Trustless IoT: A Logic-Driven Architecture for IoT Hubs

Soumya Basu and Emin Gün Sirer

Department of Computer Science, Cornell University
{soumya, egs}@cs.cornell.edu

Abstract

The proliferation of smart devices has led to a de-facto IoT architecture where devices are controlled by cloud operators. This, in turn, leads to a central point of failure where a hacked hub can lead to the failure of the entire system. In this paper, we outline OrbanHub[1], an alternate IoT architecture which rules out Byzantine behavior by centralized IoT controllers. OrbanHub works the same way as most IoT hubs, but instead of issuing commands to devices to perform their operations, OrbanHub issues *proof-carrying statements* that devices verify. To ensure that the hub cannot reissue valid commands, OrbanHub leverages hashchains to prove freshness. We demonstrate that, through the two techniques, OrbanHub cannot force devices to execute commands that are not authorized by the user's control policies and provide a feasibility study of the architecture.

1 Introduction

The Internet of Things (IoT) has been gaining prominence in recent years. Smart devices, ranging from home thermostats [19] and carbon monoxide detectors [18] to embedded devices for cars and buildings [23], are becoming commonplace. This trend has led to intense competition among providers to become the centralized controller hub for these devices [11, 12, 14]. This trend, however, is disconcerting because a de-facto architecture for IoT that's centrally controlled by a few large, centralized companies is inherently insecure and has the potential to place entire nations at risk.

The IoT hubs of today [1, 10] are opaque and typically controlled by a centralized operator. The most popular IoT hubs in production today, such as Google HomeAssistant, Alexa and Facebook Portal, are architected in such a way that any vulnerability can compromise a large number of IoT systems. The fundamental problem is that hubs are fully trusted by the devices they control, which makes them lucrative targets for attackers.

[1]Orban is the man who conquered the last remaining remnants of the Byzantine empire.

The consequences of such a centralized design are starting to be felt by users of internet-connected devices. We have seen IoT devices being remotely compromised by malicious attackers to horrifying effect. For example, Nest cameras have been compromised and used to threaten parents about kidnapping their children [20]. Tesla cars have also been remotely accessed by a team of security researchers as well [13]. These kinds of attacks can cause severe economic damage, such as failed climate control spoiling crops.

This paper presents an alternate architecture, OrbanHub, where hubs are untrusted by the devices they control. Similar to other architectures, OrbanHub stores and manages user policies related to their smart devices. However, in other designs, when it is time for some device to perform an action, the hub unilaterally issues commands to the corresponding devices. OrbanHub requires hubs to first produce a *proof-carrying statement* (PCS) where the hub must prove that the command that it wishes to issue is consistent with policies set by the user. To verify that the PCS is well formed, each device is augmented with a minimal proof checker to check the work from the hub and ensure that the hub is applying the user's policies correctly. The proof checker is extremely lightweight as checking a proof is vastly simpler than generating a proof, which enables these proof checkers to be implemented with the resources available on internet-enabled smart devices. PCSes ensure that the hub can never compromise safety and issue a directive that conflicts with user policies.

A second class of hub violations are liveness violations, where the hub fails to take an action dictated by a user policy and thus violates the user's intent through inaction. For instance, failing to turn on a cooling unit can lead to the loss of perishable products and cause great economic damange. To prevent this class of misbehavior, OrbanHub uses hashchain commitments in the PCS so the hub can prove that commands are issued in a timely manner. One of the benefits of the OrbanHub architecture is that the user can compensate for a liveness violation with multiple competing hubs where the failure of one hub can be compensated with another.

OrbanHub points to a new design for IoT hubs, one

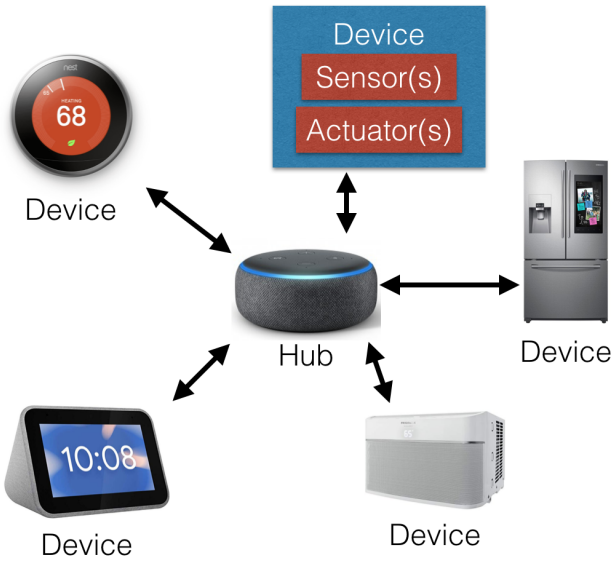

Figure 1: Model of an IoT system. A hub serves as the central controller and the devices it controls. Devices consist of zero or more actuators and zero or more sensors.

where users do not need to trust the hub to be correctly implemented. As a result, OrbanHub simultaneously enables a new generation of IoT systems that are more resilient and grants centralized providers the flexibility to implement their hubs however they want. This allows for innovation by the centralized hub providers while protecting users against errant hubs, and limiting the economic impact of hub compromises.

2 Problem Definition

We first describe the high level architecture of an IoT system and then enumerate the properties the hub needs to satisfy in OrbanHub.

Model An Internet of Things system has a few key pieces in its model. It consists of a *hub* that controls the IoT platform and orchestrates coordination between components. The hub's implementation details are not considered in the model, including whether or not the hub is controlled by a cloud service provider. Each component is called a *device* which consists of a set of *sensors* and a set of *actuators*, where either of the two sets can be empty. Devices communicate with the hub in order to accomplish their tasks. Figure 1 shows a high level overview of this model.

Alternate architectures that are more decentralized and peer to peer have been suggested by others [2], but the de-facto emerging architecture is centrally controlled. A fundamental reason for this is that devices are issued by separate manufacturers and have limited input/output mechanisms. This makes it difficult to maintain many connections, which is a prequisite for a peer to peer architecture.

Users specify high level *policies* that they wish the system to enforce. An example of such a policy is "the temperature in the living room should be between 65 and 70 degrees Fahrenheight between 8 AM and 10 PM every day". The hub enforces these policies by issuing *commands* for devices to execute. In the previous example, if a thermostat read "71 degrees" and the clock said it was 1 PM in the afternoon, then the air conditioning unit would be issued a command to turn on by the hub. These commands are issued, ideally, to enforce the policies that the user has specified.

Properties An IoT platform must provide two kinds of guarantees: safety and liveness.

Safety. No command shall be executed by a device unless it is authorized by a corresponding policy. For example, a Byzantine hub, for whatever reason, must not be able to allow a heater and air conditioner to work in opposition at the same time. In particular, a malicious hub must not cause an unauthorized command to be executed.

Liveness. If the policy calls for an action, then the system must eventually execute that action. For instance, if a warehouse must be kept within a certain temperature range, then the heating and cooling units must kick on when temperature at the warehouse is outside its specified range.

Device Requirements Devices cannot be expected to have extensive resources in order to be part of the network. Luckily, the emerging embedded hardware on systems on chips contain significant resources. For instance, an ESP-8266 system on a chip are 32-bits, have 128kB of memory, 4MB of storage, are Wi-Fi enabled and only cost $6.50 [8]. These new embedded hardware systems on a chip are powering everything that is internet-connected including the next generation IoT devices. This gives designers and architects much more flexibility in what requirements can be placed on IoT devices for the next generation IoT architectures.

3 Design

Our design revolves around two core ideas: proof carrying statements for safety and a hashchain for liveness. We start by describing proof carrying statements and how the execution flow works in OrbanHub. Then, we discuss how OrbanHub can use hashchains in order to prove that no messages have been dropped.

3.1 Proof-Carrying Statements

The basic unit of communication that OrbanHub uses is a proof-carrying statement (PCS). Such a statement carries a

logical deduction that given the current policy, the trust assumptions made by the owner, and the current inputs from the varying devices, the requested action is a logical consequence of the policy. Devices then check that the PCS is well-formed, which implies that the requested action is indeed one that complies with user-defined policy, and for freshness, implying that the requested action is operating on the latest state. We first focus on what it means for a PCS to be well-formed and then describe some strategies to enforce freshness in Section 3.2.

We adopt an authorization logic [25], based loosely on NexusOS's authorization logic, for expressing the proofs that are sent by hubs and checked by devices. This logic system supports the notion of reasoning within the worldview of a given principal. In particular, a broken or compromised device that issues conflicting statements, can only affect logical deductions that depend on its statements. Put another way, a device cannot corrupt the logical reasoning infrastructure and cause consequences beyond that with which it has been trusted.

Each device has its own worldview and the veracity of any statements are evaluated in its particular worldview. A device has the ability to state its beliefs about the world, e.g. a thermostat can say "The temperature at 2 PM is 67 degrees". This allows a device to update its own worldview based on its own sensory information.

Devices must additionally be able to update the worldview of other devices as well in order to communicate sensory information across the system. To enable this, devices can sign their own beliefs about the world and then transmit them to the hub. Once the hub transmits the signed statement to another device, that device then gets to choose whether or not to accept this statement based on its trust assumptions. For example, if the air conditioning unit receives a statement from the thermostat saying "The temperature at 2 PM is 67 degrees", then it should update its worldview to include that statement. However, if the same statement came from a toaster, then the air conditioning unit should ignore it as it is not a trusted source on temperature related statements.

Proof-carrying statements are issued based on sensor data. A hub's central task is to take the sensor data and figure out which commands to send to which devices in accordance with user policy. Current hubs satisfy this requirement, but OrbanHub must do the above and provide a deduction proof as well. To illustrate a deduction proof, we first establish some axioms and then use them to show a deduction proof in Figure 2. In this proof, the thermostat and clock gives us our basic facts ($\langle\, b, \rho \,\rangle$ and $\langle\, c, \phi \,\rangle$). Since the user trusts the thermostat regarding temperature and the clock regarding time, we see that we can make the inferences $\langle\, c, \phi \,\rangle \rightarrow \langle\, c, \sigma \,\rangle$ and $\langle\, b, \rho \,\rangle \rightarrow \langle\, b, \sigma \,\rangle$ to conclude that $\langle\, b \wedge c, \sigma \,\rangle$. And due to the policy, $\langle\, b \wedge c, \sigma \,\rangle \rightarrow \langle\, a, \sigma \,\rangle$ so a must be true and the heater should turn on.

3.2 Inter-Device Communication

Proof-carrying statements (PCSes) largely focus on the safety of the system and do not address liveness. Thus, even with PCSes, a hub could selectively drop updates and issue valid-looking commands based on stale state. Such misbehavior may violate the user policy even though they contain proofs. In this section, we describe how to augment proof carrying statements in order to resolve the liveness problem through the use of hashchains.

Hashchains were used in prior systems [24] for freshness and we adapt this approach for OrbanHub. A hashchain is a totally ordered log with a cryptographic digest to commit to every element in the chain as well their particular order. Once an element is inserted into a hashchain and the digest is sent, the commitment can only be updated to include new elements and no old element can be deleted. Most crucially, the cryptographic digest of the hashchain can be updated without keeping the full hashchain in storage.

We use the hashchain in order to detect when the hub is dropping statements from devices. Each device keeps a local commitment of the previous hashchain consisting of all of their old statements. When a device has a new statement to

$$\frac{\langle\, b \wedge c, \sigma \,\rangle}{\langle\, a, \sigma \,\rangle}\ \text{TempPolicy}$$

$$\frac{\langle\, c, \phi \,\rangle}{\langle\, c, \sigma \,\rangle}\ \text{ClockPolicy}$$

$$\frac{\langle\, b, \rho \,\rangle}{\langle\, b, \sigma \,\rangle}\ \text{ThermPolicy}$$

$$\text{TempPolicy}\ \frac{\dfrac{\langle\, b, \rho \,\rangle}{\langle\, b, \sigma \,\rangle}\ \text{ThermPolicy} \qquad \dfrac{\langle\, c, \phi \,\rangle}{\langle\, c, \sigma \,\rangle}\ \text{ClockPolicy}}{\langle\, a, \sigma \,\rangle}$$

Figure 2: This is a sample deduction proof. Let the user policy be "If the time is between 10 PM and midnight and temperature is < 70 degrees, turn on heat". We let $a =$ `Turn on heat`, $b =$ `It is 68 degrees`, and $c =$ `It is 11 PM`. Additionally, we assign σ to be the worldview of the user, ρ to be the worldview of the thermostat and ϕ to be the worldview of the clock. $\langle\, a, \sigma \,\rangle$ means that statement a is true in worldview σ. `TempPolicy` encodes the user policy, where if $b \wedge c$ are true in σ, then the heat should turn on, i.e. a is true. `ClockPolicy` and `ThermPolicy` both allow these devices to speak for the user, but only for specific categories of statements: the clock for time-related statements and the thermostat for temperature related statements. The final proof just shows how we get from the raw device statements and apply the user policy to prove that a is true in the worldview of the user and the heat should turn on.

make, it appends the statement to the end of the hashchain and transmits it to the hub. When a device is processing a statement from another device due to the user policy, it checks to make sure that the new statement is a valid extension of the hashchain it previously had from the device. Devices can elect to send hashchain updates periodically to ensure that liveness failures by the hub will eventually be detected by the devices.

While hashchains provide a powerful abstraction for dealing with stale updates and guarantee liveness, there are some unique challenges and opportunities present in OrbanHub. In our current design, each device is responsible for maintaining its own hashchain, and the hub simply transmits hashchain information to all other devices. However, it may be possible to combine hashchains in more interesting ways. For example, a user may want a *single* hashchain for the climate control subsystem of their house and a different hashchain for the lighting system. At that point, the question of where the hashchain is stored and updated becomes a critical one. At another extreme, a user may simply want a single hashchain for all updates to the IoT system that is maintained by the hub. Figuring out the precise tradeoffs here remains an open question.

4 Implementation

We now present how our two ideas are combined and discuss various implementation issues. In particular, we discuss how to initialize our system and some remaining issues in order to have efficient runtime performance. Then, we describe some crucial features that our system must be able to provide: efficient failover during complete hub failures and compile-time model checking to manage complex user policies.

4.1 Initialization

Initializing a device merely requires setting its owner key so the device knows the user's identity. The owner key allows the user full control over the device and its internal state, including issuing commands and setting policies involving the device. Additionally, the device must also start with a set of base axioms in order to process proof-carrying statements. These base axioms include things like what data sources the device trusts as well as any policy axioms. In Figure 2, axioms for trusted data sources would contain implications like $\langle\, c, \phi \,\rangle \rightarrow \langle\, c, \sigma \,\rangle$, or `ClockPolicy`, which allows the user to trust statements made by the clock about the time. Similarly, $\langle\, b \wedge c, \sigma \,\rangle \rightarrow \langle\, a, \sigma \,\rangle$, or `TempPolicy`, is a policy axiom that told the heater to turn on when the temperature gets too cold at night.

These base axioms can get fairly long and complicated so the user may not wish to directly specify very many of them. In this case, the user can authorize the hub to download them from a set list of base axioms that are provided by a third party. This allows the user to obtain a baseline for their desired policy which they can then modify to suit their particular needs.

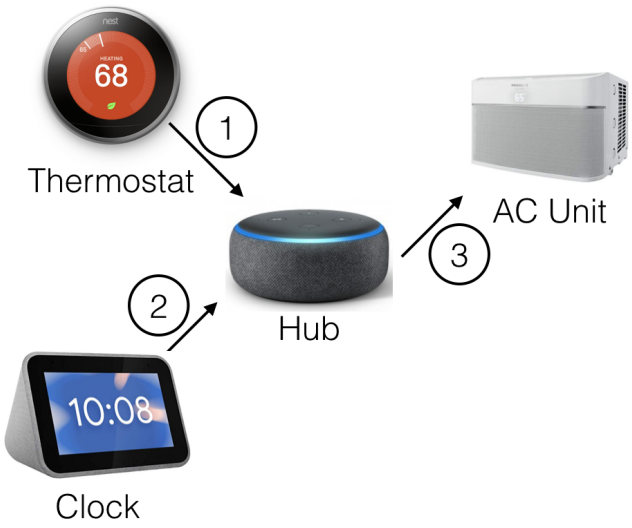

Figure 3: Control flow of a thermostat and a clock turning on the air conditioning unit. In (1), the thermostat sends a temperature update to OrbanHub. Then, in (2), the clock sends its update to OrbanHub. Finally, in (3), OrbanHub constructs a proof along with the command to the air conditioning unit.

4.2 Runtime Operation

OrbanHub consists of a single logical hub and the devices that connect to it. Devices produce statements about their beliefs along with a current hashchain commitment, and then sign them. These signed statements are sent to the hub and forwarded onto the devices that desire these updates. The hub then takes these signed statements and makes inferences about the implications of these statements. While making these inferences, the hub may deduce that some device needs to be issued a command. When issuing such a command, the hub produces a proof-carrying statement to convince the device to run that command. Upon receiving a proof-carrying statement, the device verifies the veracity of the proof and if the proof is correct, the device executes the command.

Figure 3 shows the control flow for the air conditioning unit to turn on in OrbanHub. First, the thermostat sends its temperature reading to the hub and the clock sends its time update to the hub. Then, the hub, internally, computes any and all implications of these two readings based on the user policies that were specified. If no implications were found, then the hub sends out the readings to the air conditioning unit to prove liveness. In this case, no further action is taken and OrbanHub waits for new updates from devices. If the hub finds that some actions need to be taken, the hub computes a proof that the action needs to happen and sends it on to the relevant devices. In our example, the hub computes a proof and instructs the air conditioning to turn on. Finally, the air conditioning unit verifies that the hub's proof is correct and then executes the action by turning on.

A naive hub implementation would be too slow to process updates from even moderate-sized IoT systems as it would need to check all policies on all devices to see which one may mandate an action. This is not feasible as updates arrive constantly and the policy files may grow to become too large. There are a few ways to implement this operation in order to make it more tractable. First, a hub can outsource this task to a cloud provider, which can leverage larger computational resources. Additionally, if a cloud provider provides this as a service, then they can leverage economies of scale to perform this task more cheaply. A second way to do this would be for the hub to implement it locally and preprocess the policy files. Each policy can be stated as an if-then statement, where the preconditions may be triggered by a new statement. Locally, there might be an efficient way to sort preconditions in order to match incoming statements very efficiently. However, the details of such an implementation remain unanswered.

4.3 Fault Tolerance

Fault tolerance is critical to implement as a failed hub can cause the entire IoT system to cause damage to the physical world. OrbanHub implements fault tolerance through a policy and having multiple hubs from potentially different providers. A user-defined policy can explicitly specify backups and define a policy to turn on the backup if the primary has not sent out a heartbeat update in a while. The backup hub can be from a different provider, which means that if a provider is compromised or down, the IoT system can still stay alive. Further, it allows the user to swap out hubs when they desire and as new hubs provide more powerful functionality.

4.4 Compile-Time Model Checking

Policy files are large and can often contain policies whose combinations may result in unintended and undesirable consequences. For example, if the user policy requires adjacent rooms to be maintained at vastly different temperatures, then the air conditioning and heater may have to be turned on simultaneously. This is undesirable as the air conditioning unit and heater would be working against each other, but such an event may become possible due to complex policy files. At compile time, the user, in addition to the policy, can enunciate some invariants that the compiler can ensure are held by the user policy. In the above example, the compiler can make sure that the heater and the closest air conditioning unit are never turned on at the same time. Most importantly, this optimization would not affect runtime performance, though model-checking would have to occur every time user policies are updated.

5 Related Work

IoT Hub Architectures. Various proposals for IoT hub architectures have been explored for a while now [14]. Newer architectures have been proposed with new programming abstractions [12] or for application specific purposes [21], but they tend to avoid the question of what happens if the hub is compromised. Architectures that can tolerate compromised hubs are decentralized [26], but the de-facto standard for IoT hubs is a more centralized design. In contrast, OrbanHub can tolerate compromised hubs without fully decentralizing them which fits the emerging standard while preserving most of the benefits a peer to peer architecture would bring.

Untrusted storage. Hashchains have been used in untrusted storage system in order to prevent tampering of a log of operations [6, 7, 9, 15, 17, 24]. The IoT problem domain is vastly different, however, so OrbanHub requires a more subtle use of hashchains. OrbanHub cannot require all devices to store the full log of operations as there are many operations and not enough memory on each device. Additionally, IoT devices do not need a consistent ordering of all operations in order to serve users appropriately. Taking this into account, OrbanHub uses proof carrying statements in order to provide a weaker consistency guarantee than untrusted storage systems. OrbanHub only uses hashchains to guarantee liveness, which is only a subset of the problem solved by untrusted storage systems.

Authorization. The core technique for determining safety in OrbanHub is authorization, where devices make statements that, in turn, authorize another device to take some action according to a user-defined policy. The general area of authorization has been widely studied in the past, but most works are not applicable to the IoT domain where devices need to know whether they are authorized to take an action. Many works focus on delegation of access [3–5, 16, 22] to an object as well as some actions on that object. Wave [2] is a delegated authorization mechanism for IoT devices, but it is decentralized which goes against the dominating IoT architecture. In OrbanHub, it is not enough to allow a hub or device to unilaterally start executing some function since user policies specify the precise conditions under which they can be executed.

6 Conclusion

We have seen an enormous explosion of IoT devices that use a centralized hub for communication and coordination. With only a few centralized hub operators, a compromised hub can have disastrous consequences for a large number of IoT systems. A well resourced attacker can, by compromising a single hub, cause widespread damage to a nation by taking down the electric grid or ruining the climate control on the food supply. Thus, it is critical for IoT systems to be designed so that a compromised hub cannot cause widespread damage to the system it controls.

This paper presented the first architecture for IoT where the hubs cannot take actions unless explicitly authorized by the users they serve. Even a Byzantine hub compromised by an attacker or an insider cannot take action without being programmed by the user. We implement this using two ideas:

proof-carrying statements and hashchains. Proof-carrying statements make it impossible for the hub to convince a device to execute some function that it was authorized to do so. Hashchains make it impossible for the hub to hide pertinent updates from devices so as to ensure that they actuate on fresh data. With these two techniques, OrbanHub can ensure that even if the hub is compromised, the range of misbehaviors is strictly confined to an envelope specified by the user.

7 Acknowledgements

We would like to thank the anonymous reviewers for their feedback on our paper. This work was supported by IC3, the Initiative for Cryptocurrencies and Smart Contracts. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

8 Discussion Topics

In accordance with the workshop format, we now present some topics for discussion.

Feedback We are looking to receive feedback on the design choices we made around fault tolerance and how, specifically, the hashchains should be implemented for liveness.

Discussion The premise that the hub should be untrusted is the most interesting discussion point raised in this paper. An untrusted hub raises critical issues about security and privacy in the IoT hub model. Right now, the industry has largely focused on improving functionality, but as IoT devices become more ubiquitous, privacy and security becomes more important. While this work does not solve all privacy and security issues, as far as we know, this is the first architecture that starts with the assumption that the hub is untrusted.

Controversial Points The most controversial point in this paper is our untrusted hub design. While we believe that having a trusted hub for IoT devices is a drawback, the industry providers may choose not to prioritize making untrusted hubs. The untrusted, but centralized, hub is the right problem to solve and that our techniques would solve liveness violations effectively.

Open Issues The major open issue not addressed in the paper is confidentiality about the data. We assume that the hub cannot be trusted for safety and liveness and consequently cannot take unauthorized actions, but the hub can see and process all updates and operations done by the devices. This enables large scale data collection of potentially sensitive data. Our current design is only to ensure that safety and liveness are guaranteed even if the hub is misbehaving. We view our current design as a first step in this larger vision, as there are many different ways to implement confidentiality and the best option is heavily dependent on the underlying hub implementation.

Weaknesses of the Idea The biggest weakness of our idea is if the security measures we implement are not lightweight enough. Whether this weakness can be mitigated or not depends heavily on the use cases for IoT systems as well as what our concrete design decisions will be. However, the complexity of current system on a chip designs, such as the ESP 8266, indicates that the complexity of a proof checker, which pales in comparison to the typical WiFi stack, should not be a large barrier for OrbanHub.

References

[1] https://developer.amazon.com/en-US/alexa. Accessed on 02/19/2020.

[2] Michael P Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E Culler, and Raluca Ada Popa. {WAVE}: A decentralized authorization framework with transitive delegation. In *USENIX Security Symposium*, pages 1375–1392, 2019.

[3] Moritz Y Becker, Cédric Fournet, and Andrew D Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[4] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[5] Matt Blaze, Joan Feigenbaum, and Angelos D Keromytis. Keynote: Trust management for public-key infrastructures. In *International Workshop on Security Protocols*, pages 59–63. Springer, 1998.

[6] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying computations with state. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 341–357, 2013.

[7] Christian Cachin, Idit Keidar, and Alexander Shraer. Fail-aware untrusted storage. *SIAM Journal on Computing*, 40(2):493–533, 2011.

[8] https://www.amazon.com/ESP8266-microcontroller-NodeMCU-WiFi-CH340G/dp/B071RNQPHV. Accessed on 02/20/2020.

[9] Ariel J Feldman, William P Zeller, Michael J Freedman, and Edward W Felten. Sporc: Group collaboration using untrusted cloud resources. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 337–350, 2010.

[10] https://www.home-assistant.io/integrations/google_assistant/. Accessed on 02/19/2020.

[11] https://ifttt.com/. Accessed on 02/19/2020.

[12] Kumseok Jung, Julien Gascon-Samson, and Karthik Pattabiraman. Oneos: Iot platform based on posix and actors. In *USENIX Workshop on Hot Topics in Edge Computing*, 2019.

[13] https://www.youtube.com/watch?v=c1XyhReNcHY&feature=emb_title. Accessed on 02/27/2020.

[14] Philip Levis, Samuel Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric A Brewer, and David E Culler. The emergence of networking abstractions and techniques in tinyos. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[15] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. Secure untrusted data repository (sundr). In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 9–9, 2004.

[16] Ninghui Li, John C Mitchell, and William H Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130. IEEE, 2002.

[17] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud storage with minimal trust. *ACM Transactions on Computer Systems*, 29(4):1–38, 2011.

[18] https://store.google.com/us/product/nest_protect_2nd_gen. Accessed on 02/19/2020.

[19] https://nest.com/. Accessed on 02/19/2020.

[20] https://www.nbcnews.com/news/us-news/nest-camera-hacker-threatens-kidnap- Accessed on 02/20/2020.

[21] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. Towards a serverless platform for edge {AI}. In *USENIX Workshop on Hot Topics in Edge Computing*, 2019.

[22] Ronald L Rivest and Butler Lampson. Sdsi-a simple distributed security infrastructure. In *Advances in Cryptology – CRYPTO*, 1996.

[23] http://sdb.cs.berkeley.edu/sdb/. Accessed on 02/19/2020.

[24] Srinath Setty, Soumya Basu, Lidong Zhou, Lilith Stephenson, and Ramarathnam Venkatesan. Enabling secure and resource-efficient blockchain networks with volt. Technical Report MSR-TR-2017-38, 2017.

[25] Emin Gün Sirer, Willem de Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B Schneider. Logical attestation: an authorization architecture for trustworthy computing. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[26] Aleksandr Zavodovski, Nitinder Mohan, Walter Wong, and Jussi Kangasharju. Open infrastructure for edge: A distributed ledger outlook. In *USENIX Workshop on Hot Topics in Edge Computing*, 2019.