vectors. Therefore, the following issues have to be addressed: (1) identifying the context information and associated context sources that satisfy a set of security requirements for it to be used in the access control decision process; (2) defining policies to specify context-aware access permissions; (3) enforcing these access control policies.

In order to address the first issue, we have developed a generic model for context, taking into account the device, or context-source, that produces the context information and the environment it was collected in. In addition, abstractions in this model encapsulate both context generated from IoT environments, and by (third party) access control systems, such as authorization tokens and identity assertions. System designers are able to translate security requirements to a set of conditions on properties of this model. Examples of such requirements are: the context must originate from a trusted device (authenticity + integrity), the connection must be end-to-end secured, or the context must be explicitly linked to a specific person.

The second issue defines the requirement for a context-aware access-control policy language. IoTSEAR supports the PACCo policy language [1] by default. However, it can be extended so support other policy languages as well.

To address the third issue, we propose IoTSEAR, a context-aware access control middleware designed for IoT applications. The IoTSEAR middleware framework is designed with the same design principles in mind as the Priman framework [2]. Priman provides application developers with secure and privacy-friendly authentication mechanisms in a developer-friendly manner. It offers a generic, easy to use API with simple, intuitive concepts by isolating the security- and technology-specific details into configuration policies. This *separation of concerns* between application developers and security experts furthermore

 ISBN: 978-1-61208-796-2

increases the manageability of systems, as service providers are able to modify authentication mechanisms at run-time, without requiring application code modifications. As IoTSEAR's design follows these principles, its implementation should result in a usable (from a developer's point of view), configurable (from a service provider's point of view), and extensible middleware. Moreover, while Priman is an authentication framework, IoTSEAR is a general access control middleware. It includes support for context-aware authorization, distributed authorization schemes and authentication schemes.

This work presents two main contributions:

- A generic model for context in an IoT-based access control setting. This model also encapsulates information related to data-security and data-origin, with which a third party is able to verify the authenticity and trustworthiness of the context information.
- The architecture, prototype implementation, and benchmarks of IoTSEAR, a system for enforcing access control rules in IoT environments.

This paper is structured as follows: Section II contains an overview of the related work, after which the generic model for context is explained in Section III. Furthermore, Section IV details the IoTSEAR middleware, which is discussed and evaluated in Section V.

II. Related Work

Automation is a central goal in many IoT ecosystems [3]. Several existing solutions [4] rely on cloud infrastructure to specify and enforce policies. Another cloud-based approach that defines intents and scopes on which these intents will have an effect is described in [5]. Besides academic initiatives, commercial solutions, like Google Cloud IoT [6], Home Assistant [7] and OpenRemote [8] offer intuitive cloud-based support to create automation rules. Similar to access control systems, a set of policy rules consisting of a set of conditions that must be fulfilled to trigger one or more actions are specified and enforced. However, the heterogeneity of IoT devices and the fact that many devices have low capabilities open new attack vectors [9]–[11]. An extensive access control framework to manage access to devices, however, is still missing [12].

Attribute-based access control (ABAC) [13] is a general purpose access control model which allows access rights to be constrained based on the attributes of subjects, objects, actions and the environment. In an ABAC policy, a logical expression consisting of attribute information is defined as a conditional rule. The applicability of such a policy to a request is determined by matching the attributes in the request and the environment to the attributes in the policy. The application of ABAC to the IoT has seen much interest in the last decade [14]. However, the potentially large amount of attributes required to establish dynamic policies is a challenge.

Capability-based access control (CapBAC) [15] defines a capability as a self contained key or token, that references a target object or resource along with an associated set of access rights. This allows for fine-grained, flexible access control, as holding such a token access to only those resources that are necessary for the holder's legitimate purpose. CapBAC has seen much interest in the IoT-sphere [16]–[19]. However, to our knowledge, no system exists that combines support for (1) attributes, capabilities and context in authorization, (2) dynamic verification controls for the used context, and (3) support for authentication into a complete access control middleware. IoTSEAR accomplishes this by building on previous work [1], [2], and with our context model (Section III), as all objects are internally handled as generic context structures.

PACCo [1] is a system that focuses on the secure and privacy-friendly collection of contextual information, after which capability tokens can be issued. Furthermore, a protocol is proposed in which *personal verifiable context* can be verified in a privacy-friendly, unlinkable, manner. This type of context allows a third party to cryptographically verify the authenticity and ownership of certain contextual information (i.e. the information is authentic and it belongs to the subject that makes the access request). The PACCo policy language, is a context-aware policy language which focuses on expressing rich context-based requirements and also considers the security requirements that appropriate context sources must adhere to. This policy language is used as the default policy language for the IoTSEAR implementation, in large part due to its capability to express such security constraints. However, the IoTSEAR architecture allows to support other policy languages as well.

III. A generic model for contextual information

In order to uniformly reason about different types of contextual information and their security properties, a generic model for context in access control is proposed in this section.

A. Context in access control

Abowd et al. [20] specifies a broad definition of *context*:

> "Any information that can be used to characterize the situation of an entity. This entity is a person, place, or object considered relevant to the interaction between a user and an application, including the user and applications themselves."

IoT environments produce a wide variety of information that is useful to take into account when making access control decisions. Some examples are: the current location of a subject, the proximity of a subject to a sensor or even to a specific person, the current time, etc. For access control systems, three context origins are clearly distinguishable:

a) IoT Device: The situation of an entity or environment is measured by, and accessible through IoT devices. For example, networked sensors and smart devices.

b) System state: The access controller's internal state is an important source of context, such as the current session information, connection type, internal database and clock.

c) Third party/Cloud Service: The information about a particular entity can be provided by a third party, such as a cloud service or a database. This information is often signed by the provider, and/or it is obtained through a secure, authenticated connection. Certificates, identity and authorization tokens are produced by these sources.

When access control systems consider not only *system state* information, but also information originating from *sensors* and *third party services*, they can enforce more context-rich policies. However, the context information that does not originate from the system itself should not be treated as trustworthy, but as *potentially faulty or even dangerous*. Indeed, the external context source (i.e. the sensor or third party) can

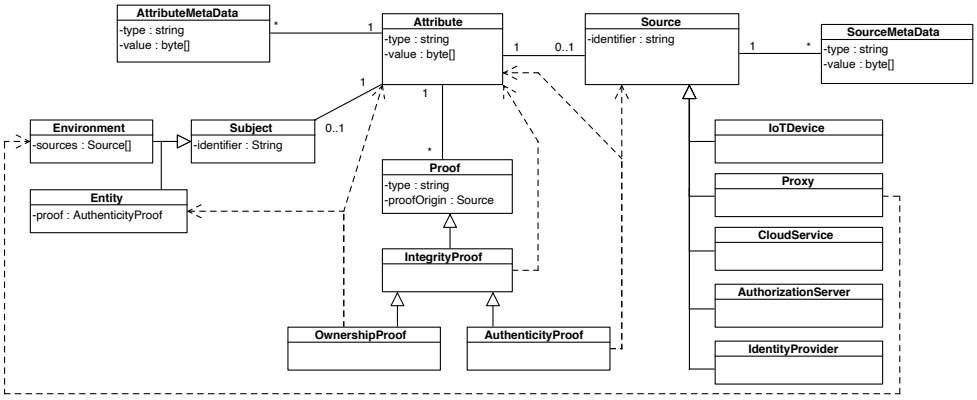

Figure 1. Model for context: the core element is *Attribute*, while *Source*, and *Proof* complete the set of the three most important elements.

be compromised, spoofed, or the context information could be forged or replayed. Therefore, it is essential that *context is validated before it is used in critical application functions*. Such validation strategies can range from simple input validation and error correction to integrity and authenticity validation, depending on the use case.

B. Context Model

In order to uniformly reason about different types of contextual information, their security requirements and the validation thereof, a generic model for contextual information is proposed. This model defines the concept of "contextual information" as an *Attribute*. This element has a variety of (optional) associated elements, that are dependent on the manner of context collection, or the environment in which the context is collected. Different validation strategies are possible depending on the availability of certain optional elements. The generic model for context is illustrated in Fig. 1.

Attribute: The core of the model is the *Attribute* element, which has a `value` and an `type` field. The *value* represents the raw output from the attribute's *Source*. This can range from a sensor-reading to an identity assertion or an authorization token. The *type* determines the actual type of the attribute, which also implies the encoding. The Attribute is the model's core, all other elements are optional. Attributes can have associated *AttributeMetaData* elements. This element also has a `type` and `value` fields. Examples of AttributeMetaData are: the time when the attribute is collected, the accuracy of a sensor reading, or other information related to the Attribute.

The Attribute is related to a *Subject*, which is either an *Entity*, or an *Environment*. Each distinct subject has a unique `identifier`. An *Environment* is characterized by the set of `sources` that are active in this particular environment. *Entities* represent people (or their personal device). An Entity contains a `proof` field, which is used to verify the authenticy of the Subject's identifier (see *Proof* paragraph below).

Source: A *Source* is either an *IoTDevice* (sensor, actuator, smart device), but it can also be a *Proxy*, *CloudService AuthorizationServer*, or an *IdentityProvider*. A *Proxy* acts similar to network proxy for a set of IoT devices (i.e. an *Environment*). Every source is uniquely identified by its `identifier` field. Similar to the Attribute element, the Source can have associated *SourceMetaData*. Examples are: the source's owner, operator, location, software version, device attestation status, etc.

Proof: The model supports three *Proof* kinds: *OwnershipProof*, *AuthenticityProof* and *IntegrityProof*. Each proof has a `type` field, which has a similar function to the attribute's type. A *Proof* has a `proofOrigin` field, which verifiably links it to its source. In addition, *Proxies* are able to add proofs and meta-data to the context structures that it relays. However, this depends on the scenario and system configuration.

Attributes that are used to establish an authenticated identity (e.g. [id='Alice', location='room123']) must contain an *OwnershipProof*, which links an *Attribute* to a specific *Entity* in a verifiable manner. For example, proving ownership of a certificate (and the attributes it contains) is done by proving knowledge of the certificate's private key, i.e. by signing a nonce. Note that a protocol for capable IoT devices (e.g., smartphone, proxy device) to create ownership proofs of contextual information is detailed in [1]. An *AuthenticityProof* allows to verify whether a specific source produced an attribute (e.g., a message signature). Finally, the *IntegrityProof* shows that the attribute value has not been modified or corrupted. Note that both Ownership- and AuthenticityProof are IntegrityProofs.

C. Instances of the context-model

To illustrate the flexibility of the generic model for context, it is applied to two distinct context types, which are extracted from the scenario illustrated in the introduction: (1) a simple Bluetooth beacon reading (presence context), and (2) a SAML [21] identity assertion (authenticated identity).

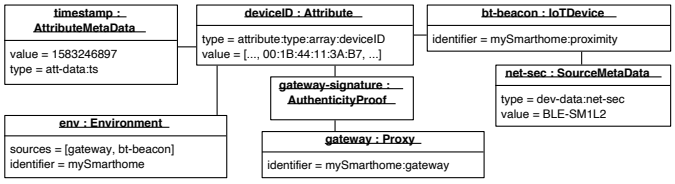

Figure 2. The context model applied to a proximity sensor reading

Fig. 2 shows the modeled context from a proximity sensor (e.g., a Bluetooth beacon). All types and identifiers, and their implications (value-encoding, device configuration, supported Attribute- and Source-MetaData types) are known to the system. In this example, the `deviceID` attribute has a `timestamp` as meta data, while the associated source

(bt-beacon) has associated meta data detailing the wireless network security properties: *BLE-SM1L2*, or Bluetooth Low Energy, security mode 1 level 2, meaning 'unauthenticated pairing with encryption'. The environment in which the attribute is collected (env) is a room consisting of bt-beacon and gateway, a device which acts as a proxy to bt-beacon.

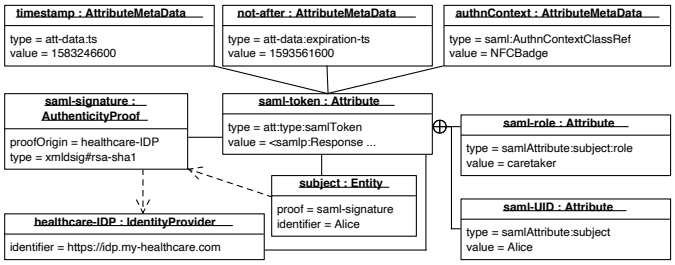

Figure 3. The context model applied to a SAML identity assertion

Fig. 3 shows the modeled context from a SAML response, received from a third party identity provider (healthcare-IDP). The value of the main attribute, saml-token, is the full content of the SAML response. This response asserts the values of two identity-attributes: 'subject=Alice' and 'role=caretaker'. The meta data attributes (timestamp, not-after, authnContext) are extracted from the raw SAML response, as is the saml-signature, which is modeled as an *AuthenticityProof*.

IV. IoTSEAR

The IoTSEAR middleware has two main responsibilities: managing contextual information and enforcing context aware access control rules.

A. Context management

Gathering context information is done by accessing IoT sensors, cloud services, and processing identity- and authorization-tokens. When IoTSEAR receives context data, it creates a *context structure* that conforms to the context model using the received data itself and known information about the environment, its devices and network properties. Fig. 4 illustrates three different ways of context collection. The middleware can process third party objects from a cloud service, identity provider (IDP) or authorization server (AS). Furthermore, part of the middleware can run on capable IoT devices, such as a smartphone. The IoTSEAR software running on these devices will read the sensor data and construct a context structure conforming to the model. Depending on the configuration, verification proofs are added to this structure. Proxy devices function similarly: they access the sensor readings, create the context structure, and (optionally) add a verification proof. Additionally, devices running the full middleware can act as a Proxy in addition to Identity Provider and Authorization Server.

Context DB is the database containing all context structures. Note that most types of contextual information have a limited lifetime. Hence, this database is regularly pruned of old context structures. The second database, *Environment DB*, contains information from which the SourceMetaData is constructed. Moreover, this database is initialized with all required verification objects: certificates, shared keys, and trust-relationships.

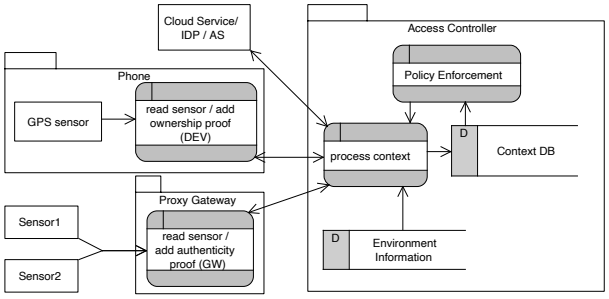

Figure 4. Context processing for three distinct sources.

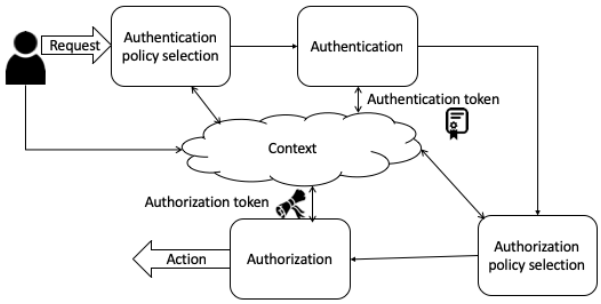

Figure 5. Context aware authentication and authorization

During *Policy Enforcement*, the middleware analyses the set of applicable policies. The necessary context structures are retrieved from *Context DB*. Absent context structures can be created at run-time by accessing the appropriate sources, after which all information is known to enforce the policies.

B. Context aware Authentication and Authorization

The typical IoTSEAR transaction is divided in four stages (see Fig. 5). The system has access to a set of context structures, represented in the figure by the 'context cloud'. The first stage starts when the system receives a request from a user. An appropriate authentication policy is selected based on the user's claimed identity. Here, the system can determine that stronger authentication is appropriate (e.g., two factor authentication), or that a more relaxed authentication is suitable, using predefined rules specified in policies. Note that the request itself also produces contextual information (i.e. session info), which can be used in the next steps, or subsequent transactions.

Biometric context and identity objects from third parties can be involved in the authentication phase. Afterwards, the system will produce a new *Authentication token*. These two steps can be skipped in case the user already authenticated, or authentication/identification is not necessary.

The authorization policy selection occurs analogous to the authentication policy selection. After the desired policies have been selected, their conditions are evaluated. The actions associated with those policies whose condition is satisfied are allowed or denied, according to the *policy effect*. Finally, when approved, an authorization token is created.

C. Architecture

The high-level architecture of the access control middleware is shown in Fig. 6. Applications interact with the middleware

 ISBN: 978-1-61208-796-2

by inserting a *policy enforcement point* (PEP) in their code. This is a code block in which a a policy decision request is sent to the middleware, or more specifically, to a *policy decision point* (PDP). The PDP can be located on a different device or network. The request contains information, such as the type of event (e.g., an access request, an authentication request, etc.) and the (claimed) identity of the subject. The high-level architecture of IoTSEAR is divided in three main components:

a) Abstraction layer: In this layer, the abstractions and APIs used by developers are defined. In addition, the mechanisms with which the plugins are loaded, initialized and configured are implemented in this layer.

b) Plugin Layer: This layer encapsulates the plugins and extensions that are available. These plugins contain the actual functionality implementations of the IoTSEAR middleware.

c) Middleware Configuration: This component contains the environment-specific configurations, and the middleware configuration settings. Policy mappings enable the middleware to load the correct plug-ins with identifiers found in the policies.

The main APIs for application developers are located in the *abstraction layer*. This layer defines the interfaces to which each plugin must comply. The *Context Model* component is also located in this layer. Next, the API, abstractions and plug-in managers for a *PDP, Policy Engine and Policy Repository* are defined in this layer. These three components are responsible for policy selection and authorization. The *Authentication Engine*, on the other hand, is responsible for loading and executing the correct authentication plugin.

The *Plugin-layer* contains the plugins that are active in the system. The *Crypto primitives* component contains implementations for cryptographic operations, such as the different encryption, hashing and signature algorithms that are required to perform authentication, and to verify context structure. The *PDP implemenations* component contains, as the name suggests, implementations for different PDPs. Different PDPs (e.g., LocalPDP, TlsPDP, ...") support different communication methods (e.g., through a local or a TLS socket). The *Policy Rules* component contains the implementation of every condition and matching function in the policy language (see [1]). Hence, extending the policy language can be accomplished by defining a new plugin. Using the same strategy, it is also possible to support a completely different policy language. The *authentication protocols* component contains the extensions that execute a specific authentication protocol. This component, together with the *Authentication Engine* originates from the Priman Framework [2] (with minor adaptations).

V. Discussion and Practical Evaluation

This section entails a discussion on how contextual security requirements are enforced by IoTSEAR in addition to a performance analysis of the middleware.

A. Enforcing security requirements

In order to enforce security requirements on context used in access control decisions, the PACCo policy language [1] allows policy conditions to be labeled with a *security attribute*. This security attribute corresponds to a set of security requirements to which the used context must comply.

One possible security attribute configuration is inspired by the Eurosmart Security Assurance levels [22]:

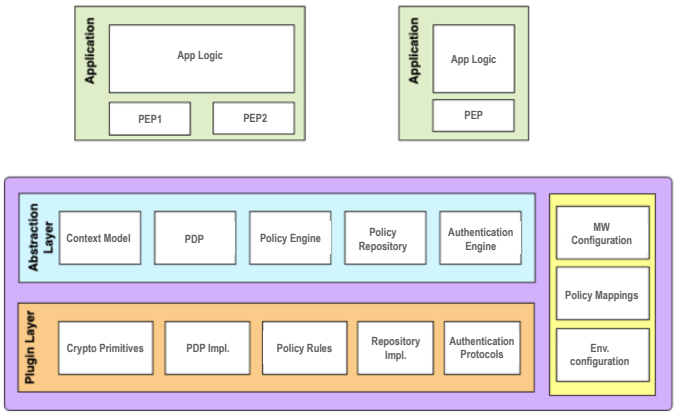

Figure 6. IoTSEAR Architecture

Basic: Minimize the basic risks of incidents.
Substantial: Minimize the known risks, and the risk of incidents carried out by actors with limited skills and resources.
High: Minimize the risk of state-of-the-art attacks carried out by actors with significant skills and resources.

These descriptions must be translated to a set of requirements on properties of a *context structure*. For example, *basic* can require that the context *Source* is trusted, i.e. the `Source identifier` is in a list of trusted sources

The level *substantial* can require the usage of a secure network, and an authenticity proof in addition to the *basic* requirements. Network information can be accessed through the *SourceMetaData*. A whitelist of allowed network types must be known, analogous to the list of trusted device identifiers. This is also the case for the whitelist of allowed proof types.

The level *high* can require that the software of the context source has no known bugs, that device attestation has been recently performed, in addition to the *substantial* requirements. The new requirements, however, require the used whitelists to be regularly updated.

The IoTSEAR middleware allows the definition and enforcement of custom security requirements. This requires a new plug-in to be created for each custom security attribute. Every plug-in implements a single method, which has one parameter, the *context structure*, and returns a boolean. Note that all elements in the model are accessible through the context structure. The security-attribute plugins are loaded based on the name of the security attribute in a policy, and checked by calling the single implemented method provided with the appropriate context structure as argument.

B. Practical evaluation

The IoTSEAR prototype is implemented in Java, and was tested on a machine with a 2,3 GHz Intel Core i5 processor and 16GB RAM. All test have been performed 100 times, and the (in memory) *ContextDB* contains 100 000 context structures. We show the average values in milliseconds (standard deviation is noted between parentheses).

The first test measures the processing of a context structure. Consider a worst case scenario, in which a SAML token is processed. Our test token contained 7KB of XML data, which is parsed and verified (SHA265 with RSA2048). Next, the context

 ISBN: 978-1-61208-796-2

TABLE I. PERFORMANCE RESULTS BASED ON THE AMOUNT OF POLICIES AND VERIFIED CONTEXT STRUCTURES (CS).

# policies / CS	Abs. Layer	CS Verif.	Evaluation	**Total**
10 / 1	1 (0)	16 (3)	4 (0)	**21**
40 / 1	1 (0)	67 (5)	7 (0)	**75**
100 / 1	2 (1)	180 (9)	11 (0)	**193**
10 / 10	2 (0)	198 (11)	16 (1)	**216**
40 / 10	4 (2)	760 (32)	54 (3)	**818**
100 / 10	7 (3)	1902 (93)	143 (9)	**2052**

structure is created and serialized using Google's protocol buffer [23]. This whole process takes 34 ms (6).

Table I shows the evaluation times for sets of simple and complex policies. The first policies that are evaluated are simple and pose two constraints: "subject=randomID" and "role=manager", which are evaluated using one SAML context structure. Next, complex policies are evaluated, which have additional constraints and require 10 context structures to evaluate. For both the simple and complex policies, the used context must adhere to the *high security level*. Using the example from Section V-A, this requires (for each CS) four SourceMetaData values to be matched to a whitelist, timestamp verification, and the verification of one AuthenticityProof (SHA265 with RSA2048 signature). The overhead introduced by the *Abstraction Layer* is minimal, while the verification of the *context structures (CS)* requires the most time. This is mainly due to the signature verification, which occurs 1000 times (100 x 10 structure) in the heaviest test. The amount of context structures to verify (and their verification requirements) have a large impact, but optimizations such as multi-threaded or ahead-of-time CS verification are possible.

Consider the scenario from the introduction, where a caregiver is given access to a patient's home if (1) the patient is present, (2) the health care provider authenticated the caregiver when she scans her badge, and (3) the visit was scheduled. Only one policy is evaluated, as it can be uniquely targeted by the access request (subject=*caregiver-id*, action=*door:open*). The healthcare-IDP is consulted at run-time, and has a response-time of 1 second. Taking this into account, it takes 1052 ms (4) to make this access control decision (of which 1021 is used to request and verify the context from the healthcare-IDP).

VI. CONCLUSIONS

This paper presented a generic model for context and described IoTSEAR, a middleware for context-aware access control. IoTSEAR uses the current context to select the most appropriate authentication and authorization policies. In doing so, the dynamic adaptation of the access control mechanism is facilitated. Furthermore, the security requirements of the used context can be precisely tailored to any given application, and are automatically enforced by the middleware. Finally, our performance tests showed that the overhead is limited, and that the middleware is suitable for commodity hardware.

REFERENCES

[1] A. Put and B. De Decker, "Attribute-based privacy-friendly access control with context," in International Conference on E-Business and Telecommunications. Springer, 2016, pp. 291–315.

[2] A. Put, I. Dacosta, M. Milutinovic, and B. De Decker, "Priman: facilitating the development of secure and privacy-preserving applications," in IFIP International Information Security Conference. Springer, 2014, pp. 403–416.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," IEEE communications surveys & tutorials, vol. 16, no. 1, 2013, pp. 414–454.

[4] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, "A survey of commercial frameworks for the internet of things," in 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA). IEEE, 2015, pp. 1–8.

[5] S. Nastic, S. Sehic, M. Vögler, H.-L. Truong, and S. Dustdar, "Patricia–a novel programming model for iot applications on cloud platforms," in 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications. IEEE, 2013, pp. 53–60.

[6] Google cloud iot. Accessed on 06-10-2020. [Online]. Available: https://cloud.google.com/solutions/iot/

[7] Home assistant. Accessed on 06-10-2020. [Online]. Available: https://www.home-assistant.io/

[8] Openremote. Accessed on 06-10-2020. [Online]. Available: https://openremote.io/

[9] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," Computer, no. 9, 2011, pp. 51–58.

[10] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: perspectives and challenges," Wireless Networks, vol. 20, no. 8, 2014, pp. 2481–2501.

[11] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in Proceedings of the 11th ACM on Asia conference on computer and communications security. ACM, 2016, pp. 461–472.

[12] M. Hossain, R. Hasan, and A. Skjellum, "Securing the internet of things: A meta-study of challenges, approaches, and open problems," in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017, pp. 220–225.

[13] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in IEEE International Conference on Web Services (ICWS'05). IEEE, 2005.

[14] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the internet of things: Big challenges and new opportunities," Computer Networks, vol. 112, 2017, pp. 237–262.

[15] L. Gong et al., "A secure identity-based capability system." in IEEE symposium on security and privacy, 1989, pp. 56–63.

[16] F. Malamateniou, M. Themistocleous, A. Prentza, D. Papakonstantinou, and G. Vassilacopoulos, "A context-aware, capability-based, role-centric access control model for iomt," in International Conference on Wireless Mobile Communication and Healthcare. Springer, 2016, pp. 125–131.

[17] D. Hussein, E. Bertin, and V. Frey, "A community-driven access control approach in distributed iot environments," IEEE Communications Magazine, vol. 55, no. 3, 2017, pp. 146–153.

[18] D. Hussein, E. Bertin, and V. Frey, "A community-driven access control approach in distributed iot environments," IEEE Communications Magazine, vol. 55, no. 3, 2017, pp. 146–153.

[19] S. Pal, M. Hitchens, V. Varadharajan, and T. Rabehaja, "Policy-based access control for constrained healthcare resources in the context of the internet of things," Journal of Network and Computer Applications, vol. 139, 2019, pp. 57–74.

[20] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in International symposium on handheld and ubiquitous computing. Springer, 1999, pp. 304–307.

[21] S. Cantor, J. Kemp, N. R. Philpott, and E. Maler, "Assertions and protocols for the oasis security assertion markup language," OASIS Standard (March 2005), 2005, pp. 1–86.

[22] Eurosmart iot certification scheme. Accessed on 06-10-2020. [Online]. Available: https://www.eurosmart.com/eurosmart-iot-certification-scheme/

[23] Protocol buffers. Accessed on 06-10-2020. [Online]. Available: https://developers.google.com/protocol-buffers/

 ISBN: 978-1-61208-796-2