A Formal Model for Delegated Authorization of IoT Devices Using ACE-OAuth

LUCA ARNABOLDI, Newcastle University, UK
HANNES TSCHOFENIG, Arm Limited, UK

As our daily lives become ever more connected the need for security becomes more and more dire. This need is, however, in conflict with the way most IoT devices are designed because devices have limited computing power, memory and security mechanisms to defend against a range of attacks. The protocol designer must therefore take all these constraints in consideration when making design decisions on how to secure the IoT. To bring authentication and authorization solutions to constrained IoT devices the IETF Authentication and Authorization for Constrained Environments (ACE) working group was formed and profiled the OAuth protocol suite. Re-using already existing security protocols offers a lot of benefits since designing security protocols is not an easy task. Over the years several technologies for automated verification, such as Tamarin [6] and Easycrypt [1], have been developed. Given a protocol design and a set of security properties (e.g. secrecy and authentication) these tools can help to prove that the properties hold for any potential scenario, under a given threat model. The use of these tools has become popular as prominent Internet security protocols have been found vulnerable by researchers who used these tools.

In this paper we use the formal analysis tool Tamarin to analyse the ACE-OAuth protocol and we illustrate how protocol designers can make use of this work for analysing their own extensions.

Additional Key Words and Phrases: IETF, OAuth,ACE-OAuth,Protocol Verification, Security, Standardization

1 INTRODUCTION

Internet of Things device often have limited processing power, storage space, and transmission capacities. In many cases, these devices do not provide user interfaces, and they are often intended to interact without human intervention. Still, there is a need to provide authentication and authorization capabilities for such devices when they are connected to the Internet. The protocol to provide delegated access for web applications and web services today is the OAuth 2.0 protocol and with the standardization work in the IETF ACE working group engineers have applied the OAuth 2.0 framework to IoT devices, which is called ACE-OAuth.

In an paper submitted to the 3rd OAuth Security Workshop we analysed a small subset of the ACE-OAuth protocol (namely the Client Token Protocol) using Scyther and Avispa, two tools for automated verification. As a consequence of our findings, the Client Token Protocol was removed from the specification. In this paper we go a step further and apply a more modern tool, namely Tamarin, to the entire ACE-OAuth framework.

The main contributions are:

Formalisation of the ACE-OAuth Architecture: We extract security requirements and security goals from the specification and describe them formally. As explained in Section 4.1 this step was subject to lots of interpretation as the properties were not specified formally in the ACE-OAuth specification [7] or any of the accompanying documents (200 pages across three documents). In fact, it is rather uncommon to find any formally specified security goals in IETF specifications today.

Formal Model of a ACE-OAuth case study: After formalising the specification we developed a fully executable model of the ACE-OAuth protocol. To the best of our knowledge this is the first fully verifiable model that can be analysed automatically for this particular protocol. We modelled the protocol so that we can perform a

Authors' addresses: Luca Arnaboldi, Newcastle University, 1 Urban Science Square, Newcastle University, NE4 5TG, UK, l.arnaboldi@ncl.ac.uk; Hannes Tschofenig, Arm Limited, 110 Fullbourne Rd. Cambridge, UK, hannes.tschofenig@arm.com.

full security analysis to work around the standardised lemmas and requirements (for applicability and higher assurance of the proof).

Analysis and Results: We carry out a full formal security evaluation of the protocol as a whole and then specifically investigate one implementation. As the spec allows for several different deployment setups, we asked our colleagues to come up with a profile adhering to the ACE-OAuth implementation and carried out a comprehensive analysis.

We hope that by making the formal model of the ACE-OAuth framework available it can be used as the baseline for analysis of future protocol extensions. As is set out it should be possible if one wishes, to test out various different ACE-OAuth setups by altering the current model slightly, as exemplified by our case study. The intent is to use the model to make informed decisions about protocol changes and protocol extensions.

2 FORMAL PROTOCOL VERIFICATION OVERVIEW

With the increasing number of protocols purpose-built for IoT it becomes more difficult to analyse the security manually. Hence, it is desirable to offload the analysis to automated verification tools. The way these protocols are formally verified can be broken down into two main methods, namely using a symbolic [3] and a computational [8] model. These approaches are very different and have different objectives. The symbolic model is mainly used to verify protocol specifications as it is very high level and quicker to implement. The computational model is more precise and can analyse specific implementation details but is more time consuming.

In the symbolic model cryptographic primitives are represented by function symbols considered as black-boxes. The messages are terms on these primitives, and the adversary is restricted to compute only using a predefined set of strategies. This model assumes perfect cryptography so there is no notion of brute forcing a password (unless specifically introduced as a rule). One can add equations to model algebraic properties of the cryptographic primitives and the only equalities that hold are those explicitly given by these equations. The main adversary in this model is described as the Dolev-Yao adversary [3], and the Channel(or the medium of transfer) is the intruder. Hence, anything sent on this channel can be read, modified and ulteriorly the adversary can fabricate messages and replay all messages sent over the channel. The one limitation is that the adversary cannot encrypt/decrypt messages without a key. This overly simplistic way can capture errors in the logic of the design, but cannot fully describe situations in which the cryptographic primitives cannot be treated as black boxes and when the security properties are defined specifically.

Conversely, in the computational model the messages are bitstrings (and take a specific form). The cryptographic primitives are functions from bitstrings to bitstrings and unlike in the Dolev-Yao model, the adversary is any probabilistic Turing machine. A proof passes if the adversary can only compute the key (or break the requirement) with negligible probability. The length of keys is determined by a value named security parameter and the run-time of the adversary is supposed to be polynomial in the security parameter (complexity). A proof in this model is more rigorous and more closely related to the eventual implementation. This gives us more assurance of the protocols security and this is especially true when trying to catch edge cases. For example, a newly designed protocol still supports older cryptographic algorithms for legacy reasons.

Whilst dealing with the design of a protocol the symbolic model offers a lot more flexibility as changes to the protocol design can be verified more quickly. We have selected Tamarin as our automatic verification tool since it has been used successfully to analyze TLS 1.3.

2.1 Tamarin

The Tamarin prover is described as: "A powerful tool for the symbolic modelling and analysis of security protocols" [6]. It takes as input a security protocol model, specifying the actions taken by agents running the protocol in different roles (e.g. the protocol initiator, the responder, and the trusted key server), a specification of the adversary, and a specification of the protocol's desired properties. Tamarin provides general support for modelling and reasoning about security protocols. Protocols and adversaries are specified using an expressive language based on multiset rewriting rules. These rules define a labelled transition system whose state consists of a symbolic representation of the adversary's knowledge, the messages on the network, information about freshly generated values, and the protocol's state. The adversary and the protocol interact by updating network messages and generating new messages. Security properties are modelled as trace properties, checked against the traces of the transition system, or in terms of the observational equivalence of two transition systems. It is fully automated but one can still use an interactive view to guide the proof search. If the tool's automated proof search terminates, it returns either a proof of correctness (for an unbounded number of role instances and fresh values) or a counterexample, representing an attack that violates the stated property.

To model a protocol Tamarin makes use of a concept known as multiset rewriting rules. The rules operate on the system's state, which is expressed as a multiset (i.e. a bag) of facts. We explain the Tamarin syntax making use of a toy example:

```
rule ExampleRule:
    [Fr(~UUID)]--[DeviceRecorded($X,~UUID)]->[!RegisterDevice($X,~UUID)]
```

Tamarin rules consist of a left-hand side, actions and a right-hand side. Within these rules the basic ingredients are:

Terms: Terms are variables in the system of the following types ~x denotes ~x:fresh (like a nonce), $x denotes x:pub (publicly available value such as a device IP), and m denotes m:msg (untyped message);

Facts: Facts are ways in which variables are derived, rules use facts to manipulate terms in the system;

Actions: Actions are unique and differ to the other two ingredients as they do not alter the state of the protocol. They are used as annotations of what is happening in the rule and are used exclusively in the property specification.

A rule can only be executed if all the facts on its left-hand side are available in the current state. When a rule is executed, it will consume the facts on the left , i.e. removing them from the global state, and produce facts on the right, i.e. adding them to the global state. In the running example we see that fact `!RegisterDevice(C,~UUID)` . When called this fact takes a public value C representing the client and relates it to a unique value representing the unique device id. If the fact has a "!" in front of it this means it can be consumed multiple times and is persistent in the global state.

3 ACE-OAUTH OVERVIEW

Since the purpose of this paper is to formally analyse the ACE-OAuth protocol we need to briefly describe how it works first. The full details can be found in [7]. The high-level architecture of ACE-OAuth is shown in Figure 1 where the OAuth Client interacts with an Authorization Server to obtain an access token. Once the OAuth Client is authenticated and authorized (potentially with the consent of the resource owner or some third party), it will receive an access token for use with a Resource Server. The Resource Server hosts the protected resource that the OAuth Client is interested in accessing. Unlike a classical OAuth where bearer tokens are used today, an access token used with ACE-OAuth is a proof-of-possession token, which has a key bound to it. When the OAuth Client presents the access token to the Resource Server it also needs to demonstrate possession of the key associated with the access token.

ACE-OAuth was designed for use in Internet of Things (IoT) environments. As the name indicates, it uses the OAuth framework. In order to make OAuth suitable for use in IoT, it may be necessary to use more lightweight protocols. For example, CoAP may be used instead of HTTP, DTLS instead of TLS, CBOR instead of JSON, and COSE instead of JOSE. These protocol and encoding alternatives ensure that the payloads transmitted over the air are significantly reduced in size. While many encoding details are not necessarily relevant for a formal analysis, they are useful for understanding the goal of the standardization activity in the IETF ACE working group. There are, however, subtle implications from switching from one protocol to another since protocols often have slightly different semantic and properties.

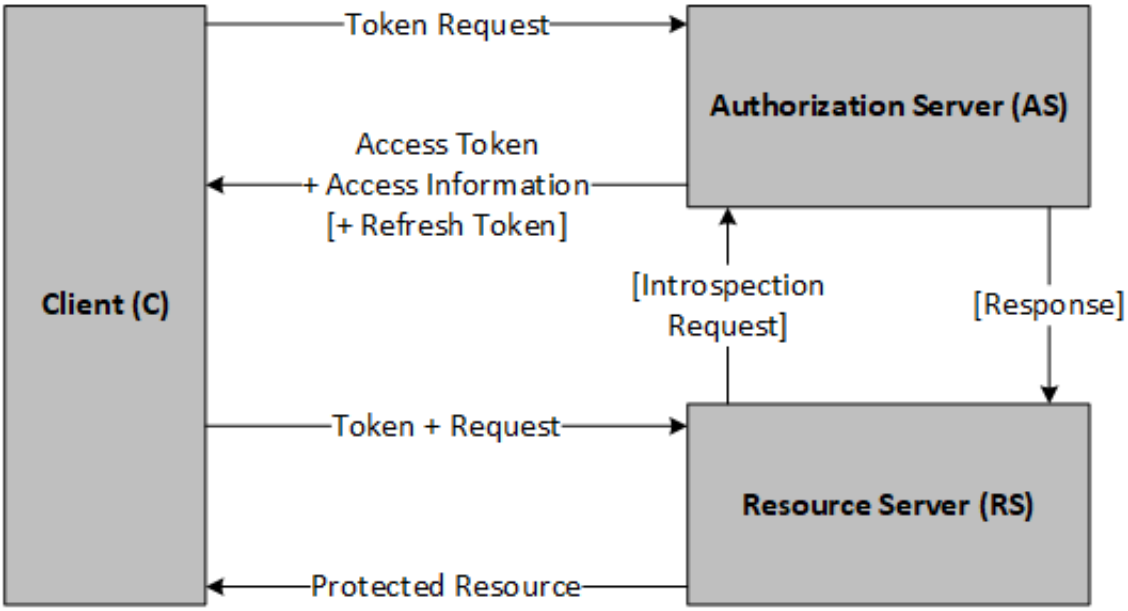

Fig. 1. Protocol Flow of ACE OAuth without implementation details or security properties

4 SECURITY ANALYSIS OF ACE-OAUTH

In this work we focus on a security analysis of the protocol flow as described in the specification and there is a careful level of abstraction we must choose to follow as a consequence. IETF specifications are often, by design, generic and offer a lot of options. We want to maintain a high a level of granularity to ensure accuracy and fealty to the original specification.

4.1 Security Requirements

In order for an authentication protocol between two parties to be considered as functioning correctly we must first discuss the meaning of a successful authentication protocol. We will consider protocols that aim to authenticate two IoT devices, referred to as agents, with or without help of a third party (such as an authentication server). We use the word “role” to refer to the purpose an agent is taking in the protocol run (i.e. initiator, responder or server). So to check whether the protocol successfully achieved this we must, for every run of the protocol, check that this definition holds.

As described in Lowe and Gavin's seminal work [4] the following properties must hold in order to successfully deploy an authentication protocol starting from the weakest definition of aliveness to the strongest of injective agreement. We also refer to these conditions as security properties, as if these properties hold, it means that the authentication is successful and therefore the attacker cannot break the protocol or impersonate one of it's participants.

The properties are described below:

Goal 1 - Aliveness: A protocol guarantees to an agent a in role A aliveness of another agent b if, whenever agent a completes a run of the protocol, apparently with agent b in role B, then agent b has previously been running the protocol.

Goal 2 - Weak Agreement: A protocol guarantees to an agent a in role A a weak agreement with another agent b in role B if, whenever agent a completes a run of the protocol, apparently with role b, then agent b has previously been running the protocol with A.

Goal 3 - Non-Injective Agreement: A protocol guarantees to an agent a in role A a non-injective agreement with another agent b in role B on a message t if, whenever an agent a completes a run of the protocol, apparently with role B, then agent b has previously been running the protocol with agent a and they both agreed on the message t.

Goal 4 - Injective Agreement: A protocol guarantees to an agent a in role A a non-injective agreement with another agent b in role B on a message t if, whenever an agent a completes a run of the protocol, apparently with role B, then agent b has previously been running the protocol with agent a and they both agreed on the message t. Additionally, there is a unique matching partner instance for each completed run of an agent, i.e. for each Commit by an agent there is a unique Running by the supposed partner (see Protocol in Appendix A for details).

More protocol specific properties can be easily added without changing the formal specification of the protocol. The added properties verify that the flow of the protocol is not altered and it adheres to the original protocol. The properties can aid to check that the addition of changes doesn't infringe on the security requirement and that they do introduce vulnerabilities.

4.2 Formalizing the ACE-OAuth Security Requirements

In an effort to ensure correctness we analyzed the specification and translated each requirement into formal definitions of security as well as determining what level of security is needed at each level (and the downsides if the level is not met). It is worth noting that as the standard allows for different implementations some security specifics are implicit and therefore not covered. For example, there is an assumption that state of the art cryptography is used. Hence, we do not investigate brute force attacks against keys.

We break the protocol down into three phases and formalise the requirements of each phase:

(1) The first phase is the token grant phase whereby a device makes a token request to the Authorization Server and specifies the need for the token. Upon being verified a token is granted.
(2) The second phase is the resource request phase. Upon having been granted a token, the device proceeds to use the token to access a resource. Via the token the resource server learns about the granted permissions.

(3) The third and final phase is the token authorization and proof-of-possession phase. The protocol specification allows the token to be verified locally at the resource server or the token introspection mechanism to be used. Once this step is successfully completed, the device is granted access.

The requirements for the secure and correct functioning of the protocol as per the specification are discussed and formalised below (in the format <Document Name> <Section> <Page No> <Requirement Number>). We explain modelling choices and how to use the model for any given scenario, and to test variations.

4.3 Token Grant

The token grant is the core principle behind OAuth and within it lies the essence of the protocol. This enables for an untrusted user to authenticate with a potential resource and consequently gain access to it's content. Consequently, if the token is unprotected then an attacker can arbitrarily access resources in the system, so it is the single point that must be secured most efficiently.

> **ACE-OAuth, Section 6, p37, r1:** The AS must provide confidentiality protection for any interaction with the client.

Trivially, when we talk about confidentiality of information, we are talking about protecting the information from disclosure to unauthorised parties. Whilst this is often associated with encryption the protocol leaves this up to interpretation, we choose to keep to this interpretation but it could be possible to ensure this property in other manners. Formally, we require that for all messages in between the AS and C an Adversary may not know the content of the message unless the private encryption key of either party has been exposed.

As this is an authentication protocol there is an expectation that the AS and the client are able to mutually authenticate each other. There is an underlying assumption that the client has previously registered with the specific AS.

> **ACE-OAuth, Section 5, p15, r2:** It is REQUIRED that the endpoint and the AS are mutually authenticated and the responses MUST be bound to the request.

Since we assume that the device has been pre-registered with the AS and there is some means for the client to know that the client is legitimate. The AS must therefore demonstrate injective agreement on the request to the client C, this is a very strong assumption, to hold and perhaps there are some environments in which this might not be needed as we will discuss in Section 5.4. However the intent is to model the spec accurately for future use so we keep these requirements strict. If one wishes to test different setups to ensure the requirement holds this would require a switch from the modelled secure channel to an authenticated one (provided in the model) with their desired cryptographic scheme. The requirement will then be checked and the setup can be altered until it is successfully met.

4.4 Resource Request

Unlike the Token Grant phase where a client is pre-registered with the AS, the initial communication between the client C and the resource server RS will be insecure. The client therefore has to provide proof that he has access to the resource and that the client is the intended owner of the token.

> **ACE-OAuth, Section 5.1.1, p16, r2:** A resource request MUST be rejected if: the request is sent on an unprotected channel, the access token is invalid or not present or if the token is valid but not for the specified resource

These requirements might seem reasonable, however their ambiguity leads to potential issues in the protocol setup. If there is no previous registration between the C and the RS then a unprotected channel will be the norm. An unprotected channel cryptographically speaking is one to which an adversary has read, write and intercept access.

We proceed by modelling this as a non protected channel as there is no means for previous authentication. This will mean that authentication will have to be performed afterwards to ensure that the requirements are met.

> **ACE-OAuth, Section 5.8.2, p35, r1:** If a client makes a request to the RS it MUST show that is has the access token that validates this request.

As the token is bound to a specific party the leakage on an insecure channel of the token does not imply that an adversary is able to access the resource. However, if there is no second authentication phase the request should not be accepted.

> **ACE-OAuth, Section 6, p39, r1:** The secure channel between the client and RS must provide encryption, integrity and replay protection.

The requirement here is very loose on how this is achieved, and can be done in multiple ways. Formally, the RS must provide injective agreement to the client on the request and subsequent granting of the resource that is requested furthermore the requested resource must not be known to an attacker unless the private encryption key of either party has been leaked.

For the purpose of usability we formally verify that if these conditions are met the desired properties hold and the protocol can be executed to completion securely. In our case study we investigate one method in which this is ensured through use of Signing and Asymmetric encryption on the request and token, but once again multiple scenarios can be explored including the addition of the further phase of token introspection.

4.5 Token Introspection

Whilst this step is stated as optional in the ACE-OAuth specification [7], it represents one of the most useful features of the ACE-OAuth framework. In opposition to pure OAuth the AS in ACE may provide aid in authenticating the client on behalf of the RS as well as creating a secure channel between them. As such, it is essential that the communication is properly secured. This is specifically pertinent to some very low power scenarios that might struggle to perform complicated asymmetric exchanges to ensure authentication.

> **ACE-OAuth, Section 5.7, p27, r1:** The communication between the AS and RS must be integrity protected and encrypted.
>
> **ACE-OAuth, Section 5.7, p27, r2:** The communication messages between the AS and the RS must have a binding between request and response
>
> **ACE-OAuth, Section 5.7, p27, r3:** The AS must ensure the requesting entity has the rights to request information about the specific token

These requirements match the previously formalised requirements and hence we do not repeat them. What we want to investigate is whether security is achieved with these requirements in place permitting the safe protocol flow. We also want to investigate whether they apply to all the possible variations of the protocol setups. We asked ourselves the following questions: Given a set of requirements how can we test that they are all needed? If different setups are tested do the same requirements apply?

We attempt to answer these questions in the case study and demonstrate how to answer them with use of automated verification tools.

5 EXAMPLE EXTENSION: SMART FACTORY

We asked our co-workers to design a profile of the ACE-OAuth framework based on a use case they cared about. Developing a profile is necessary to restrict the number of options available with the ACE-OAuth framework. We also wanted to explore the validity of our model and to determine whether our security requirements were complete. Finally, we wanted to know how easy it would be to apply the model to an extension. Already during the discussions we noticed that the translation of the extension into a formal model helped a lot to come up with good questions.

The purpose of the extension was to use the ACE-OAuth framework in a smart factory where technicians make changes to machines on the factory floor as well as perform firmware updates. Only authorized technicians working for the company should be allowed to run maintenance jobs on machines. Hence, there is a need for the technician to authenticate with the Authorization Server, which issues the tokens.

Only public key cryptography was used in the profile. Hence, the client (which runs on a tablet) is provisioned with a trust anchor of the AS and has a certificate plus a private key, the AS is provisioned with a certificate and a private key, and the RS is also provisioned with the trust anchor of the AS and a certificate plus the corresponding private key.

After the Client authenticated the AS and vice versa, the AS creates a token. This token is digitally signed by the AS and includes the proof-of-possession key (i.e. a public key provided by the Client with the token request).

When the Client interacts with the RS it not only needs to make a resource request and include the token but it also needs to demonstrate possession of the private key that corresponds to the public key included in the token. Since the communication between the Client and the RS in this profile uses Bluetooth Low Energy a new payload was defined that encapsulates the resource request, a timestamp, and the token. We called this payload the 'Operation Bundle". This Operation Bundle is digitally signed with the private proof-of-possession key.

The RS did not use token introspection in our profile but instead processed the self-contained token locally. This required the RS to first use the stored trust anchor to verify the digital signature covering the token, to extract the public key and to use it to verify the digital signature covering the Operation Bundle, to check for replay using the timestamp, and to determine whether the Client was allowed to access the protected resource based on what was contained in the claims of the tokens.

5.1 Security Properties of the Smart Factory Use Case

As highlighted in the previous section ACE-OAuth allows for deployment flexibility and to make our security analysis detailed we have to pick a subset of the offered features that are useful for a given scenario.

What we wanted to demonstrate with this case study is that it is possible to find security implications of each design decision allowed by the specification. This allows to evaluate the security implications of each decisions and to generate a set of minimum requirements for the different protocol variants. Depending on the core security requirements in the overall protocol it sometimes may not be possible to have that level of assurance given specific implementation details.

Protocol verification is not black and white but rather allows for a lot of freedom to work towards a specific scenario that matches ones needs whilst still getting assurance that the analysis is correct (for the specific setup). Below we show an example of how this is done starting from: the initial analysis, finding the minimum security requirements and risk assessment in specific scenarios.

5.2 Token Grant

The ACE-OAuth specification determines that there should be a secure channel between the Client and the AS. There is, however, no mention of requiring a secure channel between the user (in this case a technician) and the AS. Whilst this can be considered out of the scope of the ACE-OAuth protocol itself, it is a scenario that is not unlikely given the specific environments that ACE-OAuth is aimed towards. Of course, OAuth also consider the user authentication outside the scope and hence it was logical to follow the same design spirit also in ACE-OAuth. It is, however, worthwhile to think about the implications of this user authentication step and how it can be modelled.

5.3 Resource Request & Confirmation - Operation Bundle

The Operation Bundle is a construct that cannot be found in the ACE-OAuth specification itself but would, at least conceptually, be present in one of the ACE-OAuth profiles that explain how the interaction between the Client and the RS works. As described previously, the idea of the Operation Bundle was to combine the resource request, the token and a timestamp together and to subsequently protect it with a digital signature. The Operation Bundle corresponds to an application layer remote procedure call that uses CBOR/COSE.

5.4 Discussion

Automated verification tools are often thought of as ways to "break protocols". However, the main reason for using these tools is to ensuring protocol security. Designing a verification tool is much harder because to have assurance of security one must cover every scenario whilst finding an attack only requires a single insecure scenario to be found, as argued in [2, 5].

As a limitation, a protocol designer has to find the appropriate level of abstraction when translating a specification into formal language and to correctly implement the protocol in Tamarin.

The initial phase was to test out a specific case study and gather the full results in Section 5.4, then we use the results to generate new requirements that can ensure the security of the extension and then discuss these in terms of the protocol to show how the written spec can be extended and adjust to fit the analysis we conducted.

We kept strictly to the specification requirements, these requirements are two phase:

(1) Token grant requirements: secrecy of request, injective agreement on the request and injective agreement on the token,
(2) Resource access: secrecy of resource and injective agreement on the resource.

The results of a naive implementation of the ACE-OAuth profile are shown in Table 1, which lists several vulnerabilities. Subsequently, we will fix these vulnerabilities since they are, in part, introduced by the flexibility of the ACE-OAuth framework. Whilst this initial analysis might look alarming, these results are caused by small and easy to fix design decisions.

		Secrecy	Injective-Agreement
Token Grant	Token	✓	*X*
	Request	✓	*X*
Resource Request	Response	*X*	*X*

Table 1. Naive Implementation of ACE-OAuth Profile

5.5 Generation of Minimal Requirements

The desired effect of generating minimal requirements is to enable the implements of ACE-OAuth to have guided means to test variations for ACE-OAuth. One of the key characteristics of the IoT is it's dynamicity and heterogeneity, to cater to the vastly different scenarios the protocol allows for several profiles to be constructed from the base protocol. However, each variation introduces subtle differences to the protocol that in turn could introduce a vulnerability. To avoid the vulnerabilities being introduced a set of minimal requirements needs to be specified for each variation that is allowed. We define the minimal requirements as the smallest possible requirements in order to still meet the outlined security goals. These can vary depending on what rigidity of goal one has and as can be seen they can differ. One can therefore make exact decisions of how to design their protocol to meet the security objectives of their project. We outline our methodology for the above case-study, given the results shown in the Section 5.1, we adjusted for the possible attacks on the protocol we discovered and made adjustments the setup.

It is to be noted that sometimes it is not possible to adjust the setup in a real world environment and in that case it is still worth knowing what threats the design decision leaves us open to. We will discuss in the later section why there is still a risk of token leakage in our current setup, as there is a vulnerability to do with the resource request that is found in the specification.

To generate the requirements we introduced a bottom up approach, this involved adding layers of security until the requirements were met. This investigation proved very useful as it discovered design flaws as well as variations to the security necessary to meet the goals. We highlight the set of security procedures needed to achieve the outlined goals in the tables below. We use minimal assumptions (additionally to the spec required ones) for the token request to obtain the desired security properties.For our specific analysis we assume that an attacker cannot access any information created by participants in the protocol until it is sent out on a communication channel.

		Secrecy	Injective-Agreement
Token Grant	Token	$\neg K$	$\neg K \wedge S_{cA}[*]$
	Request	$\neg K$	$\neg K$

Table 2. Updated Security Requirements for Token Grant of ACE-OAuth Profile

The table should be read as follows: We use notation K to represent key reveal so $\neg K$ means assumption holds if the key is secret and notation $S_{CA}[^*]$ to represent a secure channel between two parties (in this case C and AS) with the variables that need to be passed on that channel in the [ ] and SR_{CA} confirming the need for replay protected secure channel

What was of more interest was that the requirements could be loosened for part of the protocol and still achieve the same security goal. Namely the timestamp was only used to avoid the replay of the token as the request in phase two cannot be replayed due to the request only matching the token, this also applied to the full ACE-OAuth spec and is something that should be taken in consideration for potential extensions.

This allows us to loosed the requirement so that a replay protected channel is not necessary. This change still allows us to meet the full security goals specified by the specification whilst increasing performance and strain on the devices.

Table 3 uses the same notation as Table 2 and the term V indicates that signature verification is needed (in our case through a public key infrastructure).

In phase two, three aspects need to be considered in modelling:

		Secrecy	Injective-Agreement
Resource Request	Response	$V[T, B, ts] \wedge SR_{cp}[*] \wedge \neg K[*]$	$V[T, B] \wedge SR_{cp}[*] \wedge S_{RP}[]$

Table 3. Updated Security Requirements for Resource Request of ACE-OAuth Profile

(1) Timestamps were used for replay protection in the Operation Bundle. The RS has to check for replays using time information available locally.
(2) Token introspection is not used in our profile; hence the RS is not be able to communicate with the AS in real-time.
(3) The token itself is public information and does not require confidentiality protection.

In some scenarios a requirement may not be implementable due to device constraints. While not necessarily advisable, it may be desirable to relax certain requirements while still following the protocol for interoperability. In such a case it is important to determine what vulnerabilities are thereby introduced.

6 CONCLUSION

Analysing security protocols using formal methods is valuable to demonstrate that the modelled protocol meets the indicated security goals. It also helps to justify design decisions. In this paper we focused on ACE-OAuth and an extension we developed for illustration purposes. We believe the use of tools like Tamarin make the formal protocol verification easy enough for protocol designers to utilize. We present a methodology to automatically verify extensions to the core flow of ACE-OAuth using Tamarin and show the validity of the approach through our case study. The model is able to guide designers and engineers alike in making safe and secure decisions with ease. By editing small portions of the model vast new avenues of attack are introduced and consequently found through formal analysis. We firmly believe that using formal models for protocol verification is no longer optional and we seek to provide means to simplify the burden for future work on the ACE-OAuth protocol by using our model.

During our work we made the following observations:

- Earlier formal analysis of ACE-OAuth was incomplete and used Avispa and Scyther, which are also tools for automatic verification. These tools are, however, older and less frequently used.
- We hope that extensions to the ACE-OAuth protocol will make use of our model. The added effort for designers of such extensions will be relatively small.
- We found a few security aspects that are important to consider I.E. The use of multiple audiences needs to be carefully guarded with stronger verification, some allowed for variants could be implemented in a security breaching manner.
- Perhaps most importantly, we have provided the first formal proof that the ACE-OAuth protocol is verifiably secure (albeit if implemented as per the requirements and without improper variations)

7 APPENDIX A: TAMARIN CODE

The code for the generalized ACE-OAuth model as well as for the ACE-OAuth profile can be found at https://github.com/Yiergot/ACE-OAuth-FormalModel.

REFERENCES

[1] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. 2014. Easycrypt: A tutorial. In *Foundations of security analysis and design vii*. Springer, 146–166.

[2] Bruno Blanchet. 2003. Automatic verification of cryptographic protocols: a logic programming approach. In *PPDP*, Vol. 3. Citeseer, 1–3.

[3] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (March 1983), 198–208. https://doi.org/10.1109/TIT.1983.1056650

[4] Gavin Lowe. 1997. A hierarchy of authentication specifications. In *Computer security foundations workshop, 1997. Proceedings., 10th*. IEEE, 31–43.

[5] Simon Meier. 2013. *Advancing automated security protocol verification*. Ph.D. Dissertation. ETH Zurich.

[6] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*. Springer, 696–701.

[7] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. 2019. *Authentication and Authorization for Constrained Environments (ACE)*. Internet-Draft draft-ietf-ace-oauth-authz-18. Internet Engineering Task Force. https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-18 Work in progress.

[8] Goldwasser Shafi and Silvio Micali. 1984. Probabilistic encryption. *Journal of computer and system sciences* 28, 2 (1984), 270–299.