AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle

Antonio L. Maia Neto
UFMG,Brazil
lemosmaia@dcc.ufmg.br

Artur L. F. Souza
UFMG, Brazil
arturluis@dcc.ufmg.br

Italo Cunha
UFMG, Brazil
cunha@dcc.ufmg.br

Michele Nogueira
UFPR, Brazil
michele@inf.ufpr.br

Ivan Oliveira Nunes
UFMG, Brazil
ivanolive@dcc.ufmg.br

Leonardo Cotta
UFMG, Brazil
leonardo.cotta@dcc.ufmg.br

Nicolas Gentille
LG Electronics, Brazil
nicolas.gentille@lge.com

Antonio A. F. Loureiro
UFMG, Brazil
loureiro@dcc.ufmg.br

Diego F. Aranha
Unicamp, Brazil
dfaranha@ic.unicamp.br

Harsh Kupwade Patil
LGE Mobile Research, USA
harsh.patil@lge.com

Leonardo B. Oliveira
UFMG, Brazil
leob@dcc.ufmg.br

ABSTRACT

The consumer electronics industry is witnessing a surge in Internet of Things (IoT) devices, ranging from mundane artifacts to complex biosensors connected across disparate networks. As the demand for IoT devices grows, the need for stronger authentication and access control mechanisms is greater than ever. Legacy authentication and access control mechanisms do not meet the growing needs of IoT. In particular, there is a dire need for a holistic authentication mechanism throughout the IoT device life-cycle, namely from the manufacturing to the retirement of the device. As a plausible solution, we present *Authentication of Things* (AoT), a suite of protocols that incorporate authentication and access control during the entire IoT device life span. Primarily, AoT relies on Identity- and Attribute-Based Cryptography to cryptographically enforce Attribute-Based Access Control (ABAC). Additionally, AoT facilitates secure (in terms of stronger authentication) wireless interoperability of new and guest devices in a seamless manner. To validate our solution, we have developed AoT for Android smartphones like the LG G4 and evaluated all the cryptographic primitives over more constrained devices like the Intel Edison and the Arduino Due. This included the implementation of an Attribute-Based Signature (ABS) scheme. Our results indicate AoT ranges from highly efficient on resource-rich devices to affordable on resource-constrained IoT-like devices. Typically, an ABS generation takes around 27 ms on the LG G4, 282 ms on the Intel Edison, and 1.5 s on the Arduino Due.

CCS Concepts

•**Security and privacy** → **Authentication; Access control;** *Embedded systems security;* •**Networks** → **Security protocols;**

Sensys 2016 November 14–16, 2016, Stanford, CA, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4263-6/16/11.

DOI: http://dx.doi.org/10.1145/2994551.2994555

Keywords

Internet of Things; Access Control; Authentication; Security; Identity-Based Cryptography; Attribute-Based Cryptography

1. INTRODUCTION

The Internet of Things (IoT) [1, 2] is now part of our daily lives. The vision where highly heterogeneous networked embedded computing elements are interconnected to provide a smart environment has finally come true. IoT combines physical objects (e.g., vehicles and appliances) with sensors and actuators in a cyber-physical system (e.g., smart cities, grids, and homes). Projections suggest that we will soon be surrounded by billions of IoT-enabled devices.[1]

Authentication is a fundamental security property for IoT [3]. Although IoT security received enormous attention from the research community (e.g., [4–10]), we still claim that existing approaches do not fulfill the needs for seamless authentication. First and foremost, traditional schemes are based on Public Key Infrastructure (PKI) and certificates incur significant processing, memory, storage, communication, and management overheads, and are deemed unfit for IoT devices [11]. Second, the IoT paradigm beckons safe and seamless interoperability between disparate home and foreign domains. However, several authentication schemes targeting resource-constrained devices (e.g., [12–25]) assume that such devices belong to a single domain and, therefore, cannot be directly applied to IoT. Last, to the best of our knowledge, there is no mechanism that facilitates authentication during the entire IoT device life-cycle (i.e., from the manufacturing to the retirement of a device). As a result, there is a need for novel solutions exclusively tailored to meet the requirements of authentication in the IoT paradigm.

Our goal: In this paper, we aim at designing, developing, and evaluating an authentication and access control scheme for the entire IoT device life-cycle. Our solution, *Authentication of Things* (AoT), is a suite of cryptographic protocols that provides authentication and access control to all stages in a device's life-cycle (Figure 1), in particular: pre-deployment, ordering, deployment, functioning, and retirement. This implies AoT can authenticate devices

[1]http://www.gartner.com/newsroom/id/2636073

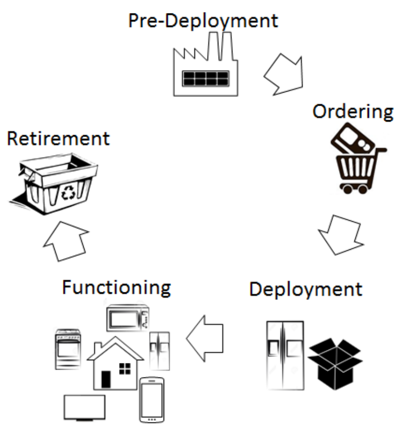

Figure 1: Entire IoT device life-cycle.

in multiple distinct domains.

AoT allows authenticated device interoperation across multiple domains. A device can join a foreign domain (i.e., a domain different from its own) and securely authenticate itself to access operations on devices in that domain. AoT also allows secure transfer of device ownership. Finally, AoT performs authentication and access control over wireless mediums, but provides stronger security than wireless authentication mechanisms widely deployed today (e.g., WPA2).

In order to accomplish our goals, AoT relies on Identity-Based Cryptography (e.g., [26–29]) to distribute keys and authenticate devices as well as Attribute-Based Cryptography (e.g., [30, 31]) to cryptographically enforce Attribute-Based Access Control [32,33]. We chose these cryptosystems because they are certificate-free and thus do not impose certificate-related overheads on devices. Our *key insight* is to tackle the well-known key escrow problem [34] of identity-based schemes designing a two-domain architecture composed of a manufacturer (*Cloud*) domain and a local (*Home*) domain. These domains manage manufacturer-to-device and domestic device-to-device trusted relationships, respectively.

We evaluate AoT both analytically and experimentally. We provide analytical worst-case performance estimates and employ tool-assisted verification [35,36] to check AoT security claims. We also implemented a multi-architecture prototype and evaluate all cryptographic primitives on resource-constrained devices like the Intel Edison and Arduino Due. Our AoT prototype contributes an open-source implementation of Attribute-Based Signatures (ABS) [37, 38], a central part of AoT, to the community. We use our prototype to quantify CPU, memory, storage, and communication overheads imposed by our suite of protocols. Our results indicate AoT ranges from affordable on resource-constrained devices like the Arduino Due to very efficient on smartphones like the LG G4.

Our contributions: AoT brings forth both theoretical and practical findings. Our main contributions are summarized as follows.

1. AoT, a suite of protocols to provide authentication and access control over the entire IoT product life-cycle. AoT provides authentication and access control on Cloud and Home domains. AoT also allows a guest device to authenticate itself with devices from a foreign Home domain.

2. An analytical and experimental evaluation of AoT. We provide analytical performance estimates for AoT and use tool-assisted verification to check AoT's security claims. Our experimental evaluation is based on a real prototype and quantifies various performance metrics on diverse hardware.

3. To our knowledge, the first ABS experimental evaluation ever published. Our implementation is publicly-available and based on the work of Maji, Prabhakaran, and Rosulek [37]. It is efficient enough to be run on resource-constrained devices.

Organization: The remainder of this paper is structured as follows. Section 2 introduces concepts, security definitions, cryptographic schemes, and the attack model we use. Section 3 presents the AoT protocol and the assumptions under which it has been designed. Section 4 describes the development of AoT detailing the implementation and optimization of cryptographic primitives. Section 5 evaluates AoT security and resource requirements. We discuss related work in Section 6 and conclude in Section 7.

2. BACKGROUND

In this section, we cover the fundamentals of AoT, namely Authentication in Section 2.1, Attribute-Based Access Control (ABAC) in Section 2.2, Identity-Based Cryptography (IBC) in Section 2.3, Attribute-Based Cryptography (ABC) in Section 2.4, and Pairing-Based Cryptography (PBC) in Section 2.5.

2.1 Authentication

Authentication [3] is allegedly the most important security property in IoT. One of authentication's major goals is to prevent illegitimate nodes from taking part in network activities. This can protect most network operations, unless legitimate nodes have been compromised.

Broadly speaking, authentication comprises two properties. The first is *source authentication* [11], which guarantees a receiver that the message indeed originated from the claimed sender. The second is *data authentication* [11], which prevents integrity violation, i.e., it prevents that a message is altered while in transit between the sender and the receiver, and ensures that the received message is "fresh", i.e., not being replayed.

In the world of cryptography, authentication can be achieved through the use of symmetric or asymmetric (public-key) cryptosystems [11]. More precisely, through the use of Message Authentication Codes (MACs) [11] and Digital Signatures [11], respectively. (It is worth noting only the latter provides nonrepudiation, i.e., prevent a device from denying previous commitments or actions.)

AoT leverages both MACs and Digital Signatures for authentication in its suite of protocols.

2.2 Attribute-Based Access Control

Access Control [39] regulates who or what (e.g., a user) can perform an operation (e.g., read or write) on an object (e.g., a file).

Traditional access control models are inconvenient for IoT [32]. For instance, Mandatory Access Control, Discretionary Access Control, and Role-Based Access Control are user centric and do not consider factors like resource information, relationship between the user (the requester) and the resource provider, and dynamic information (e.g., current time and user identifier). Moreover, in large scale networks like IoT deployments, it might be unmanageable for one to keep a list of who is granted access to what.

ABAC [32], on the other hand, simplifies access control by replacing discretionary permissions with policies based on attributes. The model grants rights to users based on attributes like resource characteristics and contextual information. The idea is based on the observation that, in a given organization, permissions are often assigned to the attributes of users rather than to their identity. Attributes, in turn, are assigned to users according to their responsibilities or qualifications. In ABAC, the possession of an attribute can be easily altered without modifying the underlying access structure; and new permissions can be conveniently granted to attributes as new objects or operations are incorporated into the system.

AoT adopts ABAC as the paradigm to implement access control for IoT.

2.3 Identity-Based Cryptography

The notion of IBC dates back to Shamir's original work [26], but it has only become practical with the advent of PBC [27, 40–42]. The main advantage of IBC is that it does not require the expensive explicit public-key authentication (like traditional PKI does). In these systems, users may have a meaningful public-key rather than a random string of bits; and those keys can thus be derived from the user's public information [34, 43]. Note this information intrinsically binds a user to its public-key, which makes other means of accomplishing this binding, e.g., digital certificates, unnecessary [34, 43]. Finally, IBC also allows secure communication between users of different IBC domains [44].

IBC, however, is not a panacea and it turns out that private-keys in IBC are not generated by their respective owners but rather by Private Key Generators (PKGs), i.e., those keys are not actually private. So, a PKG could, if it wanted to, impersonate any user in the system [34, 43]. This is the well-known key escrow problem of identity-based systems; and the main challenge for the wide adoption of IBC [34, 43].

A particularly strong requirement of IBC is that private-keys must be delivered to the user over a secure channel [34, 43]. In practical scenarios where IBC is to be used to bootstrap security, such a secure channel might not exist.

In IBC, the cryptographic primitive that provides authentication is the Identity-Based Signature (IBS). AoT makes use of IBS in its suite of cryptographic protocols.

2.4 Attribute-Based Cryptography

ABC [30, 31], also known as *Fuzzy* IBC [45], is an extension of the idea used in IBC. As such, ABC also suffers from the key escrow problem. Compared to IBC, ABC focuses on groups of users rather than solely on users' identities. The cryptosystem relies on a subset of user attributes to control private-key ownership.

Broadly speaking, there are two classes of ABC schemes, namely Key-Policy ABC (KP) (e.g., [30]) and Ciphertext-Policy ABC (CP) (e.g., [31]). In the former, the policy is attached to private keys, and attributes annotate messages. In the latter, messages carry the policy and users possess a key for each of their respective attributes. There is a close fit between KP and applications that deliver digital content like cable TV [30]. In KP, however, the sender of a message has no control over who or what will be able to access the contents of his messages. CP, on the other hand, does allow this control.

In the context of signature schemes, CP is commonly referred to *Policy-Endorsing Attribute-Based Signature* [46]. Here, users are assigned a set of attributes and the corresponding private keys. A user's ability to perform operations over a message (e.g., sign a message) depends on the attribute set associated with the user as well as the policy associated with the message. To be concrete, messages carry a boolean expression of attributes called the signing policy or the *predicate* of the message. To verify a signature correctly, a user must sign the message using the keys associated with any subset of attributes that satisfies the predicate.

It is worth stressing the synergy between ABC and ABAC (Section 2.2). On the one hand, ABC is similar to ABAC in that they both are based on user attributes. On the other hand, ABC and ABAC are also complementary, since they may be combined so the former cryptographically enforces the later.

We chose CP to be the underlying cryptographic construction of AoT's access control mechanism. Precisely, AoT employs ABC signatures (ABS for short) during its most frequent operation and maps ABAC policies onto ABC predicates.

2.5 Pairing-Based Cryptography

PBC [27, 28, 47] has paved the way for the design of original cryptographic schemes and made existing cryptographic protocols both more efficient and convenient. It has also shed some light on many long-standing open problems allowing quite a few of them to be solved elegantly. Identity-Based Encryption (IBE) [28] is most likely the main evidence of this, as IBE has enabled complete IBC schemes. (But note that there are other pairing-free ways of performing IBE today [29, 48].)

The bilinear pairing[2] is the major cryptographic primitive in PBC. Pairings, for short, were first used in the context of cryptanalysis [40], but their pioneering use in cryptosystems is due the works of Sakai, Ohgishi, and Kasahara [27], Boneh and Franklin [28], and Joux [47]. Formally, a *bilinear pairing* is a computable, non-degenerate function $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additively-written groups of order n with identity $\mathcal{O}$, and $\mathbb{G}_T$ is a multiplicatively-written group of order n with identity 1 [49]. In practice, groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are implemented using a group of points on certain elliptic curves and the group $\mathbb{G}_T$ is implemented using a multiplicative subgroup of a finite extension field [49]. When $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is said to be symmetric, although symmetric pairing constructions are less efficient. In PBC, the crucial property of pairings is bilinearity [49]:

$$\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab} \;\; \forall\, P \in \mathbb{G}_1, Q \in \mathbb{G}_2 \text{ and } \forall\, a, b \in \mathbb{Z}_n.$$

AoT relies heavily on PBC. For instance, AoT makes use of pairings to distribute keys and to implement ABS.

3. AUTHENTICATION OF THINGS

In this section, we present AoT. AoT comprises a suite of protocols that cover all stages of an IoT device's life-cycle. We provide an overview of AoT (Section 3.1), present the auxiliary and main protocols (Sections 3.2–3.7), and discuss complementary features (Section 3.8).

3.1 Overview

Assumptions. In what follows, we assume (i) every message is numbered and carries the identifiers of the interlocutors; (ii) cryptographic material can be securely loaded into devices at the manufacturer's facility; (iii) PINs can be securely entered onto devices at home; (iv) cryptographic primitives are ideal—i.e., flawless—and can be treated as black boxes; (v) the device manufacturer is trusted.

Problem. In IoT, even elementary questions as how to enable authentication and access control remain unanswered. Traditional PKC is computationally expensive and most IoT devices cannot afford to run them. Besides, because of IoT's heterogeneity and multiple domains, each with different owners and access policies, security schemes for other wireless and resource-constrained networks (e.g., [13–20, 23–25, 50]) cannot be directly applied to IoT. To make things worse, questions regarding the portability and mobility of "things" arise. Portability and mobility are typical of IoT

[2]As of now, only bilinear pairings are useful for cryptographic constructions.

and they reinforce the call for inter-operation between local and guest devices.

Goal. Our ultimate goal is to design a tailor-made authentication and access control solution for IoT. We target devices like home appliances that interact with other home appliances, personal devices, and with a remote Cloud server that plays the role of the devices' manufacturer. Our solution must meet the efficiency and security needs of IoT applications.

Approach. As usual, our approach concentrates (i) mostly on key distribution to bootstrap security (ii) and mainly on access control to govern permissions over device operations. However, we implement those in a novel way: IBC is used to distribute keys and ABC to control access to device operations.

Our key insight to tackle the key escrow problem of IBC (Section 2) is a two-domain architecture (Figure 2). More precisely, our solution comprises two distinct IBC setups, namely: a manufacturer (*Cloud*) setup and a local (*Home*) setup. They respectively define manufacturer-to-device and domestic device-to-device trusted relationships. There is no overlap in these trusted relationships and thus an artifact generated in the Cloud domain is invalid in the Home domain and vice-versa. Note that the key escrow still holds in each IBC setup individually; however, the escrow now is no longer a problem. For the Cloud's IBC keys escrow, this is because the user's privacy is preserved in that requests originated from the Cloud domain are null in the Home domain. For the Home's IBC and ABC keys escrows, the context of where it takes place already deals with the problem. In other words, the escrow will be held at a home device owned and managed, and thus trusted, by the household and its members.

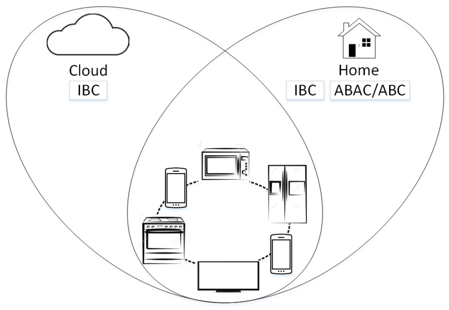

Figure 2: AoT two-domain architecture.

Our suite of protocols cryptographically provides authentication and access control over wireless mediums. One of AoT's key features is the ability to deploy devices without resorting to cabled connections. Instead, AoT leverages a home's physical protection and a reliable device (e.g., the Home domain manager's smartphone) to bridge the very first communication between a new device and the Home server to bootstrap security. This communication can happen over different encodings and transmission technologies. AoT delivers access control by combining ABAC and ABC so the former cryptographically enforces the later. Here, a permission for operating a device is ultimately tied to the ABC attributes of a requester; and the ABC attributes and predicates are governed by the higher-level ABAC policies.

Life-Cycle. In AoT, an IoT life-cycle is broken down into five main stages, namely: (1) *pre-deployment*, (2) *ordering*, (3) *deployment*, (4) *functioning*, and (5) *retirement*. By the way of example, consider a given device life-cycle. In the *pre-deployment*, the cryptographic material of the Cloud domain is loaded into the device at the factory, i.e., during its manufacturing process. Next, in *ordering*, the (about to become) owner of the device purchases it and gets a PIN that grants the owner the initial access to the device. *Deployment*, as its names suggests, is when the device is deployed home and therefore is the stage responsible for bootstrapping security in the Home domain. Here, the owner uses the PIN to access the device and the device, in turn, uses the owner's personal device to establish a trusted relationship with the Home server. (The Home server is the one in charge of managing keys and orchestrating access control inside home. For instance, the Home server issues IBC and ABC keys as well as advertises access permissions over the domestic network.) Finally, during this stage, the device is also bound to a user in the Cloud domain. Now, the way is paved for *functioning*, which corresponds to the daily operation of the device. At this point, users request the device's operations, and the latter reacts based on users' clearance levels. *Retirement* is the end point of a device life-cycle. It takes place whenever its owner will not use the device any further. During this stage there is a wipeout cryptographic material held by the device and the owner is unbound from the device in the Cloud domain.

Extra. AoT has also some bonus features. For instance, AoT enables interdomain interactions between devices on different Home domains, meaning that devices from different Home domains may interoperate seamlessly as long as their respective Home servers have previously agreed on some parameters. This strategy is appealing because it neither violates the identity-based nature of AoT, nor requires key escrow across the participating domains. Our suite of protocols also address device reassignment, enabling a user to trade or give away one or more of his devices. Finally, we explain how AoT achieves key revocation.

3.2 Auxiliary Protocols

AoT's auxiliary procedure *SessionKey* and protocols *KeyAgreement*, *KeyIssue*, *Binding*, and *Unbinding* (Figures 3–6) assist the main protocols (Sections 3.3–3.7) in deriving a session key, agreeing on a common pairwise key, issuing a private key, as well as binding and unbinding a device to a user in the Cloud domain, respectively. In ABC, multiple devices may hold the same ABS key; thus, ABS keys are often referred to as signing keys rather than private keys. For the sake of simplicity, however, we call them private keys.

SessionKey (Figure 3) relies on a pseudorandom function PRF, on a previously shared key k, and on a counter i that is kept up-to-date with the interlocutor to derive session keys. This idea is based on the work of Perrig *et al.* [50].

```
SESSIONKEY($key\ k$, $counter\ i$)
    1.    $k^i := \textsc{Prf}(i)_k$
    2.    $i := i + 1$
    3.    return $k^i$
```

Where the symbols denote:

k^i :	i-th session key
$\textsc{Prf}(m)_k$:	PRF over m using key k

Figure 3: SessionKey procedure.

KeyAgreement (Figure 4) is based on the work of Sakai, Ohgishi, and Kasahara [27] and runs a key agreement protocol between any two devices on the same Home domain. The protocol makes use of pairings ($\hat{e}$) for computing a pairwise key $k_{A,B}$, and counters (e.g., c_A) for generating unique session keys. Finally, note we also carry out a mutual challenge-response message exchange between A and B (steps 1, 6, and 8). We reproduce this sort of challenge-response message exchange in other AoT protocols to ensure freshness and prevent replay attacks.

KeyAgreement(*device* A, *device* B)

1.	$A \rightarrow B$:	n_A, session_req
2.	B :	$P^{\mathrm{I}}_{A,\mathcal{H}} := \mathrm{MAP}(id_{A,\mathcal{H}})$
3.	B :	$k_{A,B} := \hat{e}(S^{\mathrm{I}}_{B,\mathcal{H}}, P^{\mathrm{I}}_{A,\mathcal{H}})$
4.	B :	$c_A := 0$
5.	B :	$k^i_{A,B} := \mathrm{SESSIONKEY}(k_{A,B}, c_A)$
6.	$B \rightarrow A$:	$n_B, \mathrm{MAC}(n_A)_{k^i_{A,B}}$
7.	A :	{*Computes the pairwise key* $k^i_{A,B}$, *analogously*}
8.	$A \rightarrow B$:	$\mathrm{MAC}(n_B \mid n_A)_{k^i_{A,B}}$
9.	$B \rightarrow A$:	session_ack, $\mathrm{MAC}(n_A + 1)_{k^i_{A,B}}$

Where the new symbols denote:

$\rightarrow$:	unicast transmission
n_X :	nonce generated by X
session_req, _ack :	request and acknowledgment labels
I :	Identity-based cryptosystem
$\mathcal{H}$:	Home domain
$P^{\mathrm{Y}}_{X,\mathcal{Z}}$:	X's public key of cryptosystem Y in domain $\mathcal{Z}$
$\mathrm{MAP}(m)$:	mapping function over m
$id_{X,\mathcal{Y}}$:	X's identity in domain $\mathcal{Y}$
$k_{X,Y}$:	pairwise key shared by X and Y
$\hat{e}$:	crypto-friendly bilinear pairing
$S^{\mathrm{Y}}_{X,\mathcal{Z}}$:	X's private key of cryptosystem Y in domain $\mathcal{Z}$
c_X :	counter one keeps up-to-date with X
$k^i_{X,Y}$:	i-th session key shared by X and Y
$\mathrm{MAC}(n_X)_k$:	MAC over the message appended to n_X using key k
$\mid$:	concatenation

Figure 4: KeyAgreement protocol.

KeyIssue (Figure 5), in turn, issues a private key for a device D on a server S, domain $\mathcal{Z}$, and cryptosystem Y. The issuing is secured using a session key derived from a previously shared pairwise key $k_{D,S}$ between the device D and the server S.

KeyIssue(*device* D, *server* S, *domain* $\mathcal{Z}$, *cryptosystem* Y)

1.	$D \rightarrow S$:	n_D, issue_req
2.	$S \rightarrow D$:	$n_S, \mathrm{MAC}(n_D)_{k^i_{D,S}}$
3.	$D \rightarrow S$:	$\mathrm{MAC}(n_S \mid n_D)_{k^i_{D,S}}$
4.	$S \rightarrow D$:	$\mathrm{ENC}(S^{\mathrm{Y}}_{D,\mathcal{Z}})_{k^i_{D,S}}$, issue_ack, $\mathrm{MAC}(n_D + 1)_{k^i_{D,S}}$

Where the new symbols denote:

issue_req, _ack :	request and acknowledgment labels
$\mathrm{ENC}(m)_k$:	encryption over m using key k

Figure 5: KeyIssue protocol.

Binding (Figure 6) binds a device D to a user U in the Cloud domain. Briefly, the device D asks the Cloud server C to be bound to user U (step 3). Communication from the Cloud server C to the device D is authenticated using MACs (steps 2 and 5). Device D, conversely, employs IBC digital signatures to authenticate itself on the server C (step 3). The *Unbinding* protocol (not shown) is analogous to and performs the same steps as the *Binding* protocol. The only difference is that it unbinds rather than binds device D and user U.

3.3 Pre-Deployment

Pre-Deployment takes place at the factory and loads the Cloud domain's cryptographic material into devices. This material will be used, further, to establish a remote communication channel between devices and the manufacturer. Such a channel allows the execution of critical procedures by the manufacturer (e.g., software or firmware updates). Besides, it allows devices to run the auxiliary protocols with the Cloud server.

Figure 7 illustrates this stage. First, the Cloud server C generates the device D's identity $id_{D,C}$ (step 1) and, from that, C derives D's IBC private key $S^{\mathrm{I}}_{D,C}$ (step 2). Next, the Cloud server C generates a pairwise key $k_{D,C}$ and an access PIN pin_D, creates a counter c_D and sets it to zero, and loads them in tandem with all the remaining cryptographic material into device D (step 3). Note that the loading is secured via a physical channel, ensuring the communication is both confidential and authenticated. Finally, the Cloud server C deletes $S^{\mathrm{I}}_{D,C}$ and ships D to its trader T_D (step 4).

Binding(*device* D, *user* U)

1.	$D \rightarrow C$:	n_D, bind_req
2.	$C \rightarrow D$:	$n_C, \mathrm{MAC}(n_D)_{k^i_{D,C}}$
3.	$D \rightarrow C$:	bind, $id_{U,C}, \mathrm{SIG}(n_C \mid n_D)_{S^{\mathrm{I}}_{D,C}}$
4.	C :	binds U to D
5.	$C \rightarrow D$:	bind_ack, $\mathrm{MAC}(n_D + 1)_{k^i_{D,C}}$

Where the new symbols denote:

C :	Cloud server
bind_req, _ack :	request and acknowledgment labels
bind :	command label
$\mathcal{C}$:	Cloud domain
$\mathrm{SIG}(n_X)_k$:	signature over the msg appended to n_X using key k

Figure 6: Binding protocol.

Pre-deployment(*Device* D)

1.	C :	$id_{D,C} := D's\ serial\#$
2.	C :	$S^{\mathrm{I}}_{D,C} := \mathrm{GEN}^{\mathrm{I}}(\mathrm{secret}^{I}_{\mathcal{C}}, id_{D,C})$
3.	$C \rightarrow D$:	$\mathrm{PHY}(id_{D,C} \mid S^{\mathrm{I}}_{D,C} \mid k_{D,C} \mid \mathrm{pin}_D \mid c_D)$
4.	$C \Rrightarrow T_D$:	D

Where the new symbols denote:

$\mathrm{GEN}^{\mathrm{X}}(y, z)$:	private key generator of cryptosystem X using parameters y and z
$\mathrm{secret}^{X}_{\mathcal{Y}}$:	secret parameter of cryptosystem X in domain $\mathcal{Y}$
$\mathrm{PHY}(m)$:	m secured via a physical channel
pin_X:	PIN to grant acess to X
$\Rrightarrow$:	secure shipping
T_X :	trader of X

Figure 7: Pre-deployment stage.

3.4 Ordering

Ordering (Figure 8) illustrates a user U ordering a device D from trader T_D. Here, all digital communication is secured via TLS. The user U places the order and pays for the device (step 1). The trader T_D, in exchange, sends the user U an acknowledgment (step 2). At this point, the Trader lets the Cloud server C now about the order (step 3), and the Cloud server, in turn, sends the user U a PIN pin_D (step 4). Last, the Trader ships the device D out to user U's home (step 5). The PIN pin_D will be used in the *Deployment* stage (Figure 9) to grant access to D.

3.5 Deployment

Deployment (Figure 9) bootstraps the security of devices in their Home domains. It paves the way for the following protocols and is thus paramount for AoT. Here, the *root* user U_r—most likely, the household owner—and his personal device D_{U_r} play a key role. More precisely, D_{U_r} acts like a trusted bridge between the device being deployed and the Home server.

To set up a new device D, U_r enters D's access PIN pin_D onto

$\textsc{Ordering}(\textit{device } D, \textit{user } U)$

1.	$U \rightarrow T_D$:	TLS($ | order_req)
2.	$T_D \rightarrow U$:	TLS(order_ack)
3.	$T_D \rightarrow C$:	TLS(D | U)
4.	$C \rightarrow U$:	TLS(pin$_D$)
5.	$T_D \Rightarrow U$:	D

Where the new symbols denote:

TLS(m) :	m protected via TLS
$:	payment
order_req, _ack :	request and acknowledgment labels

Figure 8: Ordering stage.

$\textsc{Deployment}(\textit{device } D)$

1.	$U_r \rightarrow D$:	$\textsc{Phy}(\mathrm{pin}_D)$
2.	$D_{U_r} \rightarrow D$:	$\textsc{Phy}(id_{U_r,\mathcal{H}} \mid P^{\mathrm{I}}_{H,\mathcal{H}} \mid \mathrm{c}_{\mathbb{G}_H})$
3.	$D \rightarrow D_{U_r}$:	$\textsc{Phy}(id_{D,\mathcal{H}} \mid info_D \mid \textsc{Enc}(k_{D,H})_{P^{\mathrm{I}}_{H,\mathcal{H}}})$
4.	$U_r \rightarrow D_{U_r}$:	$\textsc{Phy}(\mathbb{A}_D \mid \mathbb{Y}_D)$
5.	$D_{U_r} \rightarrow H$:	$\mathrm{n}_{D_{U_r}}$, deploy_req
6.	$H \rightarrow D_{U_r}$:	$\mathrm{n}_H, \textsc{Mac}(\mathrm{n}_{D_{U_r}})_{k^i_{D_{U_r},H}}$
7.	$D_{U_r} \rightarrow H$:	deploy, $id_{D,\mathcal{H}}, \mathbb{A}_D, \mathbb{Y}_D, info_D,$ $\textsc{Enc}(k_{D,H})_{P^{\mathrm{I}}_{H,\mathcal{H}}}, \textsc{Sig}(\mathrm{n}_H \mid \mathrm{n}_{D_{U_r}})_{S^{\mathrm{I}}_{D_{U_r},\mathcal{H}}}$
8.	H :	$S^{\mathrm{I}}_{D_U,\mathcal{H}} := \textsc{Gen}^{\mathrm{I}}(\mathrm{secret}^{I}_{\mathcal{H}}, id_{D_U,\mathcal{H}})$
9.	H :	$S^{\mathrm{A}}_{D_U,\mathcal{H}} := \textsc{Gen}^{\mathrm{A}}(\mathrm{secret}^{A}_{\mathcal{H}}, \mathbb{A}_{D_U})$
10.	D :	$\textsc{KeyIssue}(D,H,\mathcal{H},\mathrm{I})$
11.	D :	$\textsc{KeyIssue}(D,H,\mathcal{H},\mathrm{A})$
12.	$H \Rightarrow \mathbb{G}_H$:	$\mathbb{Y}_{\mathbb{G}_H}, info_{\mathbb{G}_H}, \mathrm{c}_{\mathbb{G}_H}, \textsc{Sig}_{S^{\mathrm{I}}_{H,\mathcal{H}}}$
13.	D :	$\textsc{Binding}\ (D, U_r)$
14.	$H \rightarrow D_{U_r}$:	deploy_ack, $\textsc{Mac}(\mathrm{n}_{D_{U_r}} + 1)_{k^i_{D_{U_r},H}}$

Where the new symbols denote:

U_X :	user X
r :	root
D_{U_X} :	personal device of user X
$\mathbb{G}_X$:	domain's X group of devices
$info_X$:	X's type and supported operations
$\mathbb{A}_X$:	X's set of attributes
$\mathbb{Y}_X$:	X's set of predicates
H :	Home server
deploy_req, _ack :	request and acknowledgment labels
deploy :	command label
$\mathrm{secret}^X_{\mathcal{Y}}$:	secret parameter of cryptosystem X in domain $\mathcal{Y}$
$\Rightarrow$:	broadcast transmission

Figure 9: Deployment stage.

D itself (step 1). Next, the device D obtains from the root's device (i) its identity $id_{U_r,\mathcal{H}}$ in the domain $\mathcal{H}$; (ii) the IBC public key $P^{\mathrm{I}}_{H,\mathcal{H}}$ of the Home server H; (iii) and the domain counter $\mathrm{c}_{\mathbb{G}_{\mathrm{H}}}$ (step 2) that will provide freshness for broadcasts originated from H. The device D, subsequently, generates an ephemeral (i.e., valid for only a short period) pairwise key $k_{D,H}$ at random, encrypts it using H's public key $P^{\mathrm{I}}_{H,\mathcal{H}}$, and sends the resulting ciphertext to the root device D_{U_r} (step 3). Now U_r uses D_{U_r} to set D's attributes $\mathbb{A}_D$ and the predicates $\mathbb{Y}_D$ one needs to satisfy to execute operations on D (step 4). (Recall from Section 2.4 a predicate is a signing policy.) We assume communication in these first four steps is made via channels that are not only secure but adequate for the device D being deployed (e.g., wireless). As the *Deployment* protocol runs behind closed doors (i.e., at the device owner's home), transmission mechanisms like typing on a keyboard on the device or scanning QRCodes could be carefully employed to exchange data securely. (These are the mechanisms we use in our prototype, but other approaches are possible.) Note this strategy permits one to set up even bulky devices (like a fridge) without resorting to a cabled connection or requiring that the device is physically close to the Home server.

The protocol proceeds with the root device D_{U_r} requesting the deployment of the device D to the server H. To this end, D_{U_r} forwards to the home server H information received from both the new device and the root user (step 7). Our key insight in this stage is to use the root device D_{U_r} as an authentication bridge between the new device D and the Home server H. Particularly, the root device D_{U_r} encrypts and then "blindly signs"[3] the recently generated pairwise key $k_{D,H}$. Finally, D_{U_r} sends the resulting ciphertext ($\textsc{Enc}(k_{D,H})_{P^{\mathrm{I}}_{H,\mathcal{H}}}$) to H (steps 5–7). Note that although the new device relies on the root device to carry out this communication, D does not trust D_{U_r} to exchange cryptographic material and the key $k_{A,B}$ is encrypted.

Lastly (steps 8–11), the Home server H issues IBC and ABC private keys to the device D. (This issuing is secured using the recently shared key $k_{D,H}$.) H broadcasts information to all home devices—including the new one—containing their respective attribute and predicate sets (step 12). At this point, device D is granted access to the Internet, e.g., through the Home domain's Wi-Fi router. Device D then binds itself to the user U_r (step 13). Here, we assume D is a home appliance or shared device, and therefore it is bound to the root user. If the device D is instead a personal device of user U, D would be bound to U.

Root device deployment. It is clear from the above description the root device D_{U_r} cannot itself follow the protocol, as it plays a key role in the whole process. In fact, D_{U_r}—and D_{U_r} alone—needs to be deployed using a secure channel to the server. We believe this is an easy task since D_{U_r} is most likely a smartphone and can be easily connected to the server by using a cabled connection like USB. Note also that this procedure is carried out only once.

3.6 Functioning

Functioning (Figure 10), as its name suggests, governs the normal operation of devices. In the protocol, a user U requests an operation op on device B (e.g., show image from the internal camera of a smart fridge) using device A (step 1). The device A therefore forwards the request to device B (step 2), which in turn responds with a predicate Υ_{op} (step 3). The device A proves it may perform the operation by signing its request with an attribute (sub)set that satisfies the predicate (step 4). If device B successfully verifies the signature satisfies the predicate, then the requested operation is performed (step 5).

3.7 Retirement

Broadly speaking, *Retirement* is simply a special operation and the protocol (Figure 11) is thus analogous to *Functioning* (Figure 10). To run *Retirement*, a user U employs the device A to request the retirement of a device B. The retirement is performed if A's attributes satisfy the predicate. These steps are executed in the same way as other operations are in *Functioning* (Figure 10, steps 1–4). Here, we assume that B is owned by U. So, B unbinds itself from its owner U (step 2) and deletes all of its keys from both the Cloud and Home domains (step 3). Device B concludes the retirement by displaying "retired" on its screen, which plays the role of a retirement_ack (step 4). Note that a retired device can no longer be used in the home domain.

[3]We use double quotes because a *blind signature* refers to a specific cryptographic construction [11].

FUNCTIONING($user\ U$, $device\ A$, $device\ B$, $operation$ op)

1.	U :	uses A to request op over B
2.	$A \to B$:	n_A, op_req
3.	$B \to A$:	$\mathrm{n}_B, \Upsilon_{op}, \textsc{Mac}(\mathrm{n}_A)_{k^i_{A,B}}$
4.	$A \to B$:	op, $\textsc{Sig}(\mathrm{n}_B \mid \mathrm{n}_A)_{S^{\mathrm{A}}_{A,\mathcal{H}}}$
5.	B :	performs operation op
6.	$B \to A$:	op_ack, $\textsc{Mac}(\mathrm{n}_A + 1)_{k^i_{A,B}}$

Where the new symbols denote:

op_req, _ack :	request and acknowledgment labels
Υ_{op} :	operation op's predicate

Figure 10: Functioning stage.

RETIREMENT($user\ U$, $device\ A$, $device\ B$)

1.	A :	{*Requests retirement*}
2.	B :	UNBINDING(B, U)
3.	B :	deletes $S^{\mathrm{I}}_{B,\mathcal{C}}$, $S^{\mathrm{I}}_{B,\mathcal{H}}$, $S^{\mathrm{A}}_{B,\mathcal{H}}$ and $\mathbb{R}_B$
4.	B :	displays on the screen 'retired'

Where the new symbol denotes:

$\mathbb{R}_X$:	X's pairwise key ring

Figure 11: Retirement stage.

3.8 Extra Features

In this section we describe other capabilities in AoT, in particular how to handle device ownership transfer (Section 3.8.1), access revocation (Section 3.8.2), and guest access (Section 3.8.3).

3.8.1 Device Ownership Reassignment

Owner Reassignment (Figure 12) allows a user U to transfer the ownership of a device B he owns to another user V. The protocol is similar to *Retirement* (Figure 11) and the same observations there apply. Here, however, the device B does not delete its Cloud domain keys.

In the beginning, U tells the Cloud server C about the reassignment who, in turn, sends the user V a new PIN pin'_B (Figure 12, steps 1–2). The subsequent steps of the protocol are analogous to the first steps of any operation (Figure 10, steps 1–4). Next, B unbinds itself from U and deletes all of its keys from the Home domain (step 5). The protocol continues with B displaying the message 'ownership reassigned' on its screen, which acts as a reassignment_ack (step 6). Finally, U ships B to V (step 7).

REASSIGNMENT($user\ U$, $device\ A$, $device\ B$, $user\ V$)

1.	$U \to C$:	TLS(B | V)
2.	$C \to V$:	TLS(pin'_B)
3.	A :	{*Requests reassignment*}
4.	B :	UNBINDING(B, U)
5.	B :	deletes $S^{\mathrm{I}}_{B,\mathcal{H}}$, $S^{\mathrm{A}}_{B,\mathcal{H}}$ and $\mathbb{R}_B$
6.	B :	displays "ownership reassigned"
7.	$U \Rrightarrow V$:	B

Where the new symbol denotes:

pin'_X:	new PIN to grant acess to X

Figure 12: Reassignment stage.

3.8.2 Key Revocation

Revocation is not the main focus of AoT. In spite of that, as a revocation mechanism in AoT, we follow the scheme proposed by Boneh and Franklin [28] and later improved by Boldyreva *et al.* [51]. The strategy consists of associating keys with their respective expiring dates (a *timestamp*), so they become invalid after a given period of time. In this setup, a device D's public key becomes a map of its identity $id_{D,\mathcal{X}}$ concatenated with a timestamp, i.e., $P^{\mathrm{I}}_{D,\mathcal{X}} := \textsc{Map}(id_{D,\mathcal{X}} \mid \text{timestamp})$. The granularity of the timestamp can be set by the Home or Cloud servers. For instance, keys may be renewed on a daily basis. With this strategy, keys are instantly revoked as soon as their timestamps expire.

3.8.3 Interdomain (Guest) Operation

The *InterDomainKeyAgreement* protocol (Figure 13) allows a device from a foreign Home domain (i.e., a guest) to agree on keys and thus interoperate with devices from a local Home domain. The protocol is based on the work of McCullagh and Barreto [44]. It requires the two Home domains to agree on common public parameters of their Identity-Based Cryptosystems as well as their group generator points $G^{\mathrm{I}}_{\mathcal{H}_A}$ and $G^{\mathrm{I}}_{\mathcal{H}_B}$ before interdomain communication can occur. The protocol also requires that the guest device obtain the public key of the visited Home domain in an authenticated manner and that devices generate and use exclusive IBC private keys $S^{\mathrm{I}}_{X,\mathcal{I}}$. We call this whole new setup "Inter" domain.

A guest device A that wants to access operations on a device B that belongs to another domain first obtains B's public IBC parameters in B's own domain and sends an interdomain session request (steps 1–2). Device B will then use A's public IBC parameters to compute a session key (steps 3–6) and send a challenge-response back to A (step 7). A will repeat the operation to compute the session key (step 8) and then finish the challenge-response exchange (step 9). Note the protocol makes use of *implicit key authentication* [52] to verify the legitimacy of the first message (in step 2). The protocol will ultimately fail if this first message is forged. We discuss the security of this protocol further in Section 5.1.

INTERDOMAINKEYAGREEMENT($device\ A$, $device\ B$)

1.	A :	$P^{\mathrm{I}}_{B,\mathcal{I}} := \textsc{Map}(id_{B,\mathcal{H}}) \cdot G^{\mathrm{I}}_{\mathcal{H}_B} + P^{\mathrm{I}}_{\mathcal{H}_B}$
2.	$A \to B$:	$\mathrm{n}_A, \mathrm{x}_A \cdot P^{\mathrm{I}}_{B,\mathcal{I}}$, inter_session_req
3.	B :	$k_{A,B} := \hat{e}(S^{\mathrm{I}}_{B,\mathcal{I}}, \mathrm{x}_A \cdot P^{\mathrm{I}}_{B,\mathcal{I}})^{\mathrm{x}_B}$
4.	B :	$\mathrm{c}_B := 0$
5.	B :	$k^i_{A,B} := \textsc{SessionKey}(k_{A,B}, \mathrm{c}_B)$
6.	B :	$P^{\mathrm{I}}_{A,\mathcal{I}} := \textsc{Map}(id_{A,\mathcal{H}}) \cdot G^{\mathrm{I}}_{\mathcal{H}_A} + P^{\mathrm{I}}_{\mathcal{H}_A}$
7.	$B \to A$:	$\mathrm{n}_B, \mathrm{x}_B \cdot P^{\mathrm{I}}_{A,\mathcal{I}}, \textsc{Mac}(\mathrm{n}_A)_{k^i_{A,B}}$
8.	A :	{*Computes the pairwise key, analogously*}
9.	$A \to B$:	$\textsc{Mac}(\mathrm{n}_B \mid \mathrm{n}_A)_{k^i_{A,B}}$
10.	$B \to A$:	inter_session_ack, $\textsc{Mac}(\mathrm{n}_A + 1)_{k^i_{A,B}}$

Where the new symbols denote:

$\mathcal{I}$:	"Inter" domain
$G^{\mathrm{I}}_{\mathcal{Y}_X}$:	identity-based cryptosystem group generator point of X's $\mathcal{Y}$ domain
x_X :	random number generated by X
inter_session_req, _ack :	request and acknowledgment labels

Figure 13: InterDomainKeyAgreement protocol.

With the session key just established, device A can request operations on device B. To perform operations on devices of a foreign domain, a guest device must follow a protocol similar to our *Functioning* protocol (Figure 10). Because guest users have no attributes in domains they visit, domains must define an attribute "guest" which entitles guest devices to a limited set of permissions in the domain. Foreign operation requests are done in a manner similar to conventional operation requests. The exception is that the device does not have to prove possession of a certain attribute set. Instead, the requested operation is permitted if guests are allowed to perform it. That is, if the predicate to perform the operation is satisfied by the attribute “guest”. Besides, no signatures are used during this interaction as the Attribute-Based Cryptosystems of A and B do not interoperate. Instead, they make use of MACs generated using the session key to authenticate.

4. DEVELOPMENT

We now describe our AoT prototype, its software architecture (Section 4.1), its implementation (Section 4.2), and our proof-of-concept demo (Section 4.3).

Our prototype provides authentication and access control using device identities and attributes; isolated Cloud and Home domains, allowing users and manufactures to control devices independently; lightweight requirements, supporting resource-constrained embedded devices; and flexibility, being deployable on different platforms.

4.1 Architecture

Figure 14 shows the entities in our architecture. Our architecture comprises one Cloud server (Figure 14, label 1) to control the manufacturer’s Cloud domain. We assume the Cloud server has on-demand resource allocation (CPU, storage, memory, and bandwidth), typical of cloud environments. Our architecture also comprises multiple Home servers, one for each Home domain (Figure 14, label 2). Although Home servers need not scale like Cloud servers, we assume a Home server stays on and has sufficient resources to control hundreds of devices in its domain. Home servers can run on video-game consoles, desktop PCs, network gateways, or Hardware Secure Modules (HSMs).

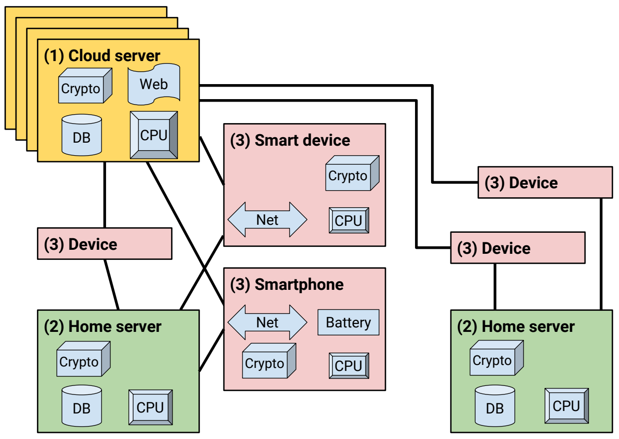

Figure 14: Entities comprising the AoT architecture.

Each device (Figure 14, label 3) connects to its Home domain and also to the manufacturer’s Cloud domain. Our architecture considers devices vary wildly (in terms of processing power, available memory, storage capacity, communication technologies, size, and weight) and may have constrained resources.

Entities in our architecture run the software stack shown in Figure 15. The implementation of the AoT protocol is the center piece of the software stack. It is built atop a cryptographic library accessed through native method calls, and communicates with the exterior world using the appropriate communication technologies and encoding mechanisms. Most entities also provide interfaces for user interaction (not shown in the software stack).

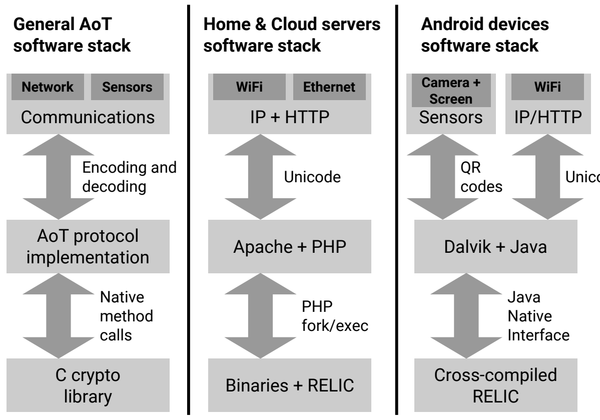

Figure 15: AoT entity software stack.

4.2 Implementation

Software stacks. Our architecture encompasses devices running heterogeneous hardware and different operating systems, each with their own specific software stack. We implemented Cloud and Home servers on LAMP (Linux, Apache, MySQL, and PHP). We also implemented AoT on smartphones as a native Android application. The realization of our software stack on these implementations is shown in Figure 15 (middle) and (right), respectively. Our Cloud and Home servers communicate with devices using the HTTP protocol. The Web servers call binaries that implement AoT’s cryptographic primitives from PHP. On Android devices, we implement the user interface using Android’s native interface, and use the Java Native Interface (JNI) to call functions on a cross-compiled cryptographic library. We note that modifications to one layer in the stack do not imply changes in other layers, e.g., we could exchange HTTP for another transport or use a different cryptographic library.

Communication. Our cryptographic code is agnostic to message encoding and transport; any two devices can exchange information using any mutually-supported encoding and protocol. For example, our prototype can exchange data using QRCodes, when deploying a fridge in a Home domain, or HTTP, when changing user attributes from a smartphone.

User interfaces. Although entities share the same underlying implementation of the AoT protocol, they run application code and user interfaces implemented using native libraries. For example, our Android applications run in Android’s Dalvik virtual machine and use the Java Native Interface (JNI) to call our low-level AoT cryptographic functions.

4.2.1 Cryptography

All entities in our architecture—Cloud servers, Home servers, and devices—share the same underlying implementation of AoT and use our extended version of the RELIC crypto library [53]. We chose RELIC because it targets resource-constrained devices and efficiently implements several curve-based cryptographic algorithms at different (e.g., 80- and 128-bit) security levels. Most primitives implemented in RELIC run on 8-bit and 16-bit embedded processors under 4 KiB of RAM. We have further improved RELIC’s performance including new architecture-dependent optimizations by means of carefully crafted assembly for the ARM architecture. The arithmetic backend makes extensive use of the wide 32-bit multiplier instruction UMLAL for implementing Karatsuba-

Comba multiplication, squaring, and Montgomery modular reduction to accelerate primitives used by AoT. Notice that targeting the 128-bit security level in resource-constrained platforms is somewhat uncommon in the research literature.

We implemented the core of our access control mechanism, ABS, on top of RELIC. To the best of our knowledge, ours is the first known open implementation of such a scheme.[4] Our ABS implementation is based on the work of Maji, Prabhakaran, and Rosulek [37], which comprises four main algorithms: master key generation, attribute generation, signature generation, and signature verification. There is also an algorithm for probabilistic signature verification. In AoT, the first two algorithms are run by Home servers, while the last two are run by devices whenever access control is executed. Our implementations rely on PBC (explained in Section 2.5) and Monotone Spam Programs (MSP). MSPs represent AoT policies as monotone boolean functions. Informally, monotone means that the boolean expression contains zero negations (*not* operators). We can work around this limitation by inverting attributes, e.g., creating attributes like "over 18" and "under 18".

The computational complexity of most ABS algorithms depends on PBC operations, more precisely, on pairings and elliptic curve scalar point multiplication. Although these operations can be executed on resource-constrained devices [22], ABS requires devices to compute a product of pairings, which increases not only compute time but also memory consumption. As a solution to this problem, we optimized RELIC to compute products of pairings simultaneously. Pairing computation can be divided in two phases: the Miller loop consisting of a square-and-multiply algorithm and the final exponentiation. When a product of pairings is computed simultaneously (a multi-pairing operation), squarings in the full extension field and the final exponentiation can be shared for all pairings, keeping a single variable accumulating partial results in the Miller loop [54]. This optimization saves one large $\mathbb{G}_T$ element to be stored and around 50% finite field multiplications per additional computed pairing in the product.

Our implementation of AoT combines the following cryptographic protocols and algorithms. (i) *Sakai-Ohgishi-Kasahara (SOK):* a non-interactive key distribution scheme allowing devices to derive symmetric keys without the need of communication. (ii) *Boneh-Franklin (BF-IBE):* an identity-based encryption scheme, adapted to employ asymmetric pairings. (iii) *Bellare-Namprempre-Nevem (vBNN-IBS):* a pairing-free identity-based signature with shorter signatures and fast verification. (iv) *Maji-Prabhakaran-Rosulek (MPR-ABS):* an attribute-based signature with attribute-privacy and collusion-resistance. (v) *keyed-Hash Message Authentication Code (HMAC):* a specific type of message authentication code (MAC) involving a cryptographic hash function, using SHA256. (vi) *Advanced Encryption Standard (AES):* the standard for symmetric encryption, using CBC mode. From now on, we refer to them as simply SOK, IBE, IBS, ABS, MAC, and AES, respectively.

Parameterization. The cryptographic protocols in AoT were implemented using pairing-friendly curves with embedding degree 12 suited to the 80- and 128-bit security level. In particular, we adopted Barreto-Naehrig [55] curves parameterized by integers $u = (2^{38} + 2^{32} + 2^5 + 1)$ at the 80-bit and $u = -(2^{62} + 2^{55} + 1)$ at the 128-bit level. Curves in this family are defined with a prime modulus $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ and group order $n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$, and support an efficient optimal pairing construction [56]. In this instantiation, elements from $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ take 40, 80, 240 bytes at the 80-bit; and 64, 128, 384 bytes at the 128-bit security level, respectively. Variable-base and fixed-base scalar multiplications were implemented using the window NAF and single-table comb methods [57], respectively. Parameters for 80-bit security address legacy systems and more constrained devices, while parameters for 128-bit security are current candidates for standardization [58] and ensure long-term security and efficiency.

[4]ABS code available at http://www.dcc.ufmg.br/~lemosmaia/aot.

4.3 Demo

We developed a demo to showcase our AoT prototype.[5] Our demo covers the entire life-cycle of a smart fridge, from pre-deployment to retirement. We demonstrate the functionalities in our architecture executing operations on multiple devices and transferring information using different communication technologies.

Our demo runs on a Intel Edison and two Android phones. On the Edison, we run both Cloud and Home servers as well as emulate a smart fridge and a smart TV. The emulated fridge allows operations such as defrosting and changing temperature, while the emulated smart TV can request and show images from other devices. Finally, our demo also has a Web interface that accesses the Cloud server to show the status of all devices in the Cloud domain.

5. EVALUATION

In this section we verify AoT's security using tool-assisted analysis. We also show that AoT supports deployment in resource-constrained embedded devices since it demands few computational resources.

5.1 AoT Security

AoT provides the following *security properties*: authentication, confidentiality, freshness, integrity, and non-repudiation. Precisely, AoT uses (i) MACs and digital signatures for authentication; (ii) ciphers for confidentiality; (iii) nonces and counters for freshness; (iv) MACs, digital signatures, and hash functions for integrity; (v) and digital signatures for non-repudiation.

We use Scyther [35]—a tool for automated verification of security protocols – to formally verify that AoT provides the aforementioned properties. Scyther analyzes protocols verifying if the protocol is prone to known attacks that could violate its security properties. We have chosen Scyther because it is efficient [36], provides novel features (e.g., multi-protocol analysis) [35], and supports different classes of attacks [35].

Scyther assumes that cryptographic primitives are ideal from a security point of view, and finds flaws arising from a protocol's message exchanges. Protocols must be described using Scyther's Security Protocol Description Language (SPDL). SPDL describes protocols as a set of *roles*. Each role specifies one of the agents in the protocol. The security properties expected from the protocol are modeled as *claims* that state that a protocol guarantees security properties. For instance, we can claim the secrecy of a term, meaning that a certain variable remains confidential throughout the protocol.

In order to verify protocol's claims, Scyther extends the strategy proposed by Song *et al.* [59]. Broadly, the tool enumerates all plausible interactions between protocol agents and generates all possible protocol executions. If a claim is violated in one of these executions, then the claim is false. If a claim is satisfied for all executions, then Scyther is able to prove the claim. A proved claim means that that property is guaranteed by the protocol, i.e., that an adversary is unable to violate that property regardless the protocol execution.

[5]Video available at http://www.dcc.ufmg.br/~lemosmaia/aot.

We used Scyther to verify all AoT protocols except *SessionKey*, *Pre-Deployment*, and *Ordering*. This is because Scyther analyzes protocols from the point of view of communication and *SessionKey* does not exchange messages, and because *Pre-Deployment* as well as *Ordering* are, by definition, protected using secure channels.

All of AoT's security properties have been successfully verified by Scyther, except the authentication in the *InterDomainKeyAgreement* protocol (Figure 13). Scyther raises a false alarm for the *InterDomainKeyAgreement* protocol because it is unable to identify the *implicit key authentication* [52] taking place to ensure the authenticity of the protocol's first message (step 2). The message in question is sent with no explicit authentication mechanism (no MAC protects it). Instead, this message is authenticated by the MACs of the subsequent messages (e.g., steps 7 and 9). These MACs, in turn, are computed using the key $k^i_{A,B}$ which the generation ultimately depends on the parameter x_A. Given $\mathrm{x}_A \cdot P^{\mathrm{I}}_{B,\mathcal{I}}$, it is considered unfeasible to derive x_A even though $P^{\mathrm{I}}_{B,\mathcal{I}}$ is public. (This is the Elliptic Curve Discrete Logarithm Problem (ECDLP) and the best general-purpose attack known against it has a fully-exponential running time for well-chosen parameters [60].) Hence, it is also considered unfeasible an adversary to derive $k^i_{A,B}$. Alternatively, the adversary could intercept A's message and launch an impersonation attack against B using a false parameter $\mathrm{x}_{A'}$. In this case, however, the session keys computed by A and B (steps 3–5 and 8, respectively) will not match, the verification of the MACs generated from them will fail, and the protocol will thus abort. In conclusion, all the aforementioned security properties hold in *InterDomainKeyAgreement* protocol as well.

Another point that might raise concern is the probabilistic ABS verification (P.ABS). Compared to its deterministic counterpart, the probabilistic approach trades determinism for speed. The security of the probabilistic ABS verification, however, is equivalent to the deterministic ABS verification [37] up to a negligible factor. The last security issue that needs to be discussed is the protection of the master keys, considering key escrow in the Cloud and Home servers. Standard techniques can be employed to protect such keys, like employing HSMs or distributing trust through Shamir's secret sharing scheme [61]. Using the scheme, domain's secret parameters can be divided into shares and assigned to different users, in a way that some or all parts of the secret can be required for reconstruction, increasing security further.

5.2 Analytical Evaluation

We analyse AoT's computational costs and communication overhead. We quantify computational costs of AoT as a function of the number of the most expensive cryptographic operations in its cryptographic primitives. In particular, we consider the number of pairings (denoted $\hat{e}$), and elliptic curve scalar multiplications (denoted p). Other operations (symmetric primitives) are executed in negligible time. We quantify communication overhead as the amount of bytes in a signature or the overhead caused by encryption techniques. Table 1 summarizes the results.

ABS is the most expensive cryptographic primitive in AoT. It is also the core of our access control mechanism (in the *Functioning* stage) and the most frequently used primitive in AoT. In later sections, we focus our analysis on ABS. The computational cost of ABS grows with the size of the span program matrix generated from the predicate. The span program matrix has dimensions $l \times t$, where l is the number of attributes and t is the number of 'and' ($\wedge$) operators in the predicate plus one.

Communication overhead for most primitives is constant. We compute the overhead considering 16-byte nonces and 1-byte labels. Communication overhead in ABS depends on the size of the span program matrix, and is around a few hundreds of KBs per signed message. We believe this overhead is still small enough to fit in one packet, or a few small packets when running over low-power communication networks that employ small frames. Network transmission delay and energy costs induced by AoT are negligible compared to CPU processing time and energy costs [24].

Primitive	**Computational Overhead** **Sender**	**Receiver**	**Communication Overhead (bytes)**
ABS [37]	$(3+2l+2lt)\hat{e}$	$(2lt+1)p+$ $(lt+t+3)\hat{e}$	$130+65l+129t$
P. ABS [37]	$(3+2l+2lt)\hat{e}$	$(2lt+t+2)p+$ $(l+2)\hat{e}$	$130+65l+129t$
IBS [62]	$1p$	$3p$	133
IBE [28]	$1\hat{e}+2p$	$1\hat{e}+1p$	129
SOK [27]	$1\hat{e}$	$1\hat{e}$	65

Table 1: Computational – number of pairings ($\hat{e}$), and point multiplications (p) – and communication overhead for AoT cryptographic primitives. For ABS, l denotes the number of attributes and t the number of 'and' operators in the predicate plus one. We compute communication overhead considering 16-byte nonces and 1-byte labels.

5.3 Experiments

We evaluate our AoT prototype on different devices, varying computational resources, as shown in Table 2. We focus on three important questions to AoT's deployment.

Resource	**G4**	**Edison**	**Due**
CPU	Arm A57	Atom	Arm M3
OS	Android 5.1	Raspbian	None
Word size	64 bits	32 bits	32 bits
Clock	2 GHz	500 MHz	84 MHz
RAM	3 GB	1 GB	96 KB
Storage	32 GB	4 GB	512 KB

Table 2: Summary of devices used in experimental evaluation.

5.3.1 Can embedded devices afford to run AoT?

We start our evaluation on our most constrained device, an Arduino Due. At the time of writing, a Due development board costs around US$ 45, and similar hardware would be even cheaper in bulk. The Due is representative of microcontrollers that could be used on low-end appliances supporting AoT.

Figure 16 shows run times for different cryptographic primitives used in AoT. We execute each algorithm to measure 100 run times and plot the quartiles as well as the 5th and 95th percentiles. We observe that all cryptographic primitives run in reasonable time.[6] ABS is the most expensive primitive and takes less than 1.6 seconds to generate signatures and less than 3.0 seconds to verify signatures for predicates on the form $A \wedge B$.

Complementary to Figure 16, Figure 17 shows run times for different ABS configurations on the Due for predicates of the form $A \wedge B$. We show results for 80- and 128-bit security levels, as for deterministic and probabilistic signature verification. We observe that run times are manageable even on the resource-constrained Due and that we can trade-off security for shorter execution times, e.g., for interactive applications that require low latency.

[6]We omit our results for symmetric primitives (AES and MAC) as they run in less than 2.5 *milli*seconds on 1K messages.

We also evaluate storage requirements for AoT on the Due (not shown). Our extended RELIC library plus our AoT implementation takes 147 KB of storage, which fits on the Due while leaving significant storage space for applications.

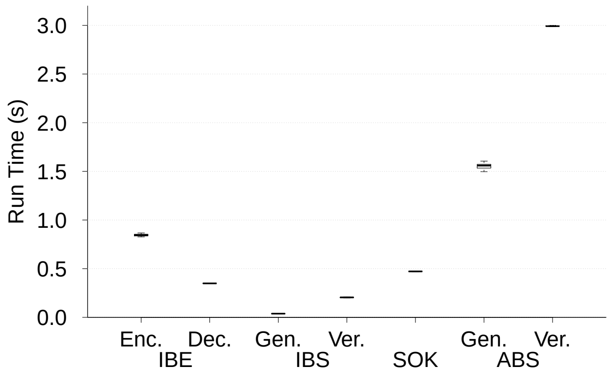

Figure 16: Run times for asymmetric cryptographic primitives on the Due. We abbreviate *enc*ryption, *dec*ryption, signature *gen*eration, and signature *ver*ification. Run times for IBE and ABS use 1KB messages and the $A \wedge B$ predicate, respectively.

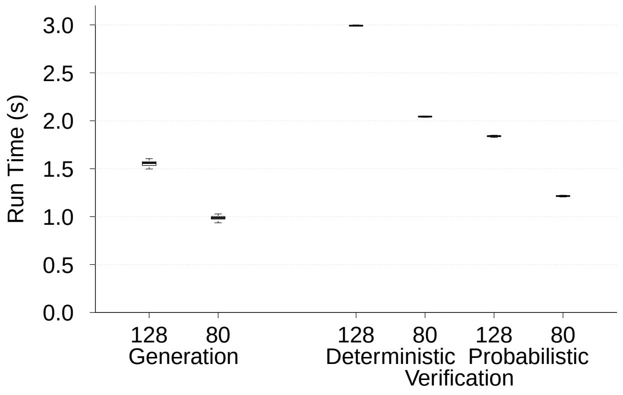

Figure 17: Run times of ABS for different security levels (80 and 128 bits) and configurations (deterministic or probabilistic verification) on the Due for predicate $A \wedge B$.

5.3.2 *Does AoT scale to realistic IoT scenarios?*

As shown before, the AoT's most expensive cryptographic primitive is ABS. ABS is also the most frequently used primitive in AoT, once it is called whenever access control is performed. In the previous section, we used a representative predicate of the form $A \wedge B$ when computing run times and memory requirements. In this section, we evaluate run times for various predicates to check whether AoT scales to realistic IoT scenarios.

Table 3 shows run times for scalar point multiplication in $\mathbb{G}_1$ and pairing computation on our platforms. These operations are the building blocks of cryptographic primitives used in AoT (Table 1).

Figure 18 evaluates ABS signature verification run times for different predicate structures. We vary the number of attributes on the x axis and show different curves varying the number of 'and' operators (l and t in Table 1, respectively). Figure 18 shows ABS verification run times estimated analytically. Curves in Figure 18 are plotted combining operation run times from Table 3 with analytical costs in Table 1. In Figure 18, we observe that the Due can verify signatures for complex predicates with up to nine attributes and only 'and' operators in less than a minute. For more usual predicates including between 1 and 3 attributes, signature verification is estimated in less than 6 seconds.

OPERATION	G4	Edison	Due
Point multiplication	1.1 ± 0.2	10.9 ± 0.2	74.5 ± 1.3
Pairing	4.1 ± 0.7	88.4 ± 0.3	355.1 ± 1.0

Table 3: Expensive cryptographic operations run times (ms).

Figure 19 shows experimental ABS verification run times and memory utilization averaged over 30 runs (coefficients of variation are below 5%, not shown). We consider worst-case predicates that contain only 'and' operators. We plot four curves for deterministic and probabilistic verification at the 80- and 128-bit security levels.

By comparing Figures 18 and 19a, we observe that analytical run times significantly overestimate our implementation run times. There are two main reasons for this. First, we optimized our code to compute multi-pairings simultaneously (Section 4.2). Second, analytical costs consider that the $t \times l$ coefficient matrix in a predicate's MSP does not contain any zeros. In practice, we find coefficient matrices are sparse, which significantly reduces the amount of executed operations. For example, while the analytical run time for verifying a signature for a predicate of the form $A \wedge B$ is 4.5s (Figure 18), our code takes only 3.0s (Figures 16 and 19a).

Figure 19a also shows that we can significantly reduce the run time of ABS in AoT by either reducing the security level (e.g., to 80-bit) or by using probabilistic verification. In particular, probabilistic verification at the 80-bit level enables our implementation to verify signatures in around 1.2s, which should be enough for most interactive applications. Remind also that predicates with 'or' operators require less computation and further reduce verification run times.

Finally, Figure 19b shows memory use for ABS. We measure maximum memory utilization as the maximum stack size reached by each ABS function during their execution (RELIC only allocates memory on the stack). As expected, ABS is well-suited for resource-constrained devices as it requires at most 19KB of RAM memory for a predicate with nine attributes and eight 'and' operators. Besides, note further improvements to the implementation or the algorithms could lead to even lower requirements.

5.3.3 *How does AoT perform on other architectures?*

We now look at how AoT performs on more resourceful architectures. As AoT's memory and storage requirements are negligible compared to the amount of RAM and storage available on the Edison and the G4, we focus on execution time.

Figure 20 shows cryptographic primitive run times on the Edison and the G4. These devices have processors significantly more powerful than the Due, which results in significantly higher performance (note the different ranges on the y axes). Figure 21 shows analytical and experimental run times on the Edison for varying predicate structures (results for the G4, not shown, are qualitatively similar). It shows that our optimizations carry over to different architectures and provide significant run time reduction over the analytical estimates. Finally, Figure 21 shows that more powerful platforms can verify complex predicates with various attributes in less than 2s.

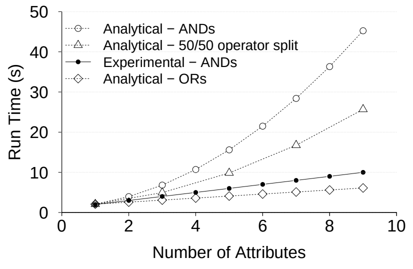

Figure 18: Experimental and analytical ABS verification run times on the Due for varying predicate structures. Plotted combining Tables 1 and 3.

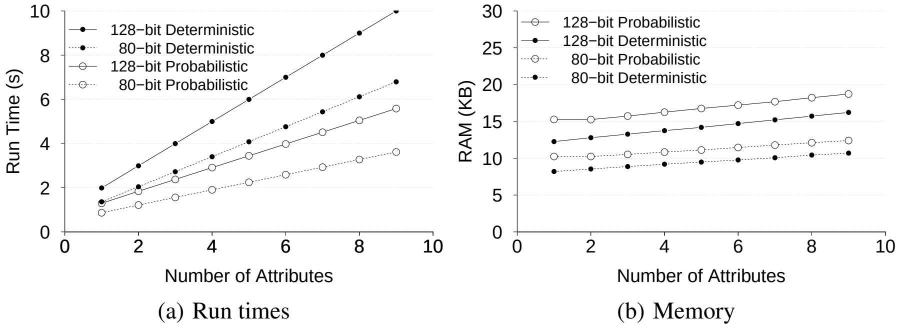

Figure 19: Experimental ABS verification run times and memory requirements on the Due for varying predicate sizes and 100% AND operators (average over 30 runs).

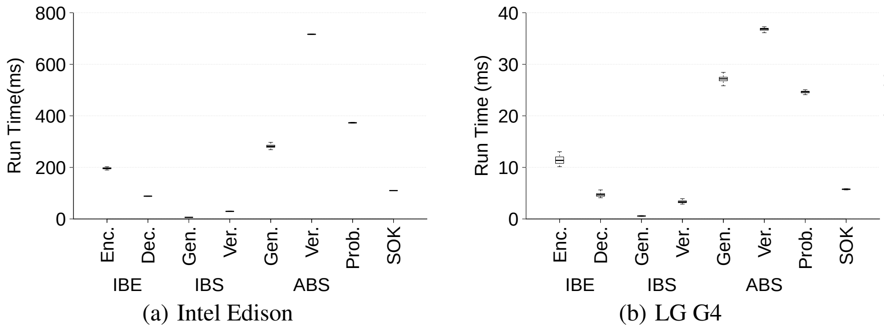

Figure 20: Asymmetric cryptographic primitives run times. IBE and ABS run times for 1KB messages and the $A \wedge B$ predicate, respectively.

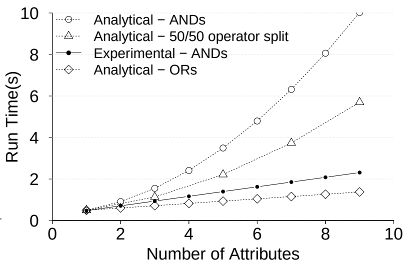

Figure 21: Experimental and analytical ABS verification run times on the Intel Edison for varying predicate structures (average over 30 runs).

6. RELATED WORK

Albeit authentication differs in concept and purpose from access control, they are closely related security subjects. Notably, the works on security for resource-constrained devices typically make use of authentication to establish an access control mechanism. In what follows, we briefly describe previous work on (i) security for constrained devices, (ii) user to device authentication in IoT, and (iii) authentication and access control schemes for IoT devices.

Security for constrained devices. Before the advent of IoT, much work has been proposed for securing MANETS and sensor networks in general. The studies for MANETs (e.g., [63–67]) are not applicable to IoT because they assumed Personal Digital Assistants (PDAs), which have more computational resources than many IoT devices. Traditional PKC-based solutions are such an example. The works tailored to sensor networks (e.g., [12–20, 22, 23, 25]) are usually ill-suited to IoT, too. This is because they often make assumptions that do not apply to IoT and thus the proposed solutions cannot be applied as-is. For instance, while sensor devices often run the same application and are owned by a single entity in sensor networks, devices in IoT may execute different applications and report to more than one authority. AoT brings authentication and access control to IoT, addressing IoT-specific requirements and constraints.

IoT user-to-device authentication. There are different types of user-to-device authentication schemes for IoT [68]. First, a user can authenticate himself onto a device by entering a previously shared secret [69–73]. This type of authentication is the most common on smart devices [74] and its security and usability are heavily affected by the chosen secret; complex secrets usually increase security but impair usability [75]. Second, users can authenticate by using a *physical authenticator* they have, e.g., a cryptographic token or smart card [76]. This mechanism is more commonly used as as a way of providing a second authentication factor. Third, users can authenticate through their own *biological features* like fingerprints, voice, or face (e.g., [77–79]). A drawback of this approach is that it requires biometric features to be stored on the device. This raises privacy concerns as compromising one's authenticator means stealing private user information, such as his fingerprint. Last, one can take advantage of *behavioral patterns* to authenticate users [78,80]. In this case, the user's behavior when using the device is used to create a pattern, this pattern is later used to authenticate the user. When the device is being used, if the usage matches the expected behavior, the user is authenticated. AoT provides device-to-device authentication in IoT deployments and is complementary to (and can benefit from) these user-to-device authentication solutions.

Authentication and access control for IoT. Works in the literature have addressed authentication and access control under the perspective of IoT using different strategies, e.g., by managing authentication policies, addressing data verification, applying IBC and specific technologies to device pairing, defining standards for security, among others. Non-extensive examples are described as follows to provide an overview of the state-of-the-art.

Recently, Liang *et al.* [9] presented Safe Internet oF Things (SIFT), a safety-centric programming platform for connected devices in IoT environments. In SIFT, users express high-level intents in declarative IoT applications, and the scheme then decides which data and operations should be combined to meet the user needs. To ensure safety and compliance, the scheme verifies whether conflicts or policy violations can occur within or between applications, which may be regarded as an access control mechanism. Strictly speaking, however, the work focuses more on safety of users.

Oliveira *et al.* [24] focused on the authentication of communication from a device to multiple users. The proposal named Secure Tiny Web Services (Secure-TWS) investigates the resource overheads for digital signatures, notably, for the Elliptic Curve Digital Algorithm (ECDSA) and the Boneh-Lynn-Shacham (BLS) short signature schemes.

Gisdakis *et al.* [7] study the impact of malicious devices on data gathering. They address the limitations of authentication and access control to tackle the problem of pollution on the measured data. The authors end up with a novel data verification framework called SHIELD.

Mora-Afonso *et al.* [81] proposed a scheme to secure communication between domestic devices. Authors employ IBC and Wi-Fi as well as Bluetooth and NFC to authenticate message exchange between devices. NFC is used during *Bluetooth pairing* and key distribution. Authors also presented prototypes of their solution for various devices.

Markmann *et al.* [8] came up with an end-to-end authentication scheme to deploy a federation of gateways in IoT subnetworks. The work is preliminary, also based on IBC, and brings in insightful ideas to the field.

Finally, Yavuz developed a cryptographic scheme well-suited to IoT devices called Efficient and Tiny Authentication (ETA) [5]. ETA is similar to AoT in that it supports authentication and access control, but it does not consider the entire live-cycle of IoT devices and relies on resource-rich devices to perform expensive operations.

7. CONCLUSION

As IoT becomes ubiquitous, exchanging private and sensitive information as well as performing automated actions on the environment, it is imperative that we authenticate communication between IoT devices and establish control access to their functionality. We proposed AoT, a suite of protocols that provide authentication and access control during the entire life-cycle of IoT devices. AoT provides strong authentication and flexible access control by combining state-of-the-art cryptographic primitives in novel ways. AoT addresses challenges that arise in real-world IoT deployments like bootstrapping secure communications, participation in multiple authentication domains, or running code on low-cost resource-constrained IoT devices.

We evaluated AoT analytically and experimentally. Our analytical results show that worst-case CPU and communication overheads imposed by AoT are manageable even in resource-constrained IoT devices. We also checked AoT security using tool-assisted verification. Our experimental evaluation using our AoT prototype considers multiple performance metrics and multiple hardware platforms, showing that AoT runs on low-cost IoT devices and imposes negligible overhead on powerful devices like current smartphones.

8. ACKNOWLEDGMENTS

We would like to thank Andre Graziani, Carlos Toneto, Frederico Sampaio, Jongsup Baek, Leandro Terra, Lucas Grossi, Pablo Marcondes, Paulo Barreto, Ronaldo Resende, Luiz Felipe Saggioro, Seungyoon Song, Timbere Nogueira, Vitor Paisante for fruitful discussions and feedback. This work was partially funded by LG Electronics, CNPq, CAPES, and FAPEMIG.

9. REFERENCES

[1] Kevin Ashton. That 'Internet of Things' Thing. *RFiD Journal*, 22:97–114, 2009.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[3] Mark Luk, Adrian Perrig, and Bram Whillock. Seven Cardinal Properties of Sensor Network Broadcast Authentication. In *Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2006.

[4] Sungmin Hong, Daeyoung Kim, Minkeun Ha, Sungho Bae, Sang Jun Park, Wooyoung Jung, and Jae-Eon Kim. SNAIL: an IP-Based Wireless Sensor Network Approach to the Internet of Things. *Wireless Communications*, 17(6):34–42, 2010.

[5] Attila Altay Yavuz. ETA: Efficient and Tiny and Authentication for Heterogeneous Wireless Systems. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2013.

[6] Salmin Sultana, Daniele Midi, and Elisa Bertino. Kinesis: a Security Incident Response and Prevention System for Wireless Sensor Networks. In *Conference on Embedded Networked Sensor Systems (SenSys)*, 2014.

[7] Stylianos Gisdakis, Thanassis Giannetsos, and Panos Papadimitratos. SHIELD: a Data Verification Framework for Participatory Sensing Systems. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2015.

[8] Tobias Markmann, Thomas C Schmidt, and Matthias Wählisch. Federated End-to-End Authentication for the Constrained Internet of Things Using IBC and ECC. In *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[9] Chieh-Jan Mike Liang, Börje F. Karlsson, Nicholas D. Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. SIFT: Building an Internet of Safe Things. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2015.

[10] Fernando A. Teixeira, Gustavo V. Machado, Fernando M. Q. Pereira, Hao Chi Wong, José M. S. Nogueira, and Leonardo B. Oliveira. SIoT: Securing the Internet of Things Through Distributed System Analysis. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2015.

[11] Douglas Stinson. *Cryptography: Theory and Practice*. CRC/C&H, 2002.

[12] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, 2002.

[13] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks. In *Conference on Computer and Communications Security (CCS)*, 2003.

[14] R. Di Pietro, L. V. Mancini, and A. Mei. Random Key-Assignment for Secure Wireless Sensor Networks. In *Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 62–71, 2003.

[15] Seyit A. Çamtepe and Bülent Yener. Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks. In *European Symposium on Research in Computer Security (ESORICS)*, 2004.

[16] Ronald J. Watro, Derrick Kong, Sue fen Cuti, Charles Gardiner, Charles Lynn, and Peter Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2004.

[17] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2004.

[18] David J. Malan, Matt Welsh, and Michael D. Smith. A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography. In *Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004.

[19] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks. *Transactions on Information and System Security (TISSEC)*, 2005.

[20] Donggang Liu, Peng Ning, and Rongfang Li. Establishing Pairwise Keys in Distributed Sensor Networks. *Transactions on Information and System Security (TISSEC)*, 2005.

[21] Leonardo B. Oliveira, Adrian Ferreira, Marco A. Vilaça, Hao Chi Wong, Marshall Bern, Ricardo Dahab, and Antonio A. F. Loureiro. SecLEACH– On the Security of Clustered Sensor Networks. *Signal Process.*, 87(12):2882–2895, 2007.

[22] Leonardo B. Oliveira, Michael Scott, Julio Lopez, and Ricardo Dahab. TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. In *International Conference on Networked Sensing Systems (INSS)*, 2008.

[23] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. On the Application of Pairing Based Cryptography to Wireless Sensor Networks. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2009.

[24] Leonardo B. Oliveira, Aman Kansal, Bodhi Priyantha, Michel Goraczko, and Feng Zhao. Secure-TWS: Authenticating Node to Multi-user Communication in Shared Sensor Networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[25] Harsh Kupwade Patil and Stephen A Szygenda. *Security for Wireless Sensor Networks Using Identity-Based Cryptography*. CRC Press, 2012.

[26] Adi Shamir. Identity-based Cryptosystems and Signature Schemes. In *International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 1984.

[27] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems Based on Pairing. In *Symposium on Cryptography and Information Security (SCIS)*, 2000.

[28] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 2001.

[29] Clifford Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *International Conference on Cryptography and Coding (IMACC)*, 2001.

[30] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Conference on Computer and Communications Security (CCS)*, 2006.

[31] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy Attribute-based Encryption. In *Symposium on Security and Privacy (S&P)*, 2007.

[32] Eric Yuan and Jin Tong. Attributed Based Access Control (ABAC) for Web Services. In *International Conference on Web Services (ICWS)*, 2005.

[33] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *NIST Special Publication*, 800:162, 2013.

[34] Sanjit Chatterjee and Palash Sarkar. *Identity-Based Encryption*. Springer Publishing Company, Incorporated, 2011.

[35] Cas Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Conference on Computer Aided Verification (CAV)*, 2008.

[36] Cas JF Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing State Spaces in Automatic Security Protocol Analysis. *Formal to Practical Security*, 5458:70–94, 2009.

[37] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance. *IACR Cryptology ePrint Archive*, 2008:328, 2008.

[38] Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie, and Kui Ren. Attribute-based Signature and its Applications. In *Symposium on Information, Computer and Communications Security (ASIACCS)*, 2010.

[39] Matt Bishop. *Computer security: art and science*, volume 200. Addison-Wesley, 2012.

[40] A. Menezes, T. Okamoto, and St Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. *Transactions on Information Theory*, 39(5):1639–1646, 1993.

[41] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[42] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–276, 2004.

[43] Jenny Torres, Michele Nogueira, and Guy Pujolle. Identity-Based Cryptography: Applications, Vulnerabilities and Future Directions. In *IT Policy and Ethics: Concepts, Methodologies, Tools, and Applications*, pages 430–450. IGI Global, 2013.

[44] Noel McCullagh and Paulo S. L. M. Barreto. A New Two-party Identity-based Authenticated Key Agreement. In *International Conference on Topics in Cryptology (CT-RSA)*, 2005.

[45] Amit Sahai and Brent Waters. Fuzzy Identity-based Encryption. In *International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2005.

[46] Huai-Xi Wang, Yan Zhu, Rong-Quan Feng, and Stephen S Yau. Attribute-Based Signature with Policy-and-Endorsement Mechanism. *Journal of Computer Science and Technology*, 25(6):1293–1304, 2010.

[47] Antoine Joux. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *International Symposium on Algorithmic Number Theory (ANTS)*, 2002.

[48] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *Symposium on Foundations of Computer Science (FOCS)*, 2007.

[49] Steven D. Galbraith. *Mathematics of Public Key*

Cryptography. Cambridge University Press, 2012.

[50] Adrian Perrig, Robert Szewczyk, Victor Wen, David E. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Netowrks. In *International Conference on Mobile Computing and Networking (MobiCom)*, 2001.

[51] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based Encryption with Efficient Revocation. In *Conference on Computer and Communications Security (CCS)*, 2008.

[52] Henk van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*. Springer US, 2011.

[53] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[54] Michael Scott. Computing the Tate Pairing. In *International Conference on Topics in Cryptology (CT-RSA)*, 2005.

[55] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography (SAC)*, 2005.

[56] Frederik Vercauteren. Optimal Pairings. *Transactions on Information Theory*, 56(1):455–461, 2010.

[57] Chae Hoon Lim and Pil Joong Lee. More Flexible Exponentiation with Precomputation. In *International Cryptology Conference on Advances in Cryptology (CRYPTO)*, 1994.

[58] A. Kato, M. Scott, T. Kobayashi, and Y. Kawahara. Barreto-Naehrig Curves. IETF draft available at https://tools.ietf.org/html/draft-kasamatsu-bncurves, 2016.

[59] Xiaodong Dawn Song. *An Automatic Approach for Building Secure Systems*. PhD thesis, University of California at Berkeley, 2002.

[60] Steven D. Galbraith and Pierrick Gaudry. Recent Progress on the Elliptic Curve Discrete Logarithm Problem. *Designs, Codes and Cryptography*, 78(1):51–72, 2016.

[61] Adi Shamir. How to Share a Secret. *Communications ACM*, 22(11):612–613, 1979.

[62] Xuefei Cao, Weidong Kou, Lanjun Dang, and Bin Zhao. IMBAS: Identity-based Multi-user Broadcast Authentication in Wireless Sensor Networks. *Computer Communications*, 31(4):659 – 667, 2008.

[63] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *IEEE Network*, 13(6):24–30, 1999.

[64] S. Capkun, L. Buttyan, and J. P. Hubaux. Self-Organized Public-Key Management for Mobile Ad Hoc Networks. *Transactions on Mobile Computing*, 2(1):17, 2003.

[65] Lakshmi Venkatraman and Dharma P. Agrawal. A novel authentication scheme for ad hoc networks. In *Wireless Communications and Networking Conference (WCNC)*, 2002.

[66] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: a Secure On-demand Routing Protocol for Ad Hoc Networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[67] Yongguang Zhang and Wenke Lee. Intrusion Detection in Wireless Ad-hoc Networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[68] Ian Timothy Fischer, Cynthia Kuo, Ling Huang, and Mario Frank. Smartphones: Not Smart Enough? In *Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM)*, 2012.

[69] Ian Jermyn, Alain Mayer, Fabian Monrose, Michael K. Reiter, and Aviel D. Rubin. The Design and Analysis of Graphical Passwords. In *USENIX Security Symposium (Security)*, 1999.

[70] Volker Roth, Kai Richter, and Rene Freidinger. A PIN-entry Method Resilient Against Shoulder Surfing. In *Conference on Computer and Communications Security (CCS)*, 2004.

[71] S M Taiabul Haque, Matthew Wright, and Shannon Scielzo. Passwords and Interfaces: Towards Creating Stronger Passwords by Using Mobile Phone Handsets. In *Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM)*, 2013.

[72] Yimin Chen, Jingchao Sun, Rui Zhang, and Yanchao Zhang. Your Song Your Way: Rhythm-based Two-factor Authentication for Multi-touch Mobile Devices. In *Conference on Computer Communications (INFOCOM)*, 2015.

[73] L. Cotta, A. L. Fernandes, L. T. C. Melo, L. F. Z. Saggioro, F. Martins, A. L. M. Neto, A. A. F. Loureiro, Í Cunha, and L. B. Oliveira. Nomadikey: User authentication for smart devices based on nomadic keys. In *International Conference on Communications (ICC)*, 2016.

[74] Serge Egelman, Sakshi Jain, Rebecca S. Portnoff, Kerwell Liao, Sunny Consolvo, and David Wagner. Are You Ready to Lock? In *Conference on Computer and Communications Security (CCS)*, 2014.

[75] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password Strength: An Empirical Analysis. In *Conference on Computer Communications (INFOCOM)*, 2010.

[76] Hristo Bojinov and Dan Boneh. Mobile Token-based Authentication on a Budget. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2011.

[77] Jiayang Liu, Zhen Wang, Lin Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications. In *International Conference on Pervasive Computing and Communications (PerCom)*, 2009.

[78] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch Me Once and I Know It's You!: Implicit Authentication Based on Touch Screen Patterns. In *Conference on Human Factors in Computing Systems (CHI)*, 2012.

[79] Masoud Rostami, Ari Juels, and Farinaz Koushanfar. Heart-to-Heart (H2H): Authentication for Implanted Medical Devices. In *Conference on Computer and Communications Security (CCS)*, 2013.

[80] Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow. Implicit Authentication for Mobile Devices. In *Conference on Hot Topics in Security (HotSec)*, 2009.

[81] V. Mora-Afonso, P. Caballero-Gil, and J. Molina-Gil. Strong Authentication on Smart Wireless Devices. In *International Conference on Future Generation Communication Technology (FGCT)*, 2013.