

HOPPER: Per-Device Nano Segmentation for the Industrial IoT

Piet De Vaere
ETH Zürich
Switzerland
piet.de.vaere@inf.ethz.ch

Andrea Tulimiero
ETH Zürich
Switzerland
andrea@andreatulimiero.com

Adrian Perrig
ETH Zürich
Switzerland
adrian.perrig@inf.ethz.ch

ABSTRACT

Today’s industrial networks heavily rely on perimeter-based security. Although this has worked well in the past, the advent of the industrial Internet of Things (IoT) is blurring the network boundary, and thereby undermining the effectiveness of perimeter-based network defences. To address this, we propose HOPPER: an industrial IoT security protocol that places each network host in its own access-controlled *nano segment*, thus minimizing the attack surface introduced by connecting devices to the network. Because HOPPER enforces nano segmentation in-fabric, it does not require modifications to how packets are routed. HOPPER achieves this by allowing each network node to verify that each packet it processes is part of a desired flow and was generated by an authorized host. Packets that fail any of these checks are dropped en route. By leveraging prevalent industrial network features, HOPPER accomplishes low management and bandwidth overhead while being suitable for a wide range of networks. Our implementation on IoT-class hardware demonstrates that HOPPER achieves high throughput and scalability, even in constrained environments.

CCS CONCEPTS

• **Networks** → **Network architectures; Network protocol design**; • **Security and privacy** → **Security protocols**.

KEYWORDS

Network security; industrial IoT; key distribution; capability-based networking; network segmentation; nano segmentation

ACM Reference Format:

Piet De Vaere, Andrea Tulimiero, and Adrian Perrig. 2022. HOPPER: Per-Device Nano Segmentation for the Industrial IoT. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS ’22)*, May 30–June 3, 2022, Nagasaki, Japan. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3488932.3501277>

1 INTRODUCTION

Over the last decade, we have increasingly witnessed industrial control systems becoming the target of sophisticated cyber attacks. These incidents range from attacks attributed to nation state adversaries, such as Stuxnet [32] and the attacks on the Ukrainian power grid in 2015 [28], to criminal ransomware attacks, such

as EKANS [19] and TRISIS [18]¹. Contrary to more traditional ransomware, EKANS explicitly targets industrial control systems. TRISIS takes this approach even further by specifically targeting the industrial safety controllers which function as a last resort to prevent catastrophic process failures.

Although not many details about these attacks are public, a common trend is that they rely on lateral movement through the victim’s network in order to reach the critical systems which they target. This approach is often successful because today’s industrial network design practices strongly rely on perimeter protection, often having no to very little defences in place once a perimeter has been breached.

For many years this perimeter-based approach has served industry well, but the advent of the industrial Internet of Things (IoT) is undermining its effectiveness. For example, the increased deployment of information technology (IT) systems in operational technology (OT) (i.e., factory automation) networks is breaking down the strong boundary between the IT and OT worlds. Moreover, trends such as (i) the increasing prevalence of cloud-assisted devices, (ii) the increasing risk of supply-chain attacks [48], and (iii) the increasing use of both long- and short-range wireless communication [41] are blurring the network boundary further. That is, every device is becoming a potential attacker entry point, rendering perimeter based defences impractical and prompting the need for strong defences against lateral movement.

Currently, industry is responding to this trend by limiting the movement of attackers by creating more—and thus smaller—network segments, a practice referred to as *micro segmentation* [39]. However, traditional segmentation mechanisms (e.g., VLANs) can only take one so far, do not scale well, and require traffic to pass through a centralized router to move between segments, thereby introducing a single point of failure in the data plane. Instead, an ideal defense must ensure that only permitted communication can take place *throughout* the network, without reducing the network’s versatility.

In enterprise networks, a similar philosophy recently gained traction under the name *zero trust networking* [23]. However, as zero trust networking techniques tend to leverage enterprise-oriented security mechanisms, they are typically not suitable for the (industrial) IoT. For example, the highly constrained nature of many IoT devices renders the use of certificates challenging. Moreover, zero trust networking focuses on the protection of resources at end-hosts, whereas in industrial networks, the network fabric itself must be protected as well. For instance, being able to generate cross-traffic on a network link carrying a control signal with hard real-time requirements can increase congestion and latency, reducing the quality of the manufactured product [31].

To address these issues, we propose HOPPER: a protocol that combines two of the best properties of zero-trust networking and

¹Also known as “Snake” and “TRITON”, respectively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASIA CCS ’22, May 30–June 3, 2022, Nagasaki, Japan

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9140-5/22/05...\$15.00

<https://doi.org/10.1145/3488932.3501277>

classical network segmentation. On a HOPPER network, each flow must be explicitly authorized, and restrictions are enforced both at end hosts, as well as in-network. Our approach de-facto results in a per-device *nano segmentation* of the network, which is enforced in-fabric. That is, in a HOPPER network, the attack surface is minimized by restricting the permitted communication to the minimum required for the operation of the deployment. By enforcing this restriction at each hop, HOPPER effectively places each device in its own nano segment.

More concretely, HOPPER achieves nano segmentation by allowing each network node to verify that each packet it processes (i) is part of an explicitly whitelisted flow, and (ii) was generated by an authorized host. HOPPER is compatible with the many constraints and networking technologies encountered in industrial IoT networks, allowing it to be uniformly deployed throughout a wide range of scenarios.

Recognizing the centrally managed nature of industrial networks, HOPPER has a centralized control plane, but is fully distributed in the data plane. Specifically, HOPPER constructs a capability mechanism based on a hierarchical key space that is used to uniquely bind each packet to an authorized flow. Once distributed, authorized senders use their capability tokens to generate authentication tags for each transmitted packet. By distributing part of the hierarchical key space to each node, HOPPER ensures that these tags can be independently verified at every network hop, while requiring only minimal local node state and using only symmetric cryptography. Further, HOPPER places no assumptions on the structure of the underlying physical network, and introduces no more per-packet overhead than a standard authentication tag. Leveraging additional common IoT characteristics allows us to keep this approach scalable while maintaining flexibility.

This paper presents the following contributions:

- (1) we present HOPPER, the first per-device nano segmentation scheme for industrial IoT networks;
- (2) we implement and evaluate a HOPPER capable host on IoT-class hardware;
- (3) we demonstrate HOPPER's scalability by implementing and evaluating HOPPER on a popular network appliance; and
- (4) we show how leveraging common features of industrial IoT deployments allows to significantly improve the scalability of key distribution schemes.

2 BACKGROUND

2.1 Traditional Industrial Networks

The structure of industrial networks highly differs from classical networks. The predominant model used in industrial automation is called the *automation pyramid*, and is illustrated in Fig. 1. The automation pyramid partitions the various systems found in industrial networks into a set of hierarchical classes based on the type of information processed in them. Although the pyramid is only loosely standardized (standards disagree on the number and names of levels), sensors and actuators can consistently be found on the lowest levels, and tools for long-term, strategic decision making (i.e., enterprise resource planning (ERP)) are found at the top [44].

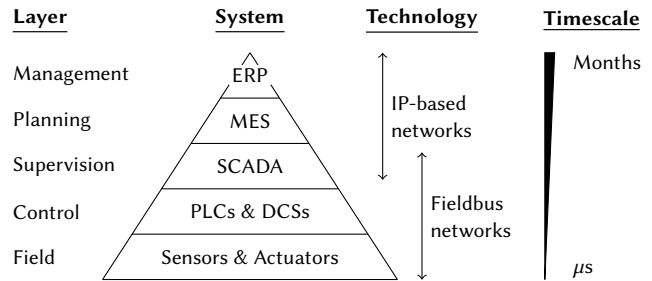


Figure 1: The automation pyramid.

A common observation throughout the pyramid is that each level operates at smaller timescales than the ones above it: where management decisions are taken on the scale of months or even years, field and control levels must operate at sub-second precision. Because of the high demands these short timescales place on the network, the lower levels of the automation pyramid use specialized *fieldbus* networks rather than standard Ethernet/IP. For historical reasons, many different fieldbusses are in use today, with common examples including Modbus, PROFIBUS, and EtherCAT.

Because they run on different protocol stacks, the lower levels of the automation pyramid are usually strongly decoupled from the top levels of the pyramid. This practice is referred to as the separation of information technology (IT) and operational technology (OT) systems. The IT partition of an organization consists of the systems used for business operations such as ERP or email. Conversely, the OT partition consists of the systems that control physical processes and infrastructure.

In many organizations this separation is further enforced by laying out the network by mapping the levels of the automation pyramid to hierarchical network segments with the primary goal of creating strong IT/OT separation through layered perimeter security. Such networks are commonly referred to as Purdue Model or ISA-95 based networks [17].

2.2 Recent Evolutions in Industrial Networks

Whereas industrial networks have traditionally been protected through IT/OT separation, recent evolutions are increasingly challenging this method, exposing industrial networks to new threats. We discuss the most relevant of these evolutions below.

IT/OT convergence. With the rising popularity of the industrial IoT, IT and OT systems are increasingly converging [16]. That is, OT systems are being closely integrated with IT networks, and thereby connected to the Internet. This convergence is exposing OT systems to threats they were never designed to handle, prompting the need for new protection mechanisms.

Time-Sensitive Networking (TSN). The convergence of IT and OT systems will likely be further accelerated by the work done on TSN by the IEEE 802.1 working group. Specifically, the TSN task group aims to provide “guaranteed packet transport with bounded latency, low packet delay variation and low packet loss” [26] over IEEE 802 (specifically, Ethernet) networks. The expectation is that TSN will make Ethernet a universally suitable replacement for today’s wired

fieldbus technologies, and will over time replace the current fieldbus hodgepodge [34, 52]. This will push systems in the lower parts of the automation pyramid towards IP protocols, further integrating them with IT systems.

Cloud-assisted devices, software updates & wireless links. Traditional industrial network defenses, such as those based on the Purdue Model, are designed to protect against adversaries attacking the network from its perimeter. We provide three examples demonstrating that this assumption no longer holds. *First*, a common IoT trend is to ship devices together with a companion cloud. We are seeing a similar trend in the industrial IoT with devices like ABB’s *Ability Smart Sensor* and Siemens’s *MindSphere*. Because cloud-assisted devices have a permanent, direct, and encrypted connection to the Internet, they violate the Purdue Model and should be considered as a potential attacker entry point directly into the OT network. *Second*, the ever-increasing complexity of industrial devices is increasing the need for software updates. If any device’s update process becomes compromised, these software updates can be used as a trojan horse, converting a previously trusted device into an adversary entry point. This also holds for devices which are not cloud assisted. *Third*, the increasing use of both long- and short-range wireless communication means that attackers can directly compromise devices without having to pass the traditional network perimeter.

As illustrated in the examples above, many new network devices should be considered as potential attacker entry points. This significantly reduces the effectiveness of traditional network segmentation as an intruder protection tool, and prompts the need for new approaches.

2.3 Constraints in Industrial IoT Networks

Whereas TSN will likely unify the wired communications in industrial environments, many industrial IoT products, especially those designed for monitoring, are using wireless communication. Contrary to the situation with TSN, it is unlikely that a universal wireless protocol will emerge in the near future. Moreover, the requirement for long (e.g., 10-year) battery lives introduces many (extreme) constraints for networks. In order to support deployment throughout industrial networks, it is important for a protocol to be able to operate under any of these constraints.

Providing a complete taxonomy of (industrial) IoT networks and devices is out of scope for this work. Instead, we discuss the diversity and possible limitations of the devices and networks that we consider. We use RFC 7228 as a basis for this discussion [11].

Devices. Industrial IoT devices range from potent, mains-powered industrial servers running standard operating systems (e.g., HPE’s *Edgeline* “converged edge systems”) to miniature “motes” with ten-year battery lives (e.g., National Control Devices’s mesh sensors). Throughout this range of devices, the following constraints may be encountered:

- C1:** *Limited state*, for program code and temporary storage.
- C2:** *Limited computation*, due to low-end processors.
- C3:** *Limited power*, additionally limiting computation and severely restricting wireless communication.

Networks. Similar to IoT devices, networks in IoT deployments span a wide range of technologies and constraints. At one extreme, IoT devices can use gigabit-speed wired connections (e.g., KUKA’s *KR C4* industrial robot controllers), while others use multi-hop mesh networks with sub-kbps data rates and highly restricting duty cycling (e.g., Analog Devices’s *SmartMesh* or Digi’s *DigiMesh* products). Throughout this range of network technologies, the following constraints may be encountered:

- C4:** *Low data rates*, due to the power constraints of the underlying devices.
- C5:** *High and unstable latency*, due to multi-hop networks and energy-conservation policies of the devices.
- C6:** *Reachability limits* of duty-cycling or default-off devices.
- C7:** *Lack of advanced network features* such as multicast.
- C8:** *Unusual routing* such as the deliberate duplication of packets for redundancy reasons, or the broadcast-like nature of flooding-based network architectures [21].
- C9:** *Instable topologies* caused, for example, by mobile nodes or opportunistic routing.

3 ADVERSARY MODEL & SECURITY GOALS

This work considers a network model in which networks consist of (i) links, (ii) forwarding elements, (iii) hosts, and (iv) a network controller, which are all managed by a network administrator. We use the generic term *forwarding element* to cover all network elements that forward packets. Network nodes can simultaneously be hosts and forwarding elements, e.g., in mesh networks.

Given this network model, we consider a network-based adversary which may have compromised a subset of hosts, forwarding elements, and network links. The attacker can have used any method (including physical access) to compromise these devices and links, and has full control over them. Moreover, the attacker can communicate out-of-band between the points of attack. However, the network controller and administrator must remain uncompromised.

The attacker’s goal is to increase the scope of his attack by exploiting the network. Possible attacker strategies include: (i) using the network to compromise additional devices, (ii) sending spoofed packets to disrupt physical processes (e.g., impersonating a sensor and sending false readings), and (iii) performing denial- or reduction-of-service attacks against the network fabric (e.g., generating traffic in order to increase network latency in critical control loops or to reduce the lifetime of battery powered IoT deployments).

HOPPER aims to mitigate such attacks by placing each device in its own access-controlled nano-segment, thus minimizing both its network exposure and its network access. Concretely, we define the following security goals:

- G1, Least privilege:** Communication involving at least one uncompromised host can only take place on the logical flows explicitly whitelisted by the network administrator.
- G2, Isolation:** An adversary that compromised a set of network elements can only generate network traffic between these elements, or towards the union of destinations these elements interact with under normal network operation.
- G3, Authentication:** Each packet is source authenticated to its receiver, preventing the adversary from spoofing packets from hosts that are not under its control.

We explicitly do not list confidentiality as a security goal,² but we briefly discuss how it could be added to our design in Section 5.6.

4 HOPPER PROTOCOL

We demonstrate how nano segmentation can be achieved using the example network shown in Fig. 2, which is based on a hydroelectric power plant. The primary section (i.e., the main plant building) of the network is Ethernet-based. The hosts on this part of the network constitute everything from servers and engineering workstations to turbine controllers and printers. Further, this section contains the network controller and the main Internet uplink. Connected to the primary network are a remote network (e.g., an upstream sluice complex) and a mesh network which contains low-power sensors mounted to the machinery in the plant. The link between the primary and remote network section may be virtual (e.g., a VPN tunnel) or physical (e.g., a long-range Wi-Fi link).

Because the network in Fig. 2 covers a single plant, it is administrated by a single entity (i.e., the plant’s network management team). This is typical for industrial networks, and we refer to it as *ownership centrality*. Even though the network administrator might not be the most privileged user on all devices (e.g., in the case of devices that are managed through a vendor-operated cloud), they still set the policies each device should comply with, and—in the worst case—can remove non-complying devices from the network.

Further, although industrial facilities may rely on distributed data flows, they are typically centrally orchestrated, which we refer to as *orchestration centrality*. Orchestration centrality holds for our plower plant, but also, for example, for a smart factory, which will be configured by its operators to execute a specific set of tasks. Moreover, from the automation pyramid introduced in Section 2.1, we know that these tasks are temporally stable and change at most a couple of times a day.

The combination of ownership and orchestration centrality, together with a focus on a small number of temporally-stable, cyber-physical workloads, is a defining characteristic for most industrial networks. Moreover, this combination of properties makes it possible for network administrators to both compile an a priori whitelist of legitimate flows, and establish network policies that only allows packets that are part of a legitimate flow to use the network. When enforced at every hop, this effectively creates a nano segment for each device on the network, while still allowing packets to be routed on the shortest physical path between their source and destination.

Further, industrial networks—similar to enterprise networks—are strongly zone-oriented. In fact, network zoning is mandated by IEC 62443 [27], the leading standard for security in industrial networks. However, contrary to zoning in the enterprise, zoning in OT networks is typically done based on the automation pyramid (see Section 2.1), and different zones may contain radically different types of devices. For example, a SCADA zone may contain standard workstations and servers, whereas a field zone can consist of a mesh of low-power sensors.

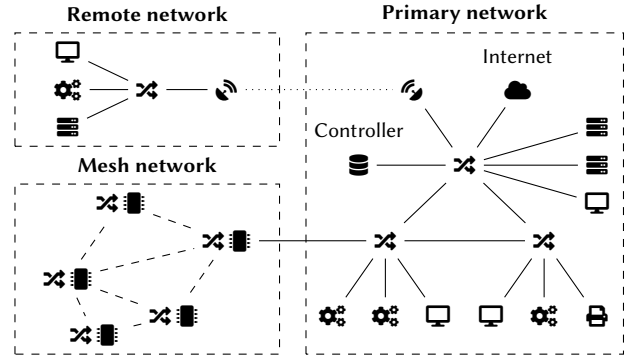









Figure 2: An example industrial network.  represents a forwarding element; , , , , and  represent hosts; and  represent a remote link. Dashed links are wireless.

Depending on the network zone, a different subset of the constraints listed in Section 2.3 will be encountered, changing the restrictions and requirements placed on (security) protocols. In order to cope with this diversity, we design HOPPER to be fundamentally compatible with *all* of the listed constraints, and simultaneously allow it to be adapted to specific deployment environments through parameterization. This allows a single, well-understood, security protocol to be deployed throughout the network, rather than requiring a different protocol design for each type of network zone.

As of yet, no protocols allow segmentation policies to be implemented homogeneously throughout the network while being compatible with the constraints of IoT and industrial networks. In the following subsections, we explain how we fill this void with the design of HOPPER, thereby enabling the nano segmentation throughout industrial IoT networks.

4.1 High-Level Design

In order to meet the least privilege goal (G1) described in Section 3, HOPPER must verify that each packet is part of a whitelisted flow at at least one point in the network: either at the receiving host or at a forwarding element. As in mesh networks or in networks using a shared medium it is not guaranteed that a flow traverses a forwarding element (C8), this check must be performed on the receiving host. However, to meet the isolation requirement (G2), packets must be checked at each forwarding element as well. Thus, every node in the network must verify the permissions of every packet it processes.

The high availability requirements posed to industry require that no single point of failure exists in the data plane. Moreover, when the network is partitioned by the failure of a link or forwarding element, the individual partitions of the network should remain operational. For example, if the remote link in Fig. 2 fails, the remote network should remain in operation. This means that packet checks must take place without the online assistance from the centralized controller. Further, because transceiving data in wireless (mesh) networks consumes multiple orders of magnitude more energy than performing (symmetric) cryptography [25, 45], nodes should be able to verify packets using only static, node-local knowledge (C2, C3). This requirement is further reinforced by the low data rates (C4),

²An attacker who can compromise packet integrity can de facto take over control over an industrial process, which in turn leads to a major safety risk. Because this is not the case for compromised confidentiality, integrity is considered a much more important goal than confidentiality in industrial settings.

high latency (C5), limits on reachability (C6), and lack of advanced features (C7) common to low-power wireless networks.

There are two general methods that can be used to allow nodes to verify if a packet is part of a whitelisted flow: (i) access lists or (ii) capabilities. Because using access lists would involve disseminating large amounts of information to each node of the network—which is impractical in constrained networks (C1, C3–C7)—HOPPER follows a capability-based approach.

Past work on deny-by-default networking has used capabilities in the form of authenticated source routes [14]. However, two characteristics of industrial IoT products render such an approach unsuitable. First, roaming nodes (e.g., a sensor mounted on an automated guided vehicle (AGV)), duty cycling nodes, and fading wireless links lead to dynamic and unpredictable network paths (C9). Second, for reliability reasons, some protocols (e.g., Ethernet TSN) include mechanisms to replicate packets in-network and to send these packets over different paths (C8), complicating source routing.

Instead, a capability scheme that allows each node in the network to verify the permissions of each packet in the network should be used. Moreover, cryptographically binding together capability tokens with their packets allows HOPPER’s authentication goal (G3) to be satisfied. However, limited computational power severely restricts the use of asymmetric cryptography (C2, C3).

Considering these requirements, we observe many similarities with multicast-authentication, in which many listeners must be able to verify that a packet was generated by an authorized sender [12, 42], and we use past results from this space as a basis for HOPPER.

Based on the discussion above, we design HOPPER to authenticate each packet by adding a cryptographic tag to it. This tag consists of a concatenation of multiple message authentication codes (MACs), which are generated using keys from a hierarchical key space. By linking the keys in this key space to the flow identifiers carried in the packets, a capability scheme is constructed.

In order to be able to verify that packets are part of an authorized flow, each forwarding element is provided with the state needed to partially verify any HOPPER tag. Hiding which forwarding element verifies which part of the tag forces the adversary to consider each forwarding element to verify the entire HOPPER tag. Moreover, using a key space with multiple levels of hierarchy allows receiving hosts to fully verify the HOPPER tags of all incoming packets.

Given the expected prevalence of UDP/IP at the lowest levels of industrial networks, we focus our design of HOPPER on this protocol stack. Although TCP is a more popular transport at higher levels off the automation pyramid, the trend in both IoT and industrial standards is towards UDP in constrained environments. For example, whereas OPC UA uses TCP on higher network levels, a UDP mapping was specified for OPC UA over TSN [22]. Nonetheless, HOPPER’s design can be easily modified for operation on different layers or protocol. For example, in the evaluation (Section 7), we also implement and evaluate HOPPER for L2 Ethernet.

4.2 Constructing Capability Tokens

The hierarchical key space is constructed as a forest consisting of n trees, as shown in Fig. 3, and where n is a configuration parameter. The keys in the i -th tree are used to generate and verify the i -th

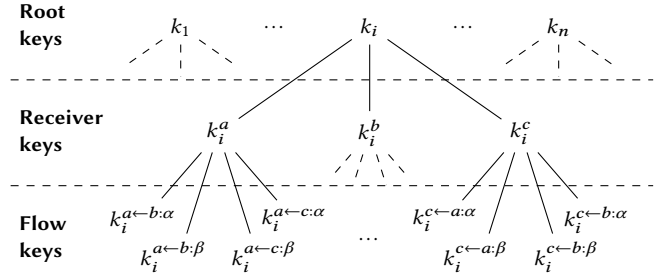


Figure 3: Excerpt from a HOPPER key forest for three hosts (a , b , and c) and two ports (α and β).

MAC in the HOPPER tag. The root of each tree i is a random secret, denoted k_i . The set of all roots

$$\mathcal{K} = \{k_i \mid i \in [n]\} \text{ with } [n] := \{1, \dots, n\}$$

is called the root key set and is only fully known to the network controller, with subsets being distributed to all forwarding elements. Further, for each receiver r , a receiver set

$$\mathcal{K}^r = \{k_i^r = \text{PRF}_{k_i}(r) \mid i \in [n]\}$$

of receiver keys is derived from \mathcal{K} . PRF_{k_i} is a pseudorandom function keyed with k_i .³ The receivers store these keys locally and use them to verify incoming packets.

If the network controller wants to issue a capability token for host s to communicate with host r on UDP port p , it generates the flow set

$$\mathcal{K}^{r \leftarrow s:p} = \{k_i^{r \leftarrow s:p} = \text{PRF}_{k_i}(s \parallel p) \mid i \in [n]\}$$

and provides it to host s . “ \parallel ” represents string concatenation.

4.3 Sending and Verifying Packets

Once a host receives a flow set, it can send on the corresponding flow. To send a packet, the host calculates a tag τ consisting of one MAC over the packet payload for each key in the flow key set, i.e.,

$$\tau = \text{MAC}(k_1^{r \leftarrow s:p}, \text{payload}) \parallel \dots \parallel \text{MAC}(k_n^{r \leftarrow s:p}, \text{payload})$$

and adds it to the packet. Because the cryptographic strength of τ is dependent on its full length, the individual MACs can be short [12].

While the packet traverses the network, each forwarding element it encounters verifies the MACs for which it can derive the flow key using the packet’s header info; i.e., the MACs at the positions for which it has the root secrets. If any of these MACs are incorrect, the packet is dropped and reported to the network controller, signaling the presence of an attacker. The receiver always verifies the entire HOPPER tag by deriving the flow key set from its receiver key set using the information in the packet header.

4.4 Distributing Root Keys

By provisioning forwarding elements with subsets of the root key set, HOPPER allows each forwarding element to partially verify each HOPPER tag without them being able to forge valid tags. However, properly distributing the root keys is a complex problem, as a good

³In practice, a block cipher encryption operation can be used to implement the PRF.

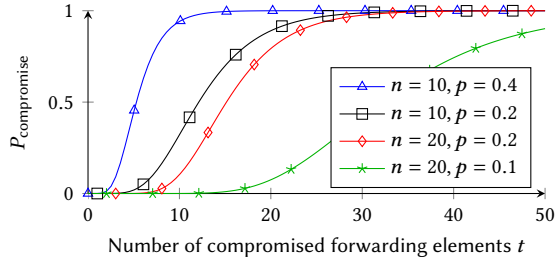


Figure 4: Probability that an attacker learns the full root key set when compromising t forwarding elements, assuming random root key distribution.

distribution scheme must satisfy two contradicting goals: (i) as much of the HOPPER tag as possible must be verified as early in the network as possible, and (ii) when an attacker compromises one or more forwarding elements, he should not learn a sufficient number of root keys to forge HOPPER tags.

The first of these goals ensures that packets are dropped early, which minimizes their effect on the network. Intuitively this can be achieved by provisioning each forwarding element with as many keys as possible, thus ensuring that it can verify a large part of the HOPPER tag. Yet, this conflicts with the second goal, as it makes it easier for an attacker to learn the full root key set. In addition, a good key-distribution scheme should ensure that even when the attacker has compromised some keying material, forged packets will still be dropped as close to the sender as possible.

Inspired by multicast security, we propose three key distribution schemes: cover-free families [42], random, and manual distribution.

Cover-free families are combinatorial constructs on sets: An (m, n, t) -cover-free family is a family of m subsets on a ground set of size n , so that none of the sets in the family is a subset of the union of t other sets in the family. Considering a network with m forwarding elements and a root key set of size n , we can assign each forwarding element the keys from a set from a (m, n, t) -cover-free family over the root key set. This guarantees that even when t forwarding elements are compromised, each forwarding element will still verify at least one MAC for which the adversary did not obtain the key. Unfortunately constructing cover-free families with large m and t is considered a hard problem [42].

Random distribution of the root keys avoids the construction problems of cover-free families. Consider again a root key set of size n , of which each key is now assigned to each forwarding element with probability p . This ensures that, in expectation, each forwarding element verifies $n \cdot p$ MACs per packet. When t nodes are compromised, the adversary will have obtained the full root key set with $P_{\text{compromise}}(t) = (1 - (1 - p)^t)^n$. The expected number of compromised keys is $n \cdot (1 - (1 - p)^t)$.

For example, setting $n = 10$ and $p = 0.2$, results in each forwarding element verifying 2 MACs on average. However, even if the adversary compromises $t = 5$ forwarding elements, it will have obtained the full root set with probability of only $P_{\text{compromise}}(5) = 0.018\%$, and is expected to know only 6.7 keys. Figure 4 shows how $P_{\text{compromise}}(t)$ varies for various values of n , p , and t . Interestingly,

for a given value of t , $P_{\text{compromise}}(t)$ is independent of the total number of forwarding elements in the deployment.

Manual distribution of the root keys can be useful for small deployments. However, as it is hard to perform a systematic evaluation of this strategy, this work focuses on random distribution.

Regardless of the key distribution scheme, a compromised forwarding element will leak part of the root key set. Such leakage reduces the effective strength of the HOPPER tag, as the adversary needs to guess fewer tag bits. Moreover, when random root key distribution is used, the adversary might be able to compute all MACs verified at a specific forwarding element. Nonetheless, even when the attacker lacks a single root key, it is highly likely that the adversary’s presence on the network will be detected. This is because (i) the receiving host always verifies the full tag, (ii) tags are different for each packet, and (iii) generating a single bad tag will lead to the tag verification failing (either at a forwarding element or at the receiver) and the network controller being notified.

4.5 Security Equivalence & Network Dichotomy

Both cover-free families and random distribution schemes are effective to make HOPPER resistant against the compromise of a small number of forwarding elements. However, IoT deployments can be highly homogeneous and can consist of thousands of nodes. Moreover, in industrial networks the same type of sensor or actuator is often reused throughout a network or facility. In such settings it is likely that when one device is compromised, many will be. This especially holds if the initial node was compromised using a network attack, as the marginal cost to compromise additional nodes will be close to zero. After all, these nodes are likely to run similar software in similar configurations.

While this may seem troublesome at first, we show how the introduction of *security equivalence classes* can mitigate the negative effects that homogeneity has on HOPPER’s resilience. Moreover, we highlight how the dichotomy found in most infrastructure-based networks further increases resilience.

Security equivalence. Typically, the robustness of key distribution schemes is expressed in the number of devices that can be compromised before security breaks. However, as discussed above, in IoT deployments it is likely that if a single device is compromised, many devices will be. We formalize this notion by partitioning forwarding elements in *security equivalence classes*. Concretely, we assume that either all devices in a security equivalence class are compromised, or that none are. We observe that under this assumption, there is no security benefit in provisioning the forwarding elements within one equivalence class with different root keys. Specifically, when an equivalence class is compromised, all the keys present in the class will be known to the adversary regardless of how they are distributed. Similarly, when the class is not compromised, provisioning all root keys known to the class to each of its members ensures that as much of the HOPPER tag as possible is verified by each element.

Distributing root keys to security equivalence classes rather than to individual devices significantly increases the scalability of HOPPER: instead of tolerating the compromise of t devices, HOPPER now tolerates the compromise of up to t device *classes*. As for

Table 1: Effect of various parameters on HOPPER performance, assuming random root key distribution.

Parameter		Sender workload	Receiver workload	Forw. el. workload	Tag forging success prob.	Resilience to bad forw. el.	Bandwidth overhead
n	Size of root key set	▲	▲	-	▼	▲	▲
m	# forwarding elements	-	-	-	-	-	-
p	Key sampling prob.	-	-	▲	▼ [†]	▼	-
l	Individual MAC length	-	-	-	▼	-	▲

Triangles indicate the properties' direction of change if the corresponding parameter increases. ▼/▲ indicates a desirable direction of change, ▼/▲ a deterioration. [†]Forged tags more likely to be detected early in network.

hybrid host-forwarding element devices there is no security benefit in having devices share receiver keys, we only apply this strategy to root keys. Doing so ensures that when a security equivalence class would nonetheless be only partially compromised, HOPPER's end-to-end properties still hold within that class.

How to divide an IoT network into security equivalence classes is an exercise that must be made individually for each network. More broadly speaking, finding optimal partition schemes for a given network opens up an interesting avenue for future research. For example, when defending against network-based attacks, the primary device properties to consider are the logical network layout, device types, and configurations. Conversely, when physical attacks are the primary concern, all devices in the same physical access zone can be combined in a security equivalence class.

Network dichotomy. Most infrastructure-based IoT networks, much like traditional computer networks, display a clear split between hosts, and forwarding elements. We refer to this as *network dichotomy*. As in such networks the hosts do not forward packets, they do not hold root keys. Hence, no number of compromised hosts will lead to the exposure of the root keys. Additionally, the forwarding elements, which do hold root keys, will be fewer in numbers and are typically more hardened than hosts connected to them.⁴ This further increases HOPPER's resilience to node compromise. We note that a similar observation was made in the multicast authentication setting by Canetti et al. [12].

5 PRACTICAL CONSIDERATIONS

5.1 Rekeying and Revocation

As part of standard security practices, HOPPER networks should be periodically rekeyed. Maintaining a symmetric key for each controller-host pair allows the network controller to securely communicate the updated HOPPER keys with each network device, meaning that rekeying can be automated. By using a hierarchical key system similar to a tree in a HOPPER forest, the secure storage requirements on the network controller can be minimized. In the case the network was compromised and part of the root key set was exposed, a rekeying of the network is also required.

⁴For example WiFi access points and backbone forwarding elements are likely to be more hardened than the low-cost IoT devices connected to them.

Periodic rekeying also prevents hosts from accumulating capabilities. That is, HOPPER requires each node to discard capabilities (i.e., flow key sets) upon receiving a revocation notice. Because uncompromised devices can generally be expected to behave as specified, in most circumstances this is sufficient. Still, periodic rekeying ensures that also the network permissions of (dormant) malicious hosts are periodically reset. By (i) using keys with overlapping validity periods, and (ii) performing rekeying together with production reconfiguration events, the impact of rekeying events on the physical process can be minimized. Coinciding rekeying events with a production reconfiguration further makes sense, as only during such events new flow keys, the accumulation of which we wish to prevent, are issued.

5.2 Handling Network Diversity

HOPPER has a number of parameters that can be adjusted depending on the needs of the network or zone it is deployed in. For example, when deployed in a mesh-like network, HOPPER tags should consist of possibly many (n is large), but short (l is small) MACs. Conversely, when deployed on a network with only a single forwarding element (e.g., a network with a single gateway and a star topology), tags could consist of a single (n is small), but long (l is large) tag. This flexibility allows a single, well-understood, protocol (i.e., HOPPER) to be used in a wide range of settings rather than requiring the design of one-off protocols for each new setting.

Table 1 summarizes how different parameters influence HOPPER's performance. We note that the number of forwarding elements in the network does not influence the data-path overhead introduced by HOPPER.

Besides modifying HOPPER's basic parameter, it is also possible to adapt HOPPER for operation on different protocols or layers. This is done by adapting HOPPER's key space, for example by deriving HOPPER keys based on Ethernet instead of IP addresses. For other protocols the required changes can be more significant. For instance, in multi-receiver protocols the flow keys must be distributed between the receivers in a similar manner as the root keys are distributed to forwarding elements.

5.3 Connecting Networks

HOPPER's centralized control gives rise to a natural notion of HOPPER *domains*: regions of the network sharing a set of HOPPER root keys and parameters. When packets cross the borders of a HOPPER domain, HOPPER tags must be removed from, or added to, the packet. This is done by HOPPER gateways, an extended version of the HOPPER delegates introduced in Section 5.5.

For outgoing traffic the gateway verifies the HOPPER tag, removes it from the packets, and transmits the packet over its outgoing interface. For incoming traffic, the gateway functions similarly to a firewall in drop-by-default mode: the gateway verifies if the traffic is part of a whitelisted flow, and if so attaches the corresponding HOPPER tag to the packet before forwarding it to the domain-internal destination. If the traffic is not whitelisted, it is dropped. Gateways must have access to the flow keys for each flow that traverses them. As gateways are typically part of the network infrastructure and do not suffer from the constraints listed in Section 2.3, this is a reasonable requirement.

5.4 Management Overhead

As HOPPER was designed to require explicit whitelisting for every flow, it naturally introduces administrator overhead to the network. Specifically, overhead is created (i) when new devices are added to the network, and (ii) when a new workload is deployed. We proceed to quantify the overhead introduced by each of those events.

New devices. When a new device is added to the network, it must be provisioned with a minimum of keying material in order to be able to start communicating. When a key provisioning server and bootstrapping mechanism similar to the ones described in Section 5.7 are used, each new device needs to be provisioned with only a single set of flow keys. This creates an administration overhead that is comparable to provisioning devices with WPA-Enterprise credentials.

New workloads. When a new workload is configured, flow keys must be created and distributed for each required flow. Also the overhead of this operation can be significantly reduced through the use of a key provisioning server as described in Section 5.7. Concretely, when using a key provisioning server, the administrator overhead required to authorize a new flow is comparable to overhead in the (hypothetical) case were all devices must communicate through a centrally-managed, drop-by-default firewall.

Although the introduced overhead can appear high at first, it is worth noting that beside the security aspects discussed before, HOPPER's whitelist-based approach also provides another valuable property: transparency of network assets. From our interactions with industry practitioners, we have learned that a lack of insight into the devices and flows present in industrial deployments is a real problem that is often encountered when performing a security assessment of industrial networks. HOPPER alleviates this problem by having a security-by-design approach that requires all devices and flows to be listed (and therefore inventoried) before they can participate in the network.

Further, we observe that the set of flows that need to be whitelisted is a direct function of the workload placed on an industrial deployment. This opens opportunities to integrate HOPPER whitelist management into production management, which could lead to a significant reduction of management overhead.

5.5 Legacy Compatibility

Industrial installations often have lifetimes that can reach in to the tens of years, during which they are incrementally updated. This means that it is important for new industrial systems and protocols to be brownfield compatible, i.e., be able to operate in coexistence with legacy systems. When naively deploying HOPPER alongside non-HOPPER aware protocols and devices, security issues may arise. For example, an adversary may be able to mount a downgrade attack by removing the HOPPER header from a packet.

In this section, we discuss how HOPPER can be securely deployed alongside legacy systems. We split this discussion into two orthogonal parts: interoperability with legacy networking nodes (i.e., hosts and forwarding elements), and interoperability with legacy networking protocols.

Legacy Nodes. We propose three brownfield strategies that allow HOPPER to be deployed on a network with legacy nodes: (i) HOPPER delegates, (ii) zone-based deployment, and (iii) an overlay strategy.

As its name suggests, the first strategy introduces the concept of a HOPPER delegate, which is used to connect hosts that are not HOPPER-aware to the network. All traffic from/to the HOPPER-unaware host is routed over its delegate before entering/leaving the HOPPER domain. The delegate holds the HOPPER keys of the host and adds, checks and removes HOPPER tags on behalf of the host. By implementing delegatee functionality in access switches, this can be accomplished with minimal management overhead and while maintaining all of HOPPER's core properties.

When using the zone-based strategy, HOPPER-aware network nodes are deployed grouped in zones. These zones are then connected to the non-HOPPER aware parts of the network using gateways which add and remove the HOPPER headers as needed. The concept of a HOPPER gateway is discussed in more detail in Section 5.3. While this strategy maintains HOPPER's properties within individual zones, inter-zone traffic travels between its source and destination zones without HOPPER's protections.

The overlay strategy is similar to the zone-based strategy, but rather than removing the HOPPER headers at the edge of a HOPPER zone, traffic is tunneled between HOPPER zones to create virtual HOPPER-aware links. This preserves the HOPPER headers and allows multiple zones to be operated as a single HOPPER domain, meaning that HOPPER's traffic properties are maintained end-to-end. However, as the virtual HOPPER links may share their underlying physical links with legacy traffic, they can be susceptible to denial-of-service attacks.

Regardless of the brownfield strategy, using the extension options provided by existing protocols (e.g., "next protocol" header fields or extensions), allows HOPPER to be implemented transparently for HOPPER-unaware forwarding elements. This results in a network (zone) where HOPPER-aware forwarding elements verify the HOPPER header, while HOPPER-unaware forwarding elements simply forward the packets without checks. This is similar to how VLAN-unaware routers are able to forward VLAN-tagged traffic and should work out-of-the-box, reserving interference caused by network middleboxes.

Legacy protocols. Because traditional network protocols are designed to make communicating as easy as possible, some of these protocols are at odds with HOPPER's design principles. Specifically, decentralized auto-configuration and auto-discovery protocols, such as Address Resolution Protocol (ARP), do not fit the deny-by-default paradigm. That is, these protocols (i) are explicitly designed to facilitate unplanned communication, and (ii) typically require broadcast communication, allowing each node to contact each other node. This stands in contrast to HOPPER's design goals that limit network permissions to the absolute minimum.

When deploying HOPPER, a threat analysis of these legacy protocols must be made. If the attack surface created by the protocol is sufficiently small, no changes are needed. However, if the protocol exposes a large attack surface, it must be eliminated. This can either be accomplished by statically configuring the normally

auto-configured state (e.g., pre-populating ARP tables), or by replacing the decentralized protocol by a centralized one. We provide an example of the latter approach in Section 5.7.

5.6 Encrypting Packets

HOPPER is not designed to provide confidentiality. However, the key distribution mechanism used by HOPPER can be modified to support per-flow, end-to-end encryption. An encryption key for a flow can be established either by cryptographically combining all flow keys into a single encryption key, or by adding a dedicated encryption tree to the HOPPER key forest. The receiver and flow keys from this tree are distributed as normal, but the root key is not distributed to any forwarding elements. Both methods guarantee that only the sender and receiver of a flow (and the network controller) can calculate the encryption key.

5.7 BOOTSTRAPPING AND PROVISIONING

HOPPER does not specify a specific bootstrapping or key provisioning method. Nonetheless, in this section we explore how a practical key distribution system can be accomplished. Concretely, we do so by extending the functionality of the network controller to distribute HOPPER keys through the network.

As HOPPER is a strict capability-based network architecture, all hosts require a minimum of state before they can start communicating, even with the controller. Moreover, in order to securely exchange keys between the controller and a host, a confidential and authentic channel between them must be established.

The first requirement can be achieved by pre-provisioning each device with flow keys for the flow from the device to the controller. For example, through physical contact following a resurrecting duckling model [47]. To meet the second requirement, we observe that at any point in time, only three entities can generate all valid flow keys for a given flow: (i) *the sender*, which owns the flow keys; (ii) *the receiver*, which can derive them from his receiver keys; and (iii) *the controller*, which can derive any key from the root keys.

Because during communication with the controller the receiver and the controller are the same entity, the controller can uniquely authenticate any device from the packet's HOPPER tag. Thus, the flow keys can be used as shared keying material to bootstrap an authenticated and encrypted channel between any device and the controller. Once this channel is established the device can send a key request to the controller, which verifies the request against an internal policy file and sends the appropriate receiver and flow keys to the device. At this point, the device and controller can also agree on additional keying material to be used to for network recovery when the root key set is compromised.

In the case of hybrid host-forwarding element devices, the controller can also supply the root keys for the forwarding elements in this manner. This allows the network to be cold-started in a hop-by-hop fashion. Moreover, as the controller has a full view of the network, it can also distribute otherwise dynamically discovered information, such as link-layer addresses, together with flow keys.

6 SECURITY ANALYSIS

As introduced in Section 3, a HOPPER network consists of four sets of elements: (i) network links, (ii) forwarding elements, (iii) hosts,

and (iv) the network controller. Motivated by ownership and orchestration centrality, HOPPER leverages the administrator, and by extension the network controller, as the trust root of the system. Hence, the network controller is uncompromised by assumption. This section analyses the consequences of attacks against each remaining network element. Devices which both act as host and forwarding element are susceptible to attacks relating to both devices classes. Moreover, as both hosts and forwarding elements have link access, link-related attacks are also applicable to them.

Network links. An adversary with access to a network link, either wired or wireless, can observe, drop⁵, duplicate, replay, or inject packets on that link. We now discuss the effect of each of these actions on the three security goals listed in Section 3.

Observing packets does not violate any of HOPPER's security goals. Moreover, as each HOPPER MAC uses full-length keys, key extraction—which would facilitate injection attacks—is not feasible.

Dropping packets can be used to mount a denial-of-service (DoS) attack, but only against the compromised link, therefore it does not violate the isolation (G2) or other security goals.

Duplicating or replaying packets can also be used to mount DoS attacks, but can only create traffic towards the destination hosts of flows that traverse a compromised link, and hence does not violate the isolation (G2) or other security goals. Moreover, implementing in-network duplicate suppression can further limit the scope of duplication attacks. Many IoT oriented wireless protocols already include explicit replay protection mechanisms (e.g., Bluetooth Mesh [9], the IETF's IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [3], and IEEE 802.15.4 [43]), and the TSN task group has also specified duplicate elimination for Ethernet networks in IEEE 802.1CB-2017 [1].⁶

If injecting forged packets on a link is possible, each forwarding element will detect and drop each forged packet with probability $1 - 2^{-l|S|}$, where $|l|$ is the length of each MAC in bits, and $|S|$ the number of root keys known to the forwarding element. When using random distribution, $E[|S|] = pn$, with p the key sampling probability, and n the size of the root key set. When using cover-free families, $|S|$ is the size of sets in the family. After injecting c packets, the in-network detection probability of the adversary is $p_{\text{detection, network}} = 1 - 2^{-l|S| \sum_{i=1}^c \{h(i)\}}$, where $h(i)$ is the number of forwarding elements traversed by the i -th packet. This probability quickly tends to 1. The isolation (G2) goals is thus probabilistically satisfied. Further, forged packets will be detected with probability $1 - 2^{-ln} \approx 1$ by each end host. For c packets, the detection probability becomes $p_{\text{detection, host}} = 1 - 2^{-lnc}$ which also quickly tends to 1, satisfying the least privilege (G1) and authentication (G3) goals.

Forwarding elements. When an adversary compromises one or more forwarding elements, he learns part of the root key, leading to the modified detection probabilities $p'_{\text{detection, network}} = 1 - 2^{-l|S'| \sum_{i=1}^c \{h(i)\}}$ and $p'_{\text{detection, host}} = 1 - 2^{-ln'c}$, where n' is the number of uncompromised root keys. When random root key distribution is used, the expected value of n' can be calculated using the equations in Section 4.4. When using a (m,n,t)-cover-free

⁵Or jam, in the case of wireless links.

⁶The duplicate frame elimination specified IEEE 802.1CB-2017 is not explicitly designed as a security mechanism, but rather as part of a reliability mechanism.

family, $n' \geq 1$ as long as less than t forwarding elements have been compromised. $|S'|$ is the number of uncompromised root keys known to a forwarding element. For random root key distribution $E[|S'|] = pn'$. When using a cover-free family, $|S'| \geq 1$ as long as less than t forwarding elements have been compromised. As both $p'_{\text{detection, network}}$ and $p'_{\text{detection, host}}$ tend to 1 as long as not all root keys are compromised, least privilege (G1), isolation (G2), and authentication (G3) remain probabilistically satisfied.

Hosts. If an adversary compromises a host, he obtains access to the receiver and flow keys stored on that device. The flow keys can be used to send arbitrary packets towards their corresponding receivers, but not towards other hosts, thus not violating the least privilege (G1) and isolation (G2) goals.

When an adversary has compromised multiple hosts, and can communicate between them, he can exchange the compromised receiver and flow keys between these hosts. The receiver keys allow the compromised hosts to generate and send traffic towards each other. The flow keys allow each host to generate traffic on the logical flows corresponding to the compromised flow keys, and to send that traffic towards the receiver of these flows. Neither of these attacks violates the least privilege (G1), isolation (G2), or authentication (G3) goals. Moreover, in networks where topology based packet filtering is possible, the significance of the second attack can be further reduced.

6.1 Mitigation of Today's Common IoT Threats

Deploying HOPPER on a network can mitigate common threats against IoT devices seen today. We discuss three examples.

Botnets. Since the Mirai botnet demonstrated the destructive power of large-scale IoT attacks, botnets have been a major IoT security concern [5, 39]. Because IoT botnets are usually constructed by finding inadvertently vulnerable devices through network scans, HOPPER's whitelist-based approach provides a strong defense against device compromise. Moreover, because the number of destinations a HOPPER-enabled host can send traffic to is severely limited, HOPPER devices are not attractive botnet members.

Worms. Whereas the focus of IoT attackers used to be on constructing botnets to perform DoS attacks, their focus has shifted towards IoT worms, such as cryptolockers [18, 19, 39]. Because HOPPER's nano segmentation minimizes the number of potential victims an infected host can reach, the spreading of worms is severely restricted. This stands in contrast to traditional segmentation methods where worms can spread freely within each segment.

Lateral movement. In many attacks against IoT devices, the compromised devices are not the primary attack target, but are used as entry points for further lateral movement in the network [39]. Similarly to how HOPPER defends against worms, HOPPER's nano segmentation significantly reduces the number of new attack vectors available to an attacker after compromising a device.

7 IMPLEMENTATION AND EVALUATION

We split the evaluation of HOPPER into four subsections. First, we perform a scalability analysis of the protocol. Second, we evaluate the performance of a HOPPER-enabled end host. In order to verify

that HOPPER is suitable for constrained environments, we perform this evaluation on IoT-class hardware. Third, we evaluate a HOPPER forwarding element by implementing HOPPER forwarding logic on a low-end network appliance and benchmarking this appliance by physically connecting it to an emulated network. Finally, we briefly report on our implementation of a HOPPER controller and confirm interoperability between HOPPER's network elements.

7.1 Scalability

We evaluate the scalability of HOPPER on the three active elements of a HOPPER network: (i) hosts, (ii) forwarding elements, and (iii) the network controller. We also discuss bandwidth overhead.

Hosts. Hosts need to store one set of receiving keys and one set of flow keys for each outgoing flow. More formally, hosts need to store $n \cdot (f + 1) \cdot |k|$ bytes, where n is the size of the root key set, f the number of outgoing flows of that host, the constant "1" represents the receiver key set, and $|k|$ is the size of an individual key. As shown in Table 1, n is not a function of the size of the deployment, but rather of the number of compromised forwarding elements that can be tolerated. The computational overhead per packet is constant in the size of the deployment: for each incoming or outgoing packet n MACs must be calculated, and for incoming packets an additional n key derivations must be performed.

Forwarding elements. Assuming random root key distribution, forwarding elements need to store $p \cdot n \cdot |k|$ bytes, where p is the key sampling probability. Also here computational overhead per packet is constant in the size of the deployment: each forwarded packet requires $2 \cdot p \cdot n$ key derivations and $p \cdot n$ MACs calculations.

Network controller. For each rekeying event, the network controller must distribute new root keys to the forwarding elements, receiver keys to the receiving hosts, and flow keys to the sending hosts. However, rekeying events are rare and take place on the control rather than data plane. Therefore the performance of the network controller is not critical to HOPPER's overall performance.

Bandwidth overhead. Each packet must carry a tag of length $n \cdot l$ bytes, where l is the length of an individual MAC. As discussed in Section 5.2, n and l are constant in the size of the deployment.

7.2 End Host Performance

We evaluate HOPPER's end host performance using two ST Nucleo-F439ZI development boards. These boards carry a STM32F439ZI microcontroller unit (MCU) which runs at 180 MHz, representing mid-range IoT devices. The MCU provides hardware cryptography acceleration, a common feature on IoT MCUs to support link-layer encryption. The boards also have on-board 10/100 Mbps Ethernet, facilitating evaluation in an isolated and reproducible environment.

We implement two versions of HOPPER: one for UDP/IP, as described in Section 4, and one for Ethernet, where we use the source address, destination address and L3 protocol as flow identifier.

Both HOPPER versions extend the open-source lwIP (lightweight IP) protocol stack [20] to support HOPPER tag generation and verification. Our HOPPER implementation for Ethernet defines a new EtherType and places tags directly behind the Ethernet header. The

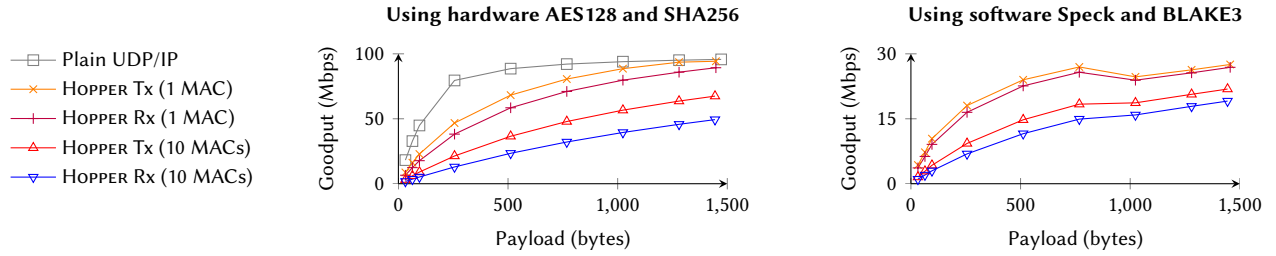


Figure 5: Plain UDP/IP vs. HOPPER on UDP/IP end host goodput.

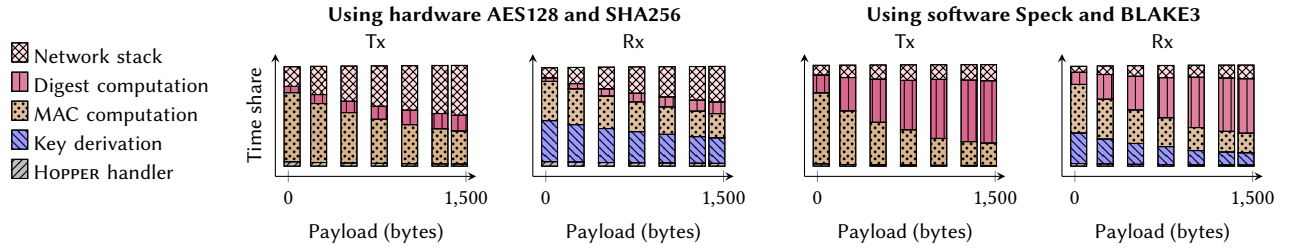


Figure 6: Simplified end host load profile for HOPPER on UDP/IP using 10 MACs per tag.

implementation for UDP/IP defines a new IP protocol number and places tags between the IP and UDP headers.

To perform the evaluation, the two boards are directly connected using an Ethernet cable. During each experiment one board continuously generates UDP datagrams (or Ethernet frames) with a dummy payload. The other board receives these datagrams (or frames) and processes their headers. No higher layer processing is performed.

We evaluate each HOPPER implementation using both hardware-accelerated and software crypto. In the hardware setting we use AES128 as block cipher and SHA256 as hash algorithm. In the software setting we use the Speck software cipher [8] with 64-bit blocks, and use BLAKE3 [37] as hash algorithm. PRFs are implemented using the block cipher in ECB mode, keys are always 128 bits long.

The hardware ciphers were chosen because our evaluation platform provides hardware acceleration for them. Different software ciphers were selected for performance reasons: using Speck instead of an AES software implementation [30] resulted in a throughput improvement of roughly 5 to 100 %, depending on the setting. Similarly, using BLAKE3 instead of software SHA256 [36] resulted in throughput gains of 35 to 100 %.

Further, we evaluate each HOPPER implementation in each cryptography setting in two tag composition settings. The first tag setting uses tags consisting of ten 16-bit MACs calculated using CBC-MAC over a hash of the packet payload.⁷ We note that hash-then-MAC is a provably secure construct [10]. The second tag setting uses tags consisting of a single 128-bit MAC for hardware crypto and a single 64-bit MAC⁸ for software crypto.

Because the results for HOPPER on Ethernet and on UDP/IP are near identical, we only discuss the latter in this section. The results for Ethernet are shown in Appendix A.

Hardware cryptography. We see in Fig. 5 that when using hardware crypto and one MAC, HOPPER packets can be generated (Tx, 94 Mbps) and processed (Rx, 89 Mbps) at 99 % and 93 % the rate of plain UDP/IP (96 Mbps), respectively. The header space required by HOPPER leads to a slight reduction in maximum payload compared to plain UDP/IP. When using ten MACs, we observe packet generation (67 Mbps) and processing (49 Mbps) rates of 70 % and 52 % the rate of plain UDP/IP, respectively. As can be seen from the load profiles in Fig. 6, the discrepancy between the results for incoming and outgoing packet is caused by the flow-key derivations required on the receiving endpoint. Key caching or probabilistic MAC evaluation could mitigate this performance gap, though the latter would lead to a reduction in security. Figure 6 also shows that the majority of HOPPER’s workload consists of cryptographic operations. It is worth noting that when using hardware crypto acceleration, these operations are executed by the MCU’s cryptography peripheral, leaving the core free to perform other tasks.

Software cryptography. As shown in Fig. 5, when using software cryptography, throughputs between 28 Mbps and 19 Mbps are achieved. This corresponds to 29 and 20 % the rate of plain UDP/IP. The dip in performance at 1024 bytes is inherited from BLAKE3. As shown in Fig. 6 the performance is dominated by the hash function.

Given the low data-rates typically found in IoT and industrial applications⁹, we find the obtained performance results to be satisfactory. Although the use of software cryptography results in a significant performance penalty, most MCUs targeted at IoT applications already provide hardware cryptography acceleration to

⁷We use CBC-MAC, as the hash algorithms ensure constant input sizes.

⁸64 bits corresponds to native Speck block size on a 32-bit architecture. Brute forcing a 64-bit MAC using minimal HOPPER packets over a gigabit link would, in expectation, take on the order of 10^5 years.

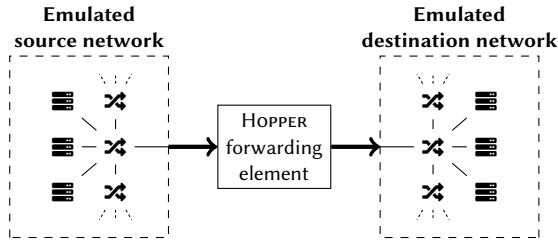


Figure 7: Measurement setup used to benchmark the HOPPER forwarding element. Bold arrows indicate physical cables.

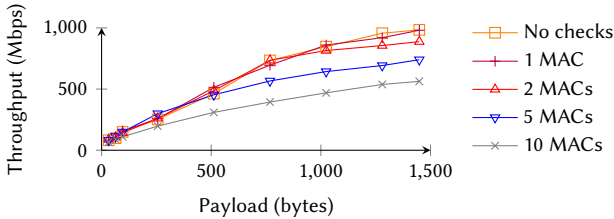


Figure 8: HOPPER forwarding element performance for different numbers of MACs checked.

support link-layer encryption. Hence, we expect software cryptography to only be used in the most constrained settings, which typically feature low throughput requirements.

Binary size. The MCU binaries for the tests without HOPPER, with hardware crypto, and with software crypto were 75, 100, and 113 kB in size respectively. These binaries were compiled for optimal performance. When compiling for an optimized binary footprint, the sizes reduce to 61, 78, and 89 kB, respectively. Doing so results in a throughput reduction of 0 to 15 %, depending on the setting.

7.3 Forwarding Element Performance

We implement a HOPPER forwarding element using a PC Engines APU2D4 system board. The APU2 is a popular platform for low-end network appliances, and with a quad-core AMD GX-412TC CPU running at 1 GHz and 4 GB DDR3-1333 memory, its performance is roughly comparable to a Raspberry Pi 4. The board also has 3 Gigabit Ethernet interfaces.

We implement the HOPPER forwarding logic for UDP/IP using DPDK [49] and OpenSSL [38]. For each packet the forwarding element receives, it verifies the HOPPER tag using the root keys with which it was provisioned. If successful, the packet is forwarded without further processing. If the verification fails the packet is dropped. The forwarding element uses AES128 as block cipher and SHA256 as hash algorithm. PRFs are implemented using AES128 in ECB mode. We use HOPPER tags with 10 MACs of 16 bits each.

⁹For example, the Ethernet Advanced Physical Layer (Ethernet-APL), a recently developed physical layer for Ethernet which is targeted specifically at industrial applications, operates at only 10 Mbit/s [2].

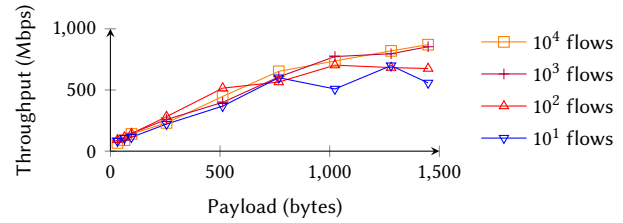


Figure 9: HOPPER forwarding element performance for different emulated network sizes. Two MACs are verified.

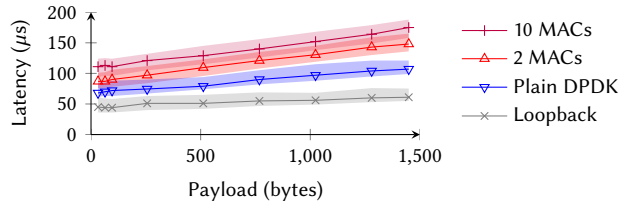


Figure 10: Median HOPPER forwarding element latency. *Loopback* shows the base latency measured without forwarding element, *Plain DPDK* uses a forwarding element that forwards all traffic without any checks or packet parsing. The shaded areas show the 25th and 75th percentiles.

Test traffic generation. We create a gopacket [24] implementation of HOPPER. For each experiment, we first generate a pool of randomized 5-tuples. We then generate HOPPER packets by randomly sampling (with replacement) 5-tuples from this pool, and adding a random payload. The generated packets are then stored in a pcap file to be replayed using tcpreplay during the experiment.

Throughput. To measure the forwarding element’s throughput, we physically connect it to two hosts that each emulate a network, as illustrated in Fig. 7. We then let the source host transmit 10^6 packets at line rate and measure the incoming traffic volume at the receiver host. We repeat this experiment multiple times varying the number of root keys the forwarding element is provisioned with. We also perform a baseline measurement in which the forwarding element immediately forwards all packets without parsing them.

We see in Fig. 8 that for small packets, or when verifying only one MAC, using HOPPER does not meaningfully reduce the forwarding element’s throughput. When verifying 2 MACs, performance is reduced by approximately 10 % to 890 Mbps. Verifying all 10 MACs results in a throughput of 565 Mbps.

Scalability. In Section 7.1, we showed that scaling the size of a HOPPER deployment does not influence the performance of the forwarding elements. We now verify this result by varying the size of the 5-tuple pool (which effectively corresponds to changing the size of the emulated network), and rerunning the throughput experiments. For these experiments we configure the forwarding element to verify 2 MACs per packet.

Figure 9 shows that, somewhat counterintuitively, increasing the number of network flows *increases* the throughput of the forwarding element. This is a side effect of the hash-based receiver

side scaling used by DPDK: the larger the set of flows, the more uniformly the workload is spread across the four processor cores. We verified this hypothesis by running the same experiment using only a single processor core, and we observe that doing so results in identical throughput regardless of the traffic mix (not shown).

Latency. We evaluate the latency added to the forwarding element by the HOPPER checks. We again use the test setup shown in Fig. 7, but now emulate both the source and destination network on the same host, using different host interfaces. In each test setting, we transmit 10^3 packets, one at a time, and we capture the kernel-generated send and receive timestamps of each packet using tcpdump. We also measure two baselines: one using a loopback cable which bridges the source and destination interface on the emulation host, and one in which we configured DPDK to directly forward all packets without performing any checks or parsing.

The results of the latency measurements are shown in Fig. 10. We see that verifying two HOPPER MACs per packet results in a median latency increase of 20 to 40 μ s, depending on the packet size. The increase in latency overhead with increasing packet sizes is a direct consequence of SHA256 hash that is calculated over the packet payload. Verifying 10 MACs instead of 2, adds an additional 20 μ s to the median latency, regardless of the packet size. Further, we observe only a minor effect of HOPPER on latency jitter, measuring an average standard deviation of 17, 17, 20, and 22 μ s for the loopback, plain DPDK, 2 MACs, and 10 MACs settings, respectively.

7.4 Controller Implementation

We implemented a proof of concept HOPPER controller that allows an administrator to specify which flows are whitelisted using a policy file and dynamically distributes flow keys on request. We also extended our HOPPER implementations to automatically request missing flow keys from the controller and performed a functional evaluation of interoperability and the bootstrapping mechanism described in Section 5.7. Because the speed of the initial key distribution is not critical, we did not benchmark the controller.

8 RELATED WORK

IoT gateways. A common method to secure IoT networks is by deploying a specialized IoT firewall, often called a *gateway*. Past proposals include DeadBolt [29], IoT Sentinel [35], IoTSec [55], and a design by Simpson et al. [46]. Concretely, Simpson et al. explore how to protect and isolate vulnerable devices and how to securely apply patches; IoT Sentinel and IoTSec focus on automated identification of compromised devices; and DeadBolt aims to enforce the application of good security practices on IoT devices while mediating traffic to non-complying devices.

These proposals all consist of a security proxy that all network traffic must be routed through. While IoTSec envisions security proxies to be instantiated at various points in the network with overlay routing forcing traffic through them, the others effectively require a physical or overlay network with a star topology.

Apple recently started including a basic version of an IoT firewall in their *HomeKit* platform [6]. The vendors of HomeKit devices must supply a manifest file stating which connections their product is supposed to establish, and by default all other communication flows are blocked. Closely related to this, RFC 8520 [33] standardizes the

description of the expected network behavior of devices, facilitating deny-by-default policies such as the one implemented by *HomeKit*.

While suitable in some environments, approaches as discussed above are not suitable for industrial networks. That is, indirect routing introduces a single point of failure to the data plane, increased latencies, and increased network overhead. Moreover, aggregating and forcing network traffic along an indirect path directly counteracts the properties achieved by TSN. The proposals listed above also do not achieve the source-authentication properties of HOPPER.

Capability-based and Deny-by-Default networking. HOPPER deploys capability-based networking in order to achieve per-device nano segmentation. While our work is (to the best of our knowledge) the first to apply this concept to the IoT setting, capability-based networking, and more generally, deny-by-default networking, have been considered in other settings.

Concretely, a first set of past work considers the use of capabilities for DoS protection on the Internet. Proposals in this space include a design by Anderson et al. [4], SIFF [53], and TVA [54]. As these proposals are designed to provide DoS protection, they only prevent volumetric attacks, but still allow low volume traffic flows to reach hosts. Hence, they do not prevent lateral movement.

Around the same time, Ballani et al. [7] proposed to propagate whitelists through the Internet using a mechanism similar to BGP, dropping all non-whitelisted traffic. While this scheme does prevent all unwanted traffic from reaching an end host, it still assumes an inter-domain setting, which is inapplicable to IoT networks. For example, it requires large amounts of reachability information to be transferred between forwarding elements.

A third set of work considers enterprise networks, most notably SANE [14] and Ethane [13]. SANE uses capabilities in the form of authenticated source routes, which are checked at each switch. However, because routes in (wireless) IoT networks can be non-predictable (see Section 2.3), it is not a suitable mechanism for IoT settings. Ethane uses a complementary approach: each switch stores a whitelist of permitted traffic. High control overhead and strong assumptions on the network architecture render this mechanism impractical in (constrained) IoT settings. Further, neither SANE nor Ethane provide packet authentication.

Fieldbus authentication. There are a number of proposals that add authentication to industrial fieldbuses [15, 40, 50, 51]. However, these proposals do not provide least privilege (G1) or isolation (G2) and requires device to be pre-provisioned with per-flow keys.

9 CONCLUSION

New trends in industrial automation are challenging the ways in which industrial networks are secured. In order to address these challenges, this work presents HOPPER, a nano segmentation scheme for industrial IoT deployments. By deploying capability-based and deny-by-default networking, HOPPER extends the protections typically only provided at the edge of the network to the entire network fabric, realizing per-device network segmentation. Moreover, HOPPER was designed to be compatible with the diverse constraints encountered in IoT deployments, allowing the same protocol to be deployed across a wide range of settings.

Contrary to the ossification seen in traditional networks, the IoT networking stack is still in flux. This represents a unique, but ephemeral, opportunity to embed security-by-design into the core of the IoT stack. HOPPER accomplishes this by extending the broadly accepted principle of least privilege across the network. Moreover, as the IoT's momentum is ever increasing, the need for the strong but flexible defenses that HOPPER provides has never been so critical.

ACKNOWLEDGMENTS

We would like to thank Markus Legner, Giacomo Giuliani, and Huayi Duan for the helpful discussions and advice that improved this research. Piet De Vaere would also like to thank Ladina Hausmann for her continued support of this work. We further thank shepherd Mu Zhang and the anonymous reviewers for their insightful feedback and suggestions.

REFERENCES

- [1] 2017. IEEE Standard for Local and metropolitan area networks—Frame Replication and Elimination for Reliability. *IEEE Std 802.1CB-2017* (2017).
- [2] 2020. IEEE Standard for Ethernet - Amendment 5: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors. *IEEE Std 802.3cg-2019* (2020).
- [3] R. Alexander, A. Brandt, J.P. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550.
- [4] T. Anderson, T. Roscoe, and D. Wetherall. 2004. Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Computer Com. Review (CCR)* (2004).
- [5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium*.
- [6] Apple inc. 2020. HT210544: Use routers secured with HomeKit.
- [7] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. 2005. Off by Default!. In *ACM HotNets*.
- [8] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. 2013. The SIMON and SPECK Families of Lightweight Block Ciphers. *Cryptology ePrint Archive*, Report 2013/404. <https://eprint.iacr.org/2013/404>.
- [9] Bluetooth Special Interest Group. 2019. Mesh Profile Specification 1.0.1.
- [10] D. Boneh and V. Shoup. 2020. A Graduate Course in Applied Cryptography.
- [11] C. Bormann, M. Ersue, and A. Keränen. 2014. Terminology for Constrained-Node Networks. RFC 7228.
- [12] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. 1999. Multicast security: a taxonomy and some efficient constructions. In *IEEE INFOCOM*.
- [13] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. 2007. Ethane: Taking Control of the Enterprise. *ACM SIGCOMM CCR* (2007).
- [14] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. 2006. SANE: A Protection Architecture for Enterprise Networks.. In *USENIX Security Symposium*.
- [15] J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, and M. Ochoa. 2017. Legacy-compliant data authentication for industrial control system traffic. In *International Conference on Applied Cryptography and Network Security*. Springer.
- [16] CISCO. 2018. IT/OT Convergence. https://www.cisco.com/c/dam/en_us/solutions/industries/manufacturing/ITOT-convergence-whitepaper.pdf.
- [17] Cisco Systems and Rockwell Automation. 2008. Ethernet-to-the-Factory 1.2 Design and Implementation Guide.
- [18] Dragos, inc. 2017. TRISIS: Analyzing Safety System Targeting Malware.
- [19] Dragos, inc. 2020. EKANS Ransomware and ICS Operations.
- [20] A. Dunkels. 2001. Design and Implementation of the lwIP TCP/IP Stack. *Swedish Institute of Computer Science* (2001).
- [21] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. 2011. Efficient network flooding and time synchronization with Glossy. In *ACM/IEEE IPSN*.
- [22] OPC Foundation. 2021. OPC UA Online Reference, Section 7.3.2: OPC UA UDP. <https://reference.opcfoundation.org/v104/Core/docs/Part14/7.3.2/>.
- [23] E. Gilman and D. Barth. 2017. *Zero Trust Networks*. O'Reilly Media.
- [24] Google. 2021. GoPacket. <https://github.com/google/gopacket>.
- [25] C. Hung and W. Hsu. 2018. Power Consumption and Calculation Requirement Analysis of AES for WSN IoT. *Sensors* (2018).
- [26] IEEE 802.1. 2020. TSN Task Group. <https://1.ieee802.org/tsn/>.
- [27] International Electrotechnical Commission. 2020. IEC 62443-3-2:2020 Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design. Technical Standard.
- [28] K. Zetter, WIRED. 2016. Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid.
- [29] R. Ko and J. Mickens. 2018. DeadBolt: Securing IoT Deployments. In *ACM/IRTF ANRW*.
- [30] kokke. 2021. Tiny AES. <https://github.com/kokke/tiny-AES-c>.
- [31] M. Krotofil, A. A. Cárdenas, B. Manning, and J. Larsen. 2014. CPS: Driving Cyber-Physical Systems to Unsafe Operating Conditions by Timing DoS Attacks on Sensor Signals. In *Proc. of Annual Computer Security Applications Conference*.
- [32] R. Langner. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy* (2011).
- [33] E. Lear, R. Droms, and D. Romascanu. 2019. Manufacturer Usage Description Specification. RFC 8520.
- [34] L. Lo Bello and W. Steiner. 2019. A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proc. IEEE* (2019).
- [35] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *IEEE ICDCS*.
- [36] A. Mosnier. 2019. SHA-2 implementation. <https://github.com/amosnier/sha-2>.
- [37] J. O'Connor, J.P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn. 2020. BLAKE3: one function, fast everywhere. <https://blake3.io>.
- [38] OpenSSL Software Foundation. 2021. OpenSSL. <https://www.openssl.org/>.
- [39] Paloalto Networks. 2020. 2020 Unit 42 IoT Threat Report. <https://unit42.paloaltonetworks.com/iot-threat-report-2020/>.
- [40] A. Radu and F.D. Garcia. 2016. LeiA: A lightweight authentication protocol for CAN. In *European Symposium on Research in Computer Security*. Springer.
- [41] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In *IEEE Symposium on Security and Privacy*.
- [42] R. Safavi-Naini and H. Wang. 1998. New results on multi-receiver authentication codes. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.
- [43] N. Sastry and D. Wagner. 2004. Security Considerations for IEEE 802.15.4 Networks. In *ACM WiSec*.
- [44] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich. 2011. The Evolution of Factory and Building Automation. *IEEE Industrial Electronics Magazine* (2011).
- [45] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and j. Nieminen. 2012. How low energy is bluetooth low energy?. In *IEEE WCNCW*.
- [46] A. K. Simpson, F. Roesner, and T. Kohno. 2017. Securing vulnerable home IoT devices with an in-hub security manager. In *IEEE PerCom Workshops*.
- [47] F. Stajano and R. Anderson. 2000. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Security Protocols*. Springer.
- [48] D. Temple-Raston. 2021. A "Worst Nightmare" Cyberattack: The Untold Story Of The SolarWinds Hack. National Public Radio.
- [49] The Linux Foundation. 2021. Data Plane Development Kit. <https://dpdk.org/>.
- [50] P. P. Tsang and S. W. Smith. 2008. YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems. In *IFIP International Information Security Conference*. Springer.
- [51] A. Van Herrewege, D. Singelee, and I. Verbauwhede. 2011. CANAuth—a simple, backward compatible broadcast authentication protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography*.
- [52] M. Wollschlaeger, T. Sauter, and J. Jasperneite. 2017. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine* (2017).
- [53] A. Yaar, A. Perrig, and D. Song. 2004. SIFF: a stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*.
- [54] X. Yang, D. Wetherall, and T. Anderson. 2005. A DoS-limiting network architecture. *ACM SIGCOMM Computer Comm. Review (CCR)* (2005).
- [55] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. 2015. Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. In *ACM HotNets*.

A RESULTS FOR HOPPER ON ETHERNET

Figures 11 and 12 show the results for our Ethernet HOPPER implementation. They are functionally equivalent to Figures 5 and 6.

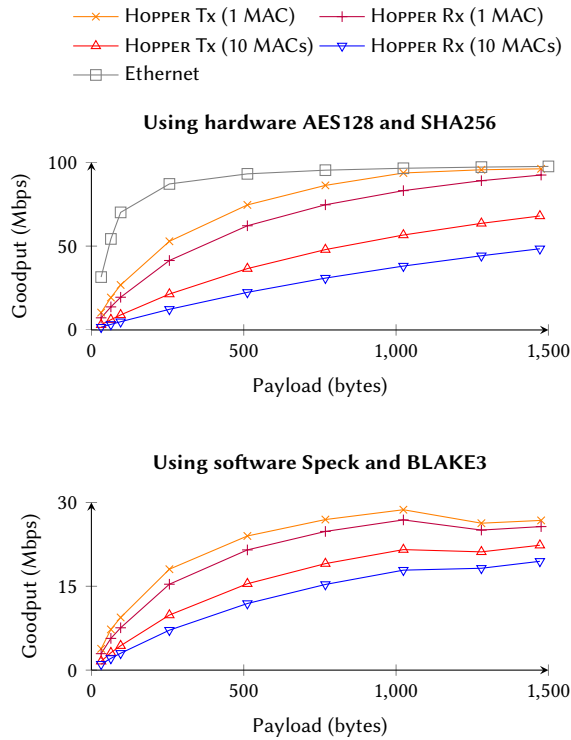


Figure 11: Plain Ethernet vs. HOPPER on Ethernet goodput performance.

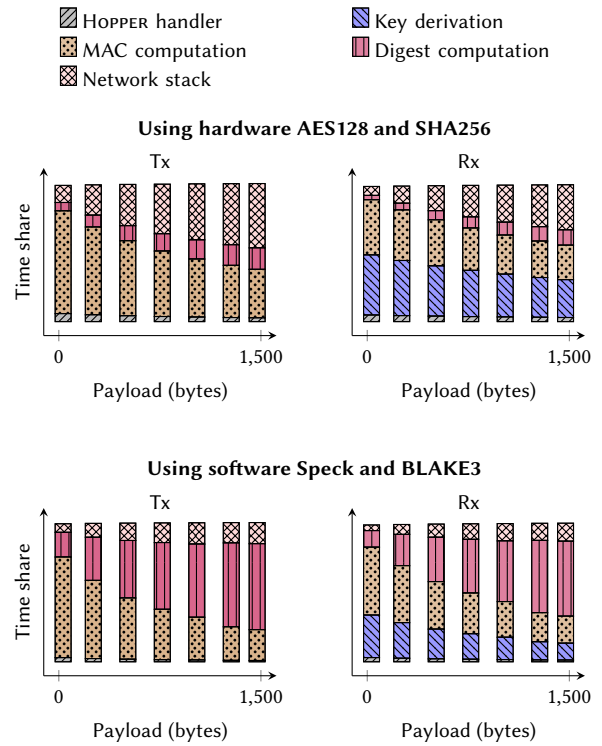


Figure 12: Simplified load profile for HOPPER on Ethernet using 10 MACs per tag.