GrandDetAuto: Detecting Malicious Nodes in Large-Scale Autonomous Networks

Tigist Abera
Technical University Darmstadt
Darmstadt, Germany
tigist.abera@trust.tu-darmstadt.de

Ferdinand Brasser
Technical University Darmstadt
Darmstadt, Germany
ferdinand.brasser@trust.tu-darmstadt.de

Lachlan Gunn
Aalto University
Greater Helsinki, Finland
lachlan.gunn@aalto.fi

Patrick Jauernig
Technical University Darmstadt
Darmstadt, Germany
patrick.jauernig@trust.tu-darmstadt.de

David Koisser
Technical University Darmstadt
Darmstadt, Germany
david.koisser@trust.tu-darmstadt.de

Ahmad-Reza Sadeghi
Technical University Darmstadt
Darmstadt, Germany
ahmad.sadeghi@trust.tu-darmstadt.de

ABSTRACT

Autonomous collaborative networks of devices are rapidly emerging in numerous domains, such as self-driving cars, smart factories, critical infrastructure, and Internet of Things in general. Although autonomy and self-organization are highly desired properties, they increase vulnerability to attacks. Hence, autonomous networks need dependable mechanisms to detect malicious devices in order to prevent compromise of the entire network. However, current mechanisms to detect malicious devices either require a trusted central entity or scale poorly.

In this paper, we present GrandDetAuto, the first scheme to identify malicious devices efficiently within large autonomous networks of collaborating entities. GrandDetAuto functions without relying on a central trusted entity, works reliably for very large networks of devices, and is adaptable to a wide range of application scenarios thanks to interchangeable components. Our scheme uses random elections to embed integrity validation schemes in distributed consensus, providing a solution supporting tens of thousands of devices. We implemented and evaluated a concrete instance of GrandDetAuto on a network of embedded devices and conducted large-scale network simulations with up to 100 000 nodes. Our results show the effectiveness and efficiency of our scheme, revealing logarithmic growth in run-time and message complexity with increasing network size. Moreover, we provide an extensive evaluation of key parameters showing that GrandDetAuto is applicable to many scenarios with diverse requirements.

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**; **Mobile and wireless security**; • **Networks** → *Network security*.

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471868

KEYWORDS

malicious device detection, autonomous networks, security

ACM Reference Format:
Tigist Abera, Ferdinand Brasser, Lachlan Gunn, Patrick Jauernig, David Koisser, and Ahmad-Reza Sadeghi. 2021. GrandDetAuto: Detecting Malicious Nodes in Large-Scale Autonomous Networks. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3471621.3471868

1 INTRODUCTION

The growing trend towards the Internet of Things (IoT) and Autonomous Systems allows connected devices to collaborate, enabling more efficient as well as new applications. This opens up new opportunities in many domains, from self-driving cars and smart factories to critical infrastructure. Various industries are motivated by higher efficiency and increased flexibility, which can be achieved by connecting devices within individual factories as well as by interconnecting facilities collaborating within a supply-chain [68]. Other industry branches, like the automotive and associated industries, strive to increase safety through connection and collaboration, e.g., cars sharing information about potential hazards [41]. The extensive efforts to standardize vehicle communications by major industry leaders shows the relevance of this trend, such as the cellular network-based C-V2X [2] and the WiFi extension standard 802.11p [39], which, for instance, Volkswagen announced to support in *all* 2020 Golf 8 [70]. However, despite these advantages autonomous systems bear various security risks. Hence, it is important to develop solutions for these challenging scenarios. In particular, in safety-related scenarios, malicious devices can cause tremendous damage and threaten human life.

To secure such systems many proposals rely on a central authority [6, 10, 15, 20, 37, 67]. However, a centralized solution constitutes a single point of failure, implying unrealistic requirements on the central authority: (1) The availability of the authority must be guaranteed at all times, i.e., the entire system must have continuous and reliable connectivity to it. This is hard to guarantee in many practical systems, e.g., with freely moving nodes. (2) A central authority is an attractive attack target, exposing it to a wide range of attacks. Any successful attack will corrupt its integrity and/or

availability, i.e., make the central authority fail. There are many real world examples how centralization of authority can be detrimental, like the compromised DigiNotar PKI (public key infrastructure) issuing fraudulent certificates for Google, Microsoft and CIA websites [75], the attack on Ukraine's power grid by compromising centrally operated Industrial Control Systems [80], or the DDoS attack on DynDNS bringing down major websites (incl. PayPal, CNN and Amazon) in parts of Europe and the US [76]. These examples show that even the most sophisticated defense mechanisms aiming to protect central services can be circumvented. Further, when multiple (mutually distrusting) stakeholders are involved, it is difficult to jointly agree on a party that acts as the trusted authority. For instance, different car manufacturers or cellular network equipment providers, which in many cases do not inherently trust each other, will not easily agree on an overarching authority with the power to control all devices.

Problem. Strongly reducing or fully eliminating the role of the central party in connected systems seems very appealing; yet, it requires the connected devices to collaborate and share information in a broadly autonomous fashion. Consequently, interdependencies within the network will increase the threat that malicious devices could pose on the entire system. Increasingly interconnected devices, including modern vehicles [17, 47, 58], industrial facilities [14, 18, 28], critical infrastructure [26, 42, 63], and even medical devices [64] have been the targets of attacks. In particular, a single malicious device could cause other devices to deviate from the correct behavior; for instance, influencing the routing of other cars by transmitting false traffic information [69]. Hence, large autonomous networks must also be able to identify faulty or malicious devices in order to react to attacks. It is paramount to prevent a (partial) compromise of the network from impairing the correct function of the overall system.

Existing defense strategies. Attack *detection* methods can uncover ongoing attacks, enabling more sophisticated reaction policies, like the recovery of a compromised device [56] to prevent an adversary taking over the network. Outlier detection is used in Wireless Sensor Networks to identify outliers on aggregated sensor data, which may be caused by malicious attacks [82]; yet, many directly rely on a central entity. Thus, they are inapplicable to autonomous systems without central authority. Approaches that do not rely on such a central entity [12, 53] do not scale for large networks commonly encountered in autonomous systems. There are collaborative intrusion detection approaches that distribute data acquisition across the network [13]. Yet, they either assume a central authority for decision-making [20, 67], or assume only few, sparsely distributed malicious devices in the network [35, 43, 81]. For real-world scenarios, this is hard to guarantee as adversarial nodes can collaborate to gain the majority in a group of nodes. Other approaches, such as swarm attestation [6, 10, 15, 37], provide an integrity proof for the whole network to a central verifying and trusted entity. Thus, they are inapplicable to autonomous systems without central authority. We elaborate further on these approaches in Section 9.

Goals and Challenges. Designing an efficient scheme for detecting and identifying malicious nodes/devices in a connected autonomous system faces us with a number of challenges. The overarching challenge is scalability: A naive solution in which each node individually performs monitoring and validation of potentially all other nodes is inefficient, especially with resource-constrained embedded devices. Thus, the naive solution does not scale. Instead, an appropriate scheme needs to combine local monitoring with efficient and scalable decision-making. For this, we derive three key challenges.

Challenge 1: Flexible and adaptive detection of malicious devices. To be able to identify compromised devices, a practical mechanism is needed to validate whether a device is in a good state, i.e., behaving as expected. There are various approaches to achieve this, each coming with a set of advantages and disadvantages. We discuss this in Section 4.

Challenge 2: Establishment of a network-wide shared state. In addition to having a scheme for device state integrity validation, a common state among the nodes is needed. However, agreeing on a common state efficiently among all individual nodes is particularly difficult in large-scale networks.

Challenge 3: Resiliency. In an autonomous system, the monitoring as well as the decision-making is generally distributed among the nodes. To guarantee resiliency, the final decision must not only rely on monitoring results raised by an individual node, but on a distributed agreement. However, a distributed decision-making scheme must be carefully designed to avoid introducing additional attack vectors.

Contributions. In this paper, we present GrandDetAuto, a novel distributed adversary detection scheme for large-scale networks. The design of GrandDetAuto is generic and modular. It combines schemes for integrity validation of devices' states with schemes for distributed election and consensus in a novel way, while each of these modules can be instantiated with different primitives that fit the requirements posed by the corresponding application. More precisely, any device may blame another device for being malicious by providing a proof that the state integrity of that device is violated. This proof will then be verified by a randomly and autonomously selected jury (a subset of devices), which in turn finds a consensus on whether the proof is valid. Because the jury-size is fixed but configurable, the consensus overhead remains constant independent of network size (aside routing).

Our main contributions include:

- GrandDetAuto is the first efficient and dependable scheme to allow a system of collaborating entities without a central authority to detect its compromised parts by distributing integrity validation schemes via random elections leading to Byzantine fault-tolerant decisions (Section 3).
- GrandDetAuto is highly flexible since its components can be instantiated by various schemes for integrity validation, random elections, and consensus protocols (Section 4).
- We introduce a novel distributed election scheme, inspired by Proof-of-Elapsed-Time [36], to randomly elect a *group* of representatives in the network.

- We implemented a GrandDetAuto prototype in the context of smart traffic based on the aforementioned election scheme, Practical Byzantine Fault Tolerance (PBFT) [16] and remote attestation (Section 5) using an ARM platform with TrustZone.
- Being a distributed system, GrandDetAuto's efficiency and security relies on a suitable choice of key parameters, which we thoroughly analyze and evaluate (Section 6). Further, we developed a large-scale network simulation with tens of thousands of devices and demonstrate GrandDetAuto's scalability through extensive evaluation (Section 7).

2 SYSTEM MODEL

We consider large distributed autonomous systems; specifically, a network of connected devices n_1, ..., n_i that collaborate with each other to perform complex tasks. We use the terms device or node interchangeably in the following. Nodes may join and leave the network; yet, the list of devices participating in the network is known[1] In order to collaborate by coordinating their actions, the individual entities of the overall system need to exchange information, such as status updates and sensor readings, which is often critical for the correct behavior of the overall system. In a smart traffic scenario, for example, false position information may lead to vehicles crashing into each other.

All devices are mutually distrusting and there is no trusted central entity or external coordinating operator on which the network must rely. GrandDetAuto has a generic design and does not assume any specific security framework or security hardware. However, depending on the instantiation in practice, it can utilize security architectures, such as Trusted Execution Environments (TEEs) for random election and integrity validation, as we present in Section 5.

2.1 Adversary Model and Assumptions

The adversary's goal is to influence the collaboration between honest nodes by manipulating the data sent to other devices. We make the following assumptions about the adversary's capabilities. The adversary $\mathcal{A}$ is able to compromise and coordinate a subset of devices in the system. We denote α as the threshold of malicious devices our scheme can endure. α depends on the system parameters, which we discuss and extensively evaluate in Section 6.2. Compromising new devices takes non-negligible time for the adversary[2]. We further assume that the adversary cannot break cryptographic primitives. Devices that participate in denial-of-service (DoS) attacks are considered malicious in our system[3].

We assume $\mathcal{A}$ can eavesdrop and manipulate messages between devices. However, $\mathcal{A}$ is limited to disturbing the communication of nodes within physical proximity, e.g., via jamming. Hence, $\mathcal{A}$ can control only a subset of all network links, preventing it to block overall communication in the network. This can be realized through various network technologies, e.g., meshed networks with robust routing [32, 57], or upcoming technologies like 5G [1] and satellite-based networks [59, 71] where malicious network-clients have very limited means to disturb the overall network communication. Finally, $\mathcal{A}$ isolating individual devices can be inherently tolerated by GrandDetAuto as faults in the consensus phase.

In addition, for a concrete instantiation we inherit the security guarantees and assumptions of the components used by GrandDetAuto. For instance, if we use remote attestation to validate the software state of a device, the respective assumptions of the remote attestation framework will apply to GrandDetAuto. This means that we may assume the existence of some trust anchor on the involved devices and consider physical attacks out of scope. Similarly, GrandDetAuto inherits the protection capabilities of the used components, e.g., different attestation schemes can detect different types of software attacks.

2.2 Requirements

A scalable and flexible malicious device detection scheme for collaborative autonomous networks shall fulfill the following properties:

- R.1 *Detection and Identification:* On the one hand, an adversary trying to maliciously interfere with the network shall be detected. On the other hand, if the adversary tries to manipulate the detection scheme itself at any point, it shall be detected as well.
- R.2 *Efficiency:* Validating the integrity of a device must be significantly more efficient than letting all nodes validate that device individually.
- R.3 *Scalability:* The computational effort and communication complexity grows sub-linear with respect to the number of devices (scaling to large networks).
- R.4 *Interchangeable Components:* Individual components have clearly separated roles and objectives, making them easily replaceable.

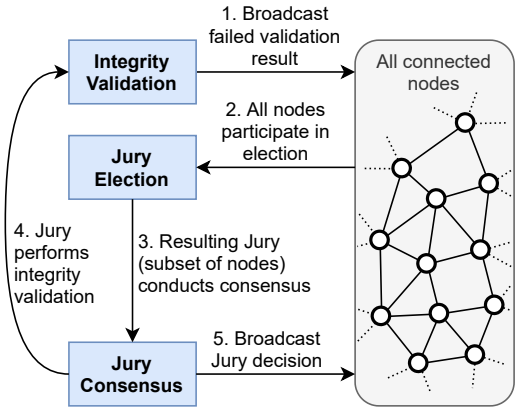

Figure 1: Interactions between the components.

3 GRANDDETAUTO DESIGN

GrandDetAuto provides a scalable solution to detect malicious devices in truly autonomous networks, i.e., without external supervision from a central entity. It consists of three main components, as seen in Figure 1, and works as follows. A node uses *Integrity*

[1] Managing membership is an orthogonal problem with existing solutions; we outline one in Section 5.2.
[2] Assuming basic security like memory layout randomization, exploiting devices requires many attempts [11, 19, 33, 44, 61, 62, 79].
[3] As a result those devices will be handled by the recovery mechanism, e.g., by expelling them.

Validation on another node and if this validation fails, the node will announce the other node as suspicious by broadcasting the validation result (1. in the figure). This starts the second phase of the protocol. In this phase, the *Jury Election* will select a group of nodes acting on behalf of the whole network, i.e., the jury (2.). The resulting jury will then use the *Jury Consensus* to reach an agreement (3.) by confirming the initial integrity validation (4.). Finally, after the jury reached a consensus, the decision will be broadcast to the rest of the network (5.). This jury decision can enforce an action, e.g., excluding a malicious node from the network. We discuss this aspect in Section 8, which is not in scope of this work.

Figure 2 illustrates an exemplary run of GrandDetAuto. After n_2 notices n_1 suspicious behavior, we call the announcement of this suspicion to the rest of the network *blaming*. Next, the network randomly elects the jury in a distributed manner, in this case n_3, n_4, n_5. Each juror will individually validate the claim made by n_2, find a consensus about the blamed n_1 as well as the decision how the network shall react, and broadcast the result among the network. This example solely illustrates one round of GrandDetAuto, i.e., one processed suspicion.

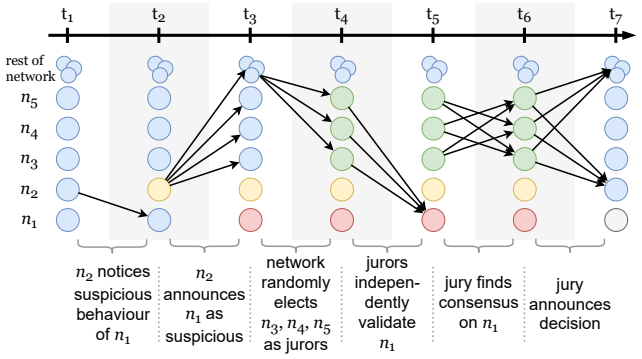

Figure 2: An exemplary setup with one adversary (red) and three jurors (green), *rest of network* refers to n_6, ..., n_i.

GrandDetAuto is triggered on-demand by individual nodes, and thus does not entail any overhead in a network of benignly acting nodes. Instead, all nodes individually look for suspicious behavior among local nodes that are potentially malicious. Depending on the scenario in which GrandDetAuto is used, suspicious behavior can be detected in various ways. For instance, inconsistencies in sensor readings exchanged between devices can be used to detect adversarial devices, as demonstrated in industrial control systems [5, 25]. Similar approaches could be used in settings like smart traffic, where different cars can match their sensor readings against reported data from other cars to detect inconsistencies. Suspiciously acting nodes are then examined thoroughly. We achieve this by giving each node the ability to *blame* another node, i.e., broadcasting to the whole network that a node acts suspiciously and may be adversarial. This approach drastically limits the adversary's impact on the overall system. As soon as the adversary starts to impact other devices, e.g., by sending false information, it will be quickly detected and sanctioned. In Section 8 we also discuss a modification for GrandDetAuto to detect passive adversaries as well.

Once a node is blamed, the network has to reach a decision whether the node is malicious. For a sustainable autonomous network, it is important to have a consistent view across the network including the order of processed blames, as concurrent ones can conflict. Thus, a form of consensus is needed. However, consensus protocols do not scale to large networks, due to their exponential message complexity [16].

To overcome this fundamental limitation, GrandDetAuto randomly elects a jury as the representatives to make a decision for the whole network. Ensuring a fair election in a distributed system is important for the security of the overall system, as the mutually distrusting nodes need to reliably agree on a common jury. Due to the distributed nature of GrandDetAuto, the challenge is to reliably converge on an election result. Otherwise, the individual views on which nodes are part of the jury may diverge and cause additional faults for the consensus or at worst, entirely prevent finding a quorum. Therefore, the right choice of key parameters guiding the election is critical. We examine the effect of such parameters in Section 7.3 and demonstrate a suitable trade-off between election reliability and the run-time of GrandDetAuto.

While executing consensus only on a subset of nodes improves scalability, it comes at the expense of the consensus' safety. As the election of the jury is random, there is a chance that a sufficient number of malicious nodes are among the elected jury so that they can enforce an adversarial decision within the jury. However, we can adjust the consensus so that it stalls rather than fails, as stalling can be rectified by a re-election. In Section 6.2 we analyze how these probabilities behave regarding GrandDetAuto's configurable parameters. We show that these parameters can be chosen such that the probability of electing an adversarial jury is negligible.

Resiliency. A challenging problem to address is how the system can defend itself against abuse. More specifically, an adversarial node may try to blame an honest node to disrupt the system, e.g., blaming the blamer. Furthermore, the adversary may try this multiple times to increase the chances of electing enough accomplices to successfully seize the jury, or simply try to use the blaming mechanism to overload the system with requests. In case a blame was unjustified, the jury will decide to blame the potentially dishonest blamer, immediately starting another round to determine if the blamer is indeed malicious. Further, a node clearly violating the expected behavior of an underlying component can result in the node being blamed as well. For example, when the validation process is deterministic, correct jurors can safely blame a juror that reaches a different conclusion from the same data. These automatic blaming approaches will prevent the adversarial nodes to turn the chances in their favor over time, as attempts to manipulate the protocol will in turn risk getting blamed themselves.

4 GRANDDETAUTO DESIGN DECISION

GrandDetAuto is designed to be modular; hence, individual components for each phase (integrity validation, random jury election and consensus) can be instantiated differently, based on the requirements of the underlying application; thus, GrandDetAuto fulfills R.4. This section will enumerate the options we identified

for each component and state the choices made for our instantiation of GrandDetAuto we present in Section 5, which also further elaborates on the chosen schemes.

Integrity Validation Scheme. GrandDetAuto requires a mechanism for detecting the initial suspicious behavior of a potentially malicious device as well as a mechanism for the jurors to validate blames. More concretely, it should be possible to verify the integrity of a node, whether its behavior or state deviates from what is expected. The integrity validation should not rely on a central trusted entity and be able to run on devices with limited computational resources. Further, the result of the validation (e.g., through a node) should be verifiable by other devices.

There is a rich body of literature on proposals to determine whether a device is behaving as expected. We identified the following options: Unsupervised outlier detection for sensor data [12, 53, 82], which is the prevalent method used in Wireless Sensor Networks; Intrusion Detection Systems (IDS) monitor for anomalies in network traffic in order to discover intrusions; or Remote attestation, which is a security primitive that enables a verifying party to receive direct proof that the software of a remote device is in a trustworthy state based on the verifier's trust policy (e.g., the code is not altered). We leverage remote attestation to instantiate the integrity validation component of GrandDetAuto, as this approach is agnostic towards the targeted use case, opposed to the careful consideration required to define outliers or anomalies, which are highly context-specific. As the node's program (execution) intrinsically defines its behavior, attestation can detect maliciously acting nodes. In Section Section A.1 we will elaborate on the aforementioned approaches as well.

Note, it is possible to use two distinct validation schemes for different phases of GrandDetAuto. As shown in Figure 2, the initial validation raising the suspicion can be done with a lightweight but overestimating scheme like outlier detection, e.g., by observing inconsistencies in communication with another party. Then the elected jury can perform a thorough and complex scheme like remote attestation to validate this initial suspicion.

Random Jury Election. After a node has been blamed, the network randomly elects a jury. For GrandDetAuto, the election scheme should work in a distributed and verifiable manner as well as ensure fairness, i.e., every node has the same chance of being elected. Approaches for distributed random elections can be found in the blockchain space. Their goal is to elect the proposer for the next block by a fair "lottery". Their security is usually based on monetary incentives to prevent Sybil attacks [23], i.e., a node assuming multiple identities to unfairly increase its influence. Unfortunately, this means they are not directly applicable for our purpose.

However, we identified the following non-incentivized schemes: Algorand [31] elects a delegation group to propose the next block by leveraging a Verifiable Randomness Function; Byzcoin [45] also uses a delegation for block proposal based on their success mining blocks via a Proof-of-Work scheme; or Intel's Proof-of-Elapsed-Time (PoET) [36], which forces nodes to wait for a random amount of time and the "fastest" node may propose a block. Most relevant for our instantiation of GrandDetAuto is PoET, as it can significantly reduce message overhead for the network (see Section 5.2). However, as it is designed to elect a single node, we extend the scheme to be able to elect a group of nodes, i.e. the jury[4], as described in Section 5.2. Appendix A.2 will discuss the other mentioned schemes as well.

Consensus. After all jurors performed their individual integrity validation of the blamed node, they need a consensus scheme to agree on the result and the reaction to it. Keeping a consistent order of the jury decisions is crucial, as multiple simultaneous blame requests may occur that depend on each other. For example, one round may elect a juror that is expelled from the network in another round. In GrandDetAuto, a consensus scheme should ensure that blame requests are consistently processed, including their order.

While this can be achieved via the inherent properties of the election (see Appendix A.3), the use of a consensus protocol, i.e., Byzantine Fault Tolerance (BFT), eliminates the need to do an election on every blame, significantly reducing the overhead of the elections over multiple rounds. This way, we can keep an elected jury for a selectable time window. Especially if multiple nodes are blamed in quick succession, BFT can have a significantly higher throughput. We use Practical Byzantine Fault Tolerance (PBFT) [16] for our instance of GrandDetAuto. There are variations of PBFT that may also be used, e.g., to improve performance if malicious behavior is expected to be rare. We elaborate on these alternatives in Appendix A.3. However, the de-facto baseline in the BFT literature is PBFT [16] and does not introduce additional assumptions.

5 IMPLEMENTATION

Subsequently, we present our full implementation of an instance of GrandDetAuto for a smart-traffic use-case using off-the-shelf devices. Our prototype uses an ultrasonic sensor to perform distance measurements that are shared with other road users. This is a common task in smart traffic scenarios [74], in which sharing environment-sensing data is crucial for vehicles to avoid collisions. GrandDetAuto can be used in this scenario to identify adversarial vehicles that send altered environment-sensing data, and endanger other vehicles. A neighboring node raises suspicion whenever the measured distance changes abruptly (assuming the measurement can be modeled as a continuous function), leading to an initial attestation by the neighbor. This starts a round of GrandDetAuto. Figure 3 shows an overview of the communication flow.

As the underlying platform we used ARM TrustZone [9] running Open Portable Trusted Execution Environment (OP-TEE) OS [52], in line with previous works for automotive use cases using TrustZone [34, 51, 55]. TrustZone divides the system into a secure world and a normal world by loading a trusted OS separately from the normal OS, e.g., Linux. Sensitive code is executed in the secure world, in so-called *Trusted Applications (TAs)*. TAs may only communicate with the normal world with explicitly allocated shared memory regions. TAs are managed by the secure-world OS, in our case OP-TEE OS. The main protocol, handling the communication and the Byzantine agreement, as well as the distance measurement software run as normal-world applications, while the attestation and the random election are implemented as TAs. These small TAs communicate with the main protocol executing in the normal world.

[4]We execute consensus among the elected jury, and thus, unlike with Nakamoto-style consensus, we have instant finality after a round.

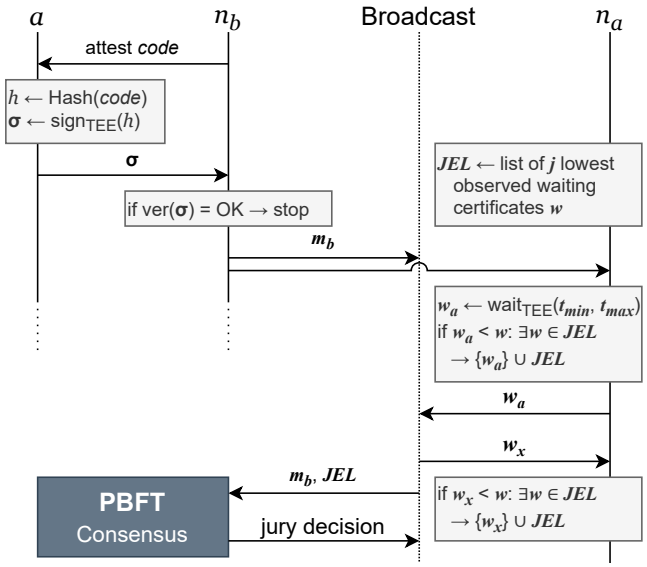

Figure 3: Overview of the protocol instance.

In the following, we describe each of the implemented components in detail.

5.1 Device Attestation

To validate the integrity of a device, remotely attesting its software can provide strong security guarantees, as it enables nodes to directly prove that their software is not altered. There are different attestation schemes, which can detect different classes of software attacks [3, 4, 21, 30, 66]. Our implementation uses binary remote attestation as the integrity validation scheme. Binary attestation is still a relatively simple mechanism, and hence is not secure against software run-time attacks using techniques such as code-reuse attacks [29] that do not require any modification to the code leaving the hash value unchanged. However, any other (e.g., more sophisticated) attestation scheme can be used as well, such as (hardware-assisted) run-time attestation schemes that can also be used to detect run-time modifications. We instantiate the integrity verification component with a more enhanced attestation scheme for evaluation in Section 7.1.

Our binary attestation scheme enables GrandDetAuto to detect any unintended modification of the code, as such modifications are recorded and included in the attestation report σ. In GrandDetAuto, if a verifier node's attestation of another node is negative, it can use the attestation report as the evidence for the jury to confirm its blame. The attestation process is implemented using a TA in TrustZone's secure world based on OP-TEE. The attestation TA computes the hash of the normal-world app, i.e., its *code* in memory, and signs it with a signing key, which is kept confidential inside the TA. The signing key needs to be issued by the device vendor and must be part of a public key infrastructure such that other nodes can verify signed attestation reports. This signed attestation report σ is then sent to the verifier, i.e., another device. If the verifier finds that the state reported in σ is not trustworthy, it blames the prover device by broadcasting the *blame message* m_b containing σ.

5.2 Random Jury Election

For the jury election, we use an approach inspired by Intel's PoET for the random jury election. PoET leverages TEEs and a registration process based on linkable attestation, i.e., attestation directly bound to a specific processor, to prevent Sybil attacks. Each participant gets a publicly verifiable random number and needs to wait for this random amount of time instead. For this feature the TEE is used as well, which attests that the respective node has indeed waited for its assigned amount of time, i.e., generating a *waiting certificate*. Afterwards, the participant will broadcast this result and each node will deem the earliest observed waiting certificate as the election's winner. This approach inherits a degree of fault-tolerance, as a crashed election winner can easily be replaced with the next best node. Further, the waiting approach reduces message complexity. Honest nodes with a comparatively high number will also wait longer and may observe lower-valued waiting certificates in the meanwhile. In this case, the node will decide not to announce its own wait certificate, saving overhead as only a minor part of the network needs to announce their respective wait certificates.

However, this approach is designed to only elect a single block proposer. Thus, we designed and implemented our own approach to elect a group of nodes, i.e., the jury. To implement a waiting approach analogously to PoET, we use OP-TEE functions providing a secure timer (`TEE_GetSystemTime`) as well as a secure wait function (`TEE_Wait`). Both are used to implement the wait TA. As the wait time is derived from publicly known data, the wait time itself is publicly verifiable. Thus, the node simply passes the wait time to the wait TA, which waits for the given amount of time. By getting the trusted system time inside the TEE before and after the wait, we simply have to sign both timestamps to get a valid waiting certificate w. This procedure is done on every node in the network.

The interplay between the different nodes during the election of j jurors works as follows:

(1) As soon as a node receives a blame message m_b, it will generate a random waiting time chosen from a exponential distribution and wait for the generated amount of time. The waiting time is in the range between a defined minimum t_{min} and maximum t_{max}.
(2) Each Node will receive waiting certificates from other nodes to compile a Jury Election Leaderboard JEL, a sorted list containing waiting certificates with the lowest observed waiting times. When a node n_a receives a waiting certificate from another node w_x, the node will first check if the certificate has merit, i.e., if the waiting time is lower than the largest entry in JEL or if $|JEL| < j$. Nodes refrain from validating and forwarding any w if they do not have merit at that time. Otherwise, the node will check the validity of w_x and add it at the corresponding position in its JEL. If $|JEL| > j$, the last entry is removed so $|JEL| = j$.
(3) After n_a is finished waiting, it will check if $|JEL| < j$ or if its own waiting time is smaller than any of the entries in JEL. If its own waiting certificate w_a has merit, it will announce it to the network and add it to its JEL. If it is not the case, it will discard w_a.
(4) After n_a additionally waited for a pre-defined time threshold t_{ele}, it will assume its JEL to be mostly complete. If the

node's own w_a is still in *JEL*, it will assume to be part of the jury. If so, it will start the agreement process with the other jurors found in *JEL*.

This way, all nodes will eventually converge towards an identical Jury Election Leaderboard *JEL* consisting of the j waiting certificates with the smallest waiting time. Thus, it is essential to choose suitable values for t_{min}, t_{max}, and t_{ele} to ensure a reliable election. We extensively evaluate these parameter in Section 7.3.

5.3 Byzantine Agreement

To find a consensus among the jurors about a blamed node, we chose to implement PBFT [16]. Using a BFT scheme eliminates the need to do an election on every blame, significantly reducing the overhead of the elections over multiple rounds. Especially if multiple nodes are blamed in quick succession, BFT can have a higher throughput.

In general, BFT is used to find a consensus among a group where some might be faulty or adversarial, i.e., act *Byzantine*. In BFT at least $3f + 1$ total nodes are required to endure f Byzantine nodes [50]. Traditional BFT schemes assume all nodes participate in the agreement process; thus, if the process fails due to too many Byzantine nodes, it is impossible to succeed. However, in our case, every election will have a diverse agreement group and may succeed where the previous jury failed. A failed BFT agreement does not prevent progress in GrandDetAuto, as the failure can trigger a new election resulting in a new jury that is likely to proceed. Furthermore, we consequently need to consider an additional negative case that we label the *total fail* case. When enough adversarial jurors are elected to reach a quorum, they can enforce a malicious decision, i.e., violating BFT's *safety* guarantee. Further, the consensus can also fail due to not reaching a quorum at all, i.e., violating the *liveness* guarantee. In Section 6.2 we examine the probabilities of both re-elections and total fails.

In PBFT, the primary decides on which request is being processed next by the consensus group. In the *prepare* phase, the participants exchange this request among each other to ensure the primary sent all participants the same request. Afterwards, the participants execute the request and exchange their results in the *commit* phase. Our scheme works as follows, using PBFT as a subprotocol:

(1) After the election, the Jury Election Leaderboard *JEL* has a sorted list of the j lowest wait times for each juror. The juror with the lowest wait time will be the primary.
(2) On conflicting blame requests and elections, the jury containing the overall shortest waiting time is selected for the next round. Thus, the initial round for a new jury can skip the prepare phase entirely.
(3) Otherwise, PBFT will be executed among the jury about the validity of σ included in the agreed on m_b.
(4) The jury decision (see Section 8) is then broadcast by all jurors, containing at least two thirds of all jurors' signatures. The rest of the network can consider each valid and consistent decision message on the same blame to be equivalent. This avoids separately spreading up to j inconsequentially different decision messages.

5.4 Communication Aspects

In our implementation, two aspects with regards to communication are particularly relevant: (i) Broadcasting was implemented by using a flooding-based protocol. Every node forwards broadcast messages to all neighbors, except the one from which the message was originally received. This way, a message will take the optimal paths, and thus flooding is optimal regarding run-time. We discuss alternatives in Section 8. (ii) To reduce communication overhead in terms of message sizes (in bytes), we use a collective signature scheme. In the consensus phase, all jurors have to individually consent by providing their own signatures. As we evaluate different jury sizes, we decided to implement the Schnorr signature scheme [73]. This way, increasingly adding signatures to a message does not result in increasing BFT messages sizes.

6 SECURITY EVALUATION

In this section, we evaluate GrandDetAuto's security and present an analysis the probabilities of the jury consensus to fail. We will show how different parameters affect GrandDetAuto and provide the foundation for selecting a suitable configuration.

6.1 Security Consideration

As mentioned in the Sections 2.1 and 2.2, the adversary's goal is to either evade being identified (detected) by GrandDetAuto, or to misuse GrandDetAuto to manipulate the overall system, e.g., by having benign devices identified as malicious by the system and sanctioned. Subsequently, we will individually explain each goal and why it cannot be achieved by the adversary $\mathcal{A}$.

Evade identification. To evade the identification of nodes controlled by $\mathcal{A}$, it can follow different strategies: (1) try to prevent being detected initially, (2) prevent being blamed, (3) prevent that a consensus is found identifying the adversary-controlled node.

Strategy 1: To avoid initial identification $\mathcal{A}$ can (a) stop interacting with the overall system and not participate in the integrity validation, or (b) behave correctly according to the used integrity validation scheme used. If $\mathcal{A}$ isolates itself while at the same time not answering to integrity validation request will ultimately lead to the conclusion that a node is not behaving correctly. However, given an appropriate integrity validation scheme $\mathcal{A}$ will not be able to pass it, unless it breaks the validation scheme, which is assumed to be secure.

Strategy 2: Once an adversary-controlled node has been recognized by another node, this node will send out a blame message to inform the network. To prevent this, the adversary (a) can compromise the blamer node, (b) suppress the communication from the blamer node, or (c) vilify the blamer node.

For (a), $\mathcal{A}$ would need to compromise the blamer *before* it is able to send out the blame message. Although we consider this case out of scope (cf. Section 2.1), even if $\mathcal{A}$ manages to compromise the blamer node, this node will be verified eventually and reported. For (b), $\mathcal{A}$ first needs to continuously control *all* communication channels of the blamer node; yet, even then $\mathcal{A}$ will be verified and reported eventually by any other benign node. Lastly, in (c), $\mathcal{A}$ might try to discredit the blamer so other nodes will not believe the blame, i.e., the compromised node will broadcast a blame message

accusing the blamer node. In this situation, both nodes will be examined by the jury, which will uncover the real adversary.

Hence, $\mathcal{A}$ only succeeds by entirely preventing the propagation of blame messages, i.e., GrandDetAuto is secure against the second attack strategy, as long as the assumptions hold that $\mathcal{A}$ does not have complete control over the network (cf. Section 2).

Strategy 3: Finally, $\mathcal{A}$ can try to prevent that the network finds agreement regarding the compromise of a node. The adversary can (a) try to sabotage the election/forming of a jury, (b) control a quorum of the jury, (c) prevent interaction between jury members, or (d) the broadcast of the jury decision. Finally, (e) $\mathcal{A}$ can distort the random jury election process to cause inconsistencies within the network that will affect the decision-making in the subsequent consensus phase.

To sabotage the jury election and forming, $\mathcal{A}$ needs to block overall communication in the network, which is assumed to not be possible (cf. Section 2.1). The adversary could also subvert the nodes to be part of the jury, e.g., to shut them off. However, with high probability, a quorum of nodes will be elected that are not compromised by $\mathcal{A}$, as long as the total adversary share does not exceed α as we show in Section 6.2.

In order to control a quorum of jury members, $\mathcal{A}$ can either compromise the jury members on-demand once they are elected. This, however, requires $\mathcal{A}$ to be able to rapidly compromise a quorum of jurors, which contradicts our adversary model (cf. Section 2.1). Otherwise, $\mathcal{A}$ has to break the random jury election scheme to reliably get nodes that are under its control to be elected as jurors. Since the jury is randomly selected, there is a chance that the adversary-controlled nodes get elected. As we show in Section 6.2 this probability is negligible with the right choice of parameters for an adversary share up to α.

To prevent the benign jurors from finding consensus, $\mathcal{A}$ can disrupt their communication. First, $\mathcal{A}$ needs to prevent a quorum to actually disrupt the consensus. Second, when preventing a quorum, the jury will trigger a re-election that results in a new jury. To continuously disrupt each newly elected jury, $\mathcal{A}$ is required to disrupt any communication in the network; hence, contradicting our adversary model (cf. Section 2.1).

Furthermore, $\mathcal{A}$ could try to prevent the jury from announcing the agreed-on result to the network, which implies that $\mathcal{A}$ needs to prevent all broadcasts by every juror.

Finally, $\mathcal{A}$ could try to manipulate the jury election process in order to prevent devices in the network to learn the correct list of jury members. As a consequence, these devices would not accept the decision of the jury leading to inconsistencies between different nodes of the network. However, this would require $\mathcal{A}$ to *permanently* prevent the wait certificates by legitimate jury members from arriving at selected devices. Given that the random jury election scheme does provide the guarantee that the elected jury is known to the entire network, all devices will eventually accept the decision made by a quorum of legitimate jurors as soon as they learn the list of legitimate jurors. Even if some devices do not learn the decision of the jury, i.e., have differences in *JEL* due to waiting certificates being withheld by $\mathcal{A}$, this will have the effect of reducing the fault-tolerance of the subsequent Byzantine agreement, with 'shortest j' nodes missing from a node's *JEL* being replaced by other nodes from outside this set, essentially manifesting as an additional fault. Thus, the security of GrandDetAuto depends on the security provided by the used schemes.

Hence, in order for $\mathcal{A}$ to succeed with strategy 3 it has to break one of the used schemes (integrity validation, random jury election, or consensus finding), has to prevent broadcast by all jurors, or be able to quickly compromise all jury members. Each of these attacker capabilities violate our system and adversary model.

Manipulate system. The adversary can also try to manipulate the system by misusing GrandDetAuto. In particular, by blaming benign nodes $\mathcal{A}$ can try to get them sanctioned, e.g., excluded from the network to increase its own share of the network. However, to achieve this $\mathcal{A}$ has to alter the integrity validation report of a benign node to convince the jury that the node is compromised. This means $\mathcal{A}$ has to break the authentication method used by breaking a cryptographic primitive like signatures, which is not possible (cf. Section 2.1). Alternatively, $\mathcal{A}$ can aim to gain control over a decision-making majority of the jury to come to a malicious agreement that will be accepted by the entire network. Here the same arguments hold as discussed above for strategy 3b and the probability of success is analyzed in the following (Section 6.2).

In summary, the adversary can only misuse GrandDetAuto when breaking one of the underlying schemes or with negligible probability, and thus fulfills the requirement R.1.

6.2 Probabilistic Analysis

The adversarial share of nodes α GrandDetAuto can tolerate depends on the probabilities of electing compromised devices as jurors. As the election is random, it may happen that enough adversarial nodes are elected for the Byzantine agreement to fail, as described in Section 5.3. This section discusses the probabilities for different adversary shares α as well as the chosen jury size j. The joint decision is based on a Byzantine agreement, which means it fails if more than $\lfloor (j-1)/3 \rfloor$ jurors are adversarial [16]. While a larger j reduces the chances of a failed election, Byzantine Fault Tolerance (BFT) also induces a message complexity of $O(j^2)$. In Section 7.5 we will evaluate this effect in our simulation.

If we have n total nodes in our system, with f of them being adversaries and elect j jurors, the probability of electing at least $\lfloor (j-1)/3 \rfloor$ adversarial nodes is:

$$1 - \frac{\binom{j}{k+1}\binom{n-j}{f-k-1}}{\binom{n}{f}} \, {}_3F_2\left[\begin{matrix} 1,\ k+1-f,\ k+1-j \\ k+2,\ n+k+2-f-j \end{matrix} ; 1 \right] \tag{1}$$

Where $k = \lfloor (j-1)/3 \rfloor$ and ${}_pF_q$ is the generalized hypergeometric function. Equation 1 is the cumulative distribution function of the hypergeometric distribution.

While this equation models the probability for the Byzantine agreement to fail, we can rectify a liveness violation by re-election. In some applications, it may make sense to accept reduced fault-tolerance by increasing the quorum size required by PBFT from $\lfloor 2(j-1)/3 \rfloor + 1$ to some greater value q. Then, $n - q \leq \lfloor (j-1)/3 \rfloor$ faults are sufficient to cause a liveness violation, but a safety violation requires a greater number $2q - n$ of faults. If the protocol reaches an impasse, another consensus round, including a new jury, is started that may succeed. This can be modelled as a Markov chain: we begin in an initial "undecided" state and transition to a

"success" state if no more than $n - q$ adversarial nodes are elected—guaranteeing agreement—and a "failure" state if at least $2q - n$ adversarial nodes are elected—allowing a safety violation. The failure state will eventually be reached with probability

$$\mathrm{P}[\text{Eventual Failure}] = \frac{\mathrm{P}[F \geq 2q - n]}{\mathrm{P}[F \geq 2q - j] + \mathrm{P}[F \leq j - q]} \tag{2}$$

and it will take on average $1/\mathrm{P}[j - q < F < 2q - j]$ elections to leave the "undecided" state.

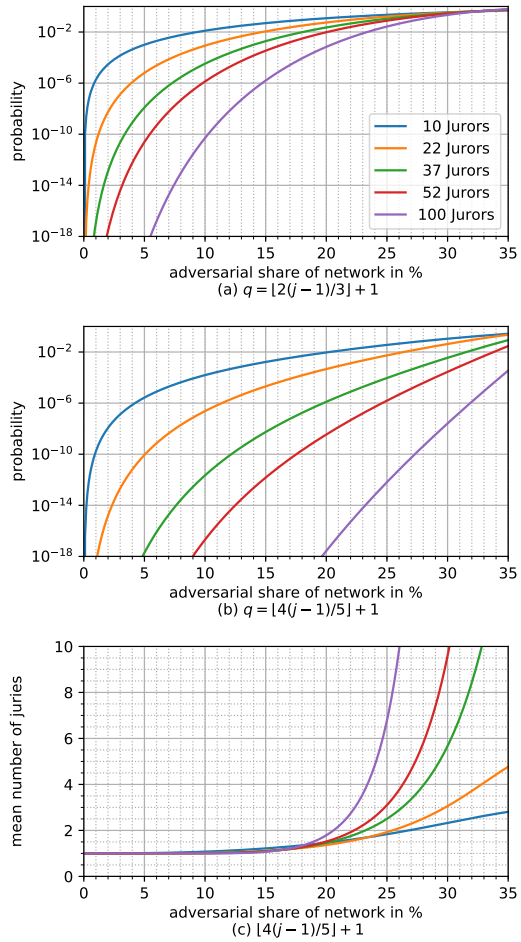

Figure 4: The probability of eventual safety violation of Byzantine agreement with a population size of 10 000 given a threshold of (a) $q = \lfloor 2(j-1)/3 \rfloor + 1$ and (b) $q = \lfloor 4(j-1)/5 \rfloor + 1$, as well as the mean number of juries needed before agreement terminates, whether in success or *total* failure, for $q = \lfloor 4(j-1)/5 \rfloor + 1$ in Figure (c). The distinctly colored graphs depict the probability development for different jury sizes j. Note that the case depicted in (a) will always either terminate or suffer a safety violation with a single jury election, unlike that in (b) and (c) where several juries may be necessary.

Besides the threshold q, a primary factor affecting the probability of an eventual safety violation is the jury size j. The more jurors are elected per round, the lower the probability for the Byzantine agreement to fail. We illustrate the influence of the jury size j and BFT threshold q in Figure 4. The choice of jury size j and threshold q is therefore application-dependent, depending upon the appropriate trade-off between failure probability, time to reach agreement, and performance. We evaluate the latter of these considerations in Section 7.5.

Table 1: The measured run-times of individual processing steps.

wait certificate generation	static attestation generation	static attestation validation	BFT process + Schnorr-sign
42 ms	166 ms	1 ms	14 ms
DIAT GPS attestation generation		DIAT GPS attestation validation	
835 ms		849 ms	

7 PERFORMANCE EVALUATION

In this section, we measure the performance of our GrandDetAuto instance using a small-scale network running on real hardware, and use these results as the basis for simulating large-scale networks. Regarding the results of these simulation campaigns, we first analyze the effects of differently chosen wait time parameters. These parameters need to be chosen carefully to ensure the random election is consistent. Afterwards, we examine the scalability of our GrandDetAuto instance for large networks regarding run-time and messaging overhead, showing sub-linear run-time growth in regards to network size. Finally, we show how this performance is affected by choosing different jury sizes.

7.1 Prototype Measurements

For reference measurements to use in the large-scale simulation (see Section 7.2) we deployed our GrandDetAuto implementation (see Section 5) on a setup of ten nodes. We used Raspberry Pi 3 Model B+ [65] as the platform and connected a distance sensor for attestation. This platform is comparable to ARM's line-up of chips specifically designed for vehicles [8] in terms of computational power and capabilities. The Raspberries are running a Raspbian Linux after OP-TEE is loaded. We connected all nodes to a router via WiFi for communication between them.

Enhanced attestation. To demonstrate that GrandDetAuto can scale with a more complex integrity validation scheme as well, we also consider Data Integrity Attestation (DIAT) [4], which targets trustworthy data exchange for collaborative autonomous networks, such as cars or drones. DIAT is a remote attestation approach that can detect even sophisticated software attacks, such as run-time attacks [29]. Due to the increased complexity, it also needs more processing for both the attestation generation and validation.

The top half of Table 1 shows our measurements regarding run-time of our implementation. The BFT and Schnorr signing is fluctuating depending on jury size, so for the simulation, described in Section 7.2, we chose to use the worst-case (14 ms). The other numbers are averages over 100 runs. The bottom half of Table 1 shows the DIAT run-time numbers for attesting a GPS module, as reported in the paper [4].

7.2 Simulation

To evaluate the performance of GrandDetAuto for large numbers of devices, we used the OMNeT++ network simulator [60]. We implemented GrandDetAuto at the application layer and used the measurements described in Section 7.1 to set the processing times for the individual steps taken by each node.

Our network is configured in a square mesh topology, with roughly the same height and width. Every node has four links to its neighbors, except the nodes at the edge of the network. To make the simulation representative, we used dynamic communication delays between nodes. We measured the latencies in our distributed setup described in Section 7.1 in different scenarios, such as highly varying distances commonly encountered in vehicle-to-vehicle communication. We measured 3 ms at best and 78 ms at worst for one-way delays. For the simulation, each communication link gets a random delay assigned between these two measurements. Routing for dynamic networks is an orthogonal problem [27, 77] and does not contribute to a meaningful evaluation of GrandDetAuto. Thus, we use a simple on-demand routing algorithm.

We evaluate GrandDetAuto for different network sizes, from 1 000 to 100 000 nodes. We also split measurements into the different phases. We simulate the first round of our GrandDetAuto instance with the following phases[5]:

(1) Initial Attestation: Generation of the initial attestation report σ by the blamed node and the integrity validation of σ by the blamer.
(2) Blame: Broadcasting the initial blame message.
(3) Election: The election process to elect j jurors.
(4) BFT: The consensus protocol, including the validation of σ by each juror.
(5) Decision: Broadcasting the outcome of the BFT.

Notice that this represents the worst case, i.e., the upper bound regarding run-time and message overhead, as it includes both the election and the complete BFT. Further, to minimize variation of individual simulation runs, due to the random nature of our scheme, we average every individual parameter configuration over 100 runs with different random numbers.

7.3 Election Wait Time

We evaluate the time parameters t_{max} and t_{ele}, as they contribute significantly to the performance characteristics of GrandDetAuto. t_{max} is the maximum wait time regarding the randomly chosen wait time for each node. After a node is done waiting, it will wait an additional time t_{ele} while collecting other waiting numbers. Afterwards, it will assume the election to be mostly complete, i.e., to have a mostly matching *JEL*. If t_{ele} is chosen very small, the election itself will be faster; however, the individual *JEL*s may also be still inconsistent among the nodes. To measure this effect we executed a parameter study for differently chosen time parameters with $n = 2\,000$, $j = 22$ and $t_{min} = 100ms$.

Figure 5 shows the results. In (a) we can see the effect on the execution time of one round. It is primarily tied to t_{ele}, as can be seen if comparing different t_{max} that result in the same t_{ele}. The graph (b) in turn shows how many unjustified BFT messages were

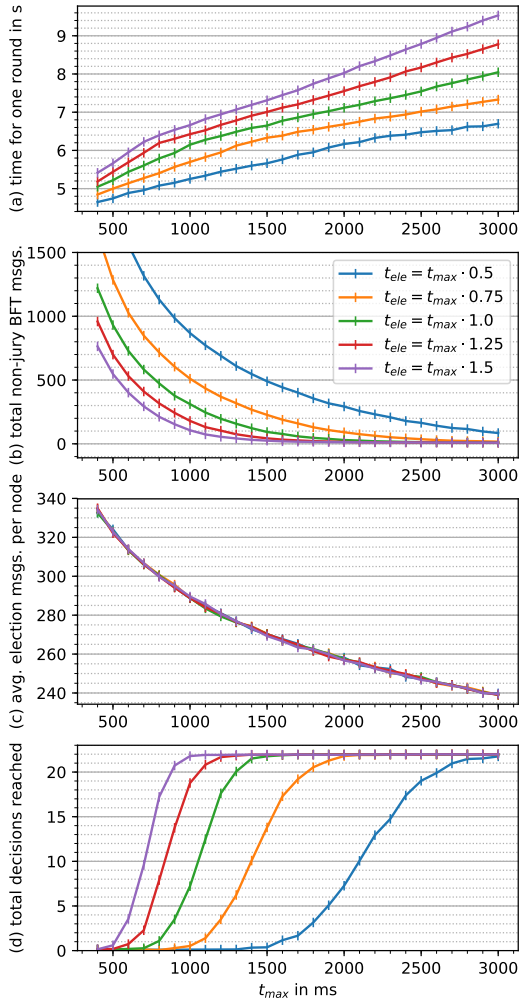

Figure 5: (a) The time for one round, (b) the total number of non-juror BFT messages, (c) the average number of election messages per node and (d) the total number of reached decisions, all for differently chosen t_{max} and t_{ele}. Simulated with $n = 2\,000, j = 22, t_{min} = 100ms$.

received. While a lower t_{ele} reduces the execution time, it also increases the number of nodes falsely assuming to be jurors.

Figure 5 (c) shows the average number of messages per node for the election phase. This shows how many nodes actively participate in the election, i.e., nodes assuming their wait time still has merit after waiting for t_{ele}. However, (d) shows how many jurors get to the point of sending out a decision. This measurement should ideally match the chosen j, so 22 in this case. Even though lower numbers would suffice, matching j allows the network to better account for other faults in the consensus phase. Yet, it can be seen that if the time parameters are chosen too small, not the entire jury can reach a decision or none at all, as the nodes' *JEL* will diverge to the point where no quorum can be established among the jury.

With these measurements in mind, we chose a time parameter configuration for our further evaluation, keeping the discovered

[5]Subsequent rounds will be faster, as no election is needed.

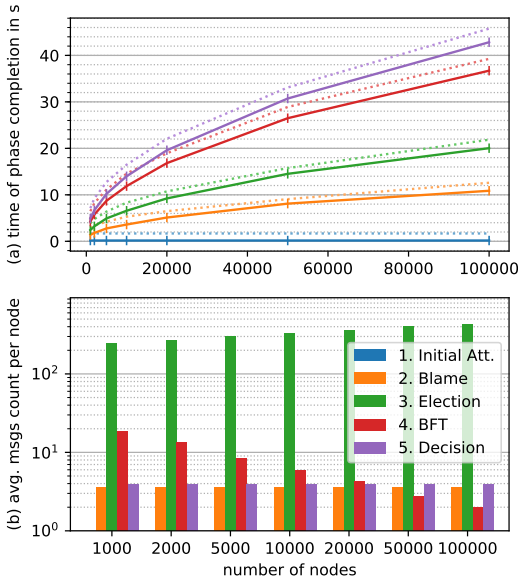

Figure 6: (a) The completion time of each phase, results for static attestation and DIAT (dotted lines), and (b) the average number of messages sent per node, all split into the individual phases for increasing n. Simulated with $j = 22$.

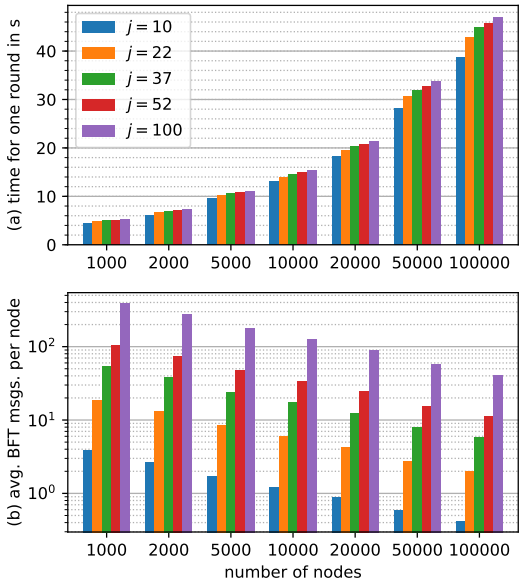

Figure 7: (a) The time for one round and (b) the average number of BFT messages sent per node for different j.

trade-offs in mind: $t_{max} = 1\,000ms$ and $t_{ele} = 1\,500ms$ for $n = 2\,000$. After many experiments for different network sizes, we found a dynamic configuration, which works for all network sizes. Based on the average delay of 37.5 ms between nodes, we set $t_{ele} = \sqrt{n} \cdot 37.5ms \cdot 0.9$ and $t_{max} = t_{ele} \cdot \frac{2}{3}$. The square root of n times the delay reflects half of the worst-case communication delay in our topology and 0.9 means it can be 10% lower than that, while still resulting in a reliable election.

7.4 Per-Phase Performance for Large Networks

In this section, we examine the *Efficiency* and *Scalability* of GrandDetAuto, two main requirements (cf. Section 2.2). Figure 6 (a) shows the run-time measurements. Note that the measurements per phase are denoted as the *absolute* simulation time at the last processed message of the respective phase—phases overlap as progress is made in parallel. The top purple line represents the time of the last received decision message in the network, and thus the total time for one entire GrandDetAuto round. We consider two attestation schemes with differing complexity (see Section 7.1). One regarding our static attestation and the other regarding the more complex DIAT [4] (shown as dotted lines). A network of $n = 100\,000$ takes $42.83s$ with our static attestation approach and $45.76s$ with DIAT, fulfilling requirement R.3. Note that the worst-case communication delay between any two nodes in this scenario is $23.71s$ on average.

A naive and simplified solution to the problem would be to let all devices attest every other device individually. The time this case takes for n nodes is the combined time of the generation and verification of an attestation multiplied by $(n - 1)$. This does not consider communication delay and assumes perfect parallelization between the nodes. This naive case would take almost 3 minutes for $n = 1\,000$ (~14 minutes with DIAT) and over 4 hours for $n = 100\,000$ (~23 hours with DIAT); hence, the naive approach is ~390x slower (~1800x slower with DIAT) than GrandDetAuto, and thus we deem GrandDetAuto efficient (R.2).

Figure 6 (a) also shows the individual measurements per phase. The third line in green shows how long the election takes. The time for the election overlaps with the blame broadcast, implying that the election requires the most time of the scheme. The red line, second from the top, shows when the BFT is finished. Figure 6 (b) show the overhead per phase in terms of message count. Note that we consider all messages for these measurements, i.e., including forwarded messages by nodes in between the route. The graph reveals that the election phase (green bars second from the left respectively) generates the most overhead. The total message overhead for $n = 100\,000$ is 333.01 messages per node. However, assuming a subsequent round with a jury already in place, the total message overhead without the election phase is reduced to 9.63 messages per node.

7.5 Jury Size

The following examines the effects of differently chosen jury sizes j on our instance of GrandDetAuto. Figure 7 (a) shows the run-time for one round. For large juries, like $j = 100$, in a large network, like $n = 100\,000$, the difference on the run-time compared to $j = 10$ is $8.3s$ (or 21.4%). This is due to the individual BFT steps being able to execute in parallel.

The second graph (b) show the average message count per node of the BFT phase. The $O(n^2)$ message complexity for two BFT phases are apparent. Nevertheless, the closest case we could find to compare the election overhead against the BFT overhead is $n = 1\,000$ and $j = 100$. Here the average message overhead per node for the election is 590.65 against the 387.11 for the BFT message overhead. Thus, a BFT round is more efficient than an election in overall terms. However, BFT also concentrates the overhead on the jurors and the

routes between them, compared to the more uniformly distributed overhead by the election.

8 DISCUSSION

This section discusses possible extensions to GrandDetAuto.

Broadcast. To reduce communication costs in GrandDetAuto, a gossip protocol can be used. Gossip protocols randomly send broadcast messages to a set number of neighbors, which in turn do the same [49]. These protocols are probabilistic in nature, yet, perform well on average and significantly reduce overhead compared to flooding [49].

Monitoring. An aspect that could be changed is the on-demand nature of the initial integrity validation. For example, one could have all nodes regularly check all their neighbors instead[6]. These validations can be entirely local per node and the GrandDetAuto process would only be triggered when a node actually notices inconsistencies. This way, every node would be validated eventually, so even a passive adversary cannot hide.

Dynamic Jury. The jury size does not have to be fixed over the life span of a GrandDetAuto instance. It might be advantageous to dynamically adjust the jury size when required, e.g., increase the jury size when many blames occur in a short time frame. This would dynamically adjust the security probabilities along the network's needs at the time.

Jury Decision. Part of a practical instantiation of GrandDetAuto is the resulting reaction of the jury to a confirmed adversarial node. This by itself is a vastly complex topic and highly dependent on the use case. Straightforward expulsion of the malicious node as a result of the jury decision is not possible in many use cases, e.g., cyber-physical systems like autonomous cars, where expulsion from the network does not prevent them from affecting the system. In these situations, some other response might be more appropriate. Mechanisms exist for *self-healing* [24], in which a faulty node is returned to a valid state. In this case, we might choose to use GrandDetAuto not to exclude an adversarial node, but to decide whether a node will be *added* to the network.

9 RELATED WORK

Distributed Outlier Detection. In the field of Wireless Sensor Networks (WSNs), outlier detection is used to identify unusual sensor readings to detect faults, exceptional events, or malicious attacks. There are different techniques to identify outliers, from simple statistical methods (e.g., Gaussian-based models), up to classification-based methods, such as machine learning [82]. The outlier evaluation can be done locally per node or by a central node trusted to handle the decision making [82]. To distribute this task among the nodes, one proposal is for each node to only consider the neighbors for outliers and keep exchanging decisions among the network until a global view is achieved [12]. An efficiency improvement to this approach is to do a majority vote among a neighborhood, which can be exchanged with other neighborhoods [53].

However, these distributed approaches require nodes to exchange a significant amount of data and messages to converge to a global view. This limits their applicability to large-scale networks, unlike GrandDetAuto which easily supports 100 000 nodes.

Distributed Intrusion Detection. Similar to outlier detection, Distributed Intrusion Detection Systems find anomalies specifically for network traffic in WSNs [40, 67] and mobile ad-hoc networks [35, 43, 72, 81]. These approaches also use similar techniques to identify anomalies; yet, there are also approaches based on prior knowledge of normal operations or a defined specification [13]. There are different ways to distribute data aggregation and anomaly processing to reduce communication overhead. However, in the context of this work, the key aspect is how decisions are made. One approach is for nodes to collaborate with their neighbors for the measurements and decide via majority vote on a suspected node [40, 81]. This assumes compromised devices are in the minority in every neighborhood, as colluding adversaries may easily form local majorities otherwise. Another method is to separate the network into clusters and have their members elect the *cluster heads*, which representatively make decisions [35, 43]. However, either the election is disregarding that it might be malicious [43], or the assumption is a low threat environment implying a low probability of electing a malicious node [35].

In contrast, GrandDetAuto can tolerate many malicious devices in any distribution. Other works in this field introduce some form of centralization, such as a trusted base station making decisions [20, 67], or a group of nodes on top of a hierarchy of clusters [72].

Collective Attestation. The first step towards scalable attestation of large groups of interconnected devices, i.e., collective attestation, was made by SEDA [10]. SEDA, like all other schemes that followed in the collective attestation literature [6, 7, 38, 46] assume a central verifier; hence, they are not applicable in the autonomous scenarios targeted by GrandDetAuto. Further, these approaches aim to verify the whole system at once on request, whereas GrandDetAuto ensures security in a sustainable way by giving all nodes a tool to continuously and autonomously validate each other.

10 CONCLUSION

In this work, we presented GrandDetAuto, the first scheme to efficiently identify adversaries in large networks of autonomous collaborating devices. GrandDetAuto combines random elections, consensus and integrity validation methods in a flexible scheme, where each of these components are interchangeable. We have demonstrated the scalability of an exemplary instance as well as provided the basis to construct use-case specific instances of GrandDetAuto.

ACKNOWLEDGMENTS

We thank N. Asokan (University of Waterloo) for his useful feedback. This work has been supported by the German Research Foundation (DFG) as part of the project S2 within the CRC 1119 CROSSING, by the ASSURED project funded by the EU's Horizon 2020 programme under Grant Agreement number 952697, by German Federal Ministry of Education and Research (BMBF) within the project iBlockchain, by the Academy of Finland (grant 309195), by the European Space Operations Centre with the Networking/Partnering Initiative, and by the Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS).

[6] As long as the use case allows this without violating our assumptions.

REFERENCES

[1] 3GPP. 2020. 5G Standard – Release 16. https://www.3gpp.org/release-16.

[2] 5G Automotive Association. 2016. C-V2X White Paper: The Case for Cellular V2X for Safety and Cooperative Driving. https://5gaa.org/news/white-paper-placeholder-news-for-testing/.

[3] Tigist Abera, N Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. C-FLAT: control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 743–754.

[4] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. 2019. DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous Systems. In *Annual Network and Distributed System Security Symposium*.

[5] Sridhar Adepu, Ferdinand Brasser, Luis Garcia, Michael Rodler, Lucas Davi, Ahmad-Reza Sadeghi, and Saman Zonouz. 2020. Control Behavior Integrity for Distributed Cyber-Physical Systems. In *11th IEEE/ACM Conference on Cyber-Physical Systems (ICCPS)*.

[6] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and Scalable Aggregate Network Attestation. In *ACM SIGSAC Conference on Computer and Communications Security*.

[7] Mahmoud Ammar, Mahdi Washha, and Bruno Crispo. 2018. WISE: Lightweight intelligent swarm attestation scheme for IoT (the verifier's perspective). In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 1–8.

[8] ARM. 2021. Solutions - Automotive. https://www.arm.com/solutions/automotive.

[9] ARM Limited. 2008. ARM Security Technology Building a Secure System using TrustZone Technology. https://developer.arm.com/documentation/genc009492/c.

[10] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. https://doi.org/10.1145/2810103.2813670

[11] Kjell Braden, Stephen Crane, Lucas Davi, Michael Franz, Per Larsen, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2016. Leakage-Resilient Layout Randomization for Mobile Devices. In *Annual Network and Distributed System Security Symposium*.

[12] Joel W Branch, Chris Giannella, Boleslaw Szymanski, Ran Wolff, and Hillol Kargupta. 2006. In-network outlier detection in wireless sensor networks. *Knowledge and Information Systems* 34, 1 (2006), 23–54.

[13] Ismail Butun, Salvatore D Morgera, and Ravi Sankar. 2013. A survey of intrusion detection systems in wireless sensor networks. *IEEE communications surveys & tutorials* 16, 1 (2013), 266–282.

[14] Eric Byres and Justin Lowe. 2004. *The Myths and Facts behind Cyber Security Risks for Industrial Control Systems*. Technical Report. PA Consulting Group.

[15] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweight Swarm Attestation: A Tale of Two LISA-s. In *ACM Symposium on Information, Computer and Communications Security*.

[16] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*.

[17] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. 2011. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *USENIX Security Symposium*.

[18] Eric Chien, Liam OMurchu, and Nicolas Falliere. 2011. *W32.Duqu - The precursor to the next Stuxnet*. Technical Report. Symantic Security Response.

[19] Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen, Ahmad-Reza Sadeghi, Stefan Brunthaler, and Michael Franz. 2015. Readactor: Practical Code Randomization Resilient to Memory Disclosure. In *IEEE Symposium on Security and Privacy*.

[20] Ana Paula R da Silva, Marcelo HT Martins, Bruno PS Rocha, Antonio AF Loureiro, Linnyer B Ruiz, and Hao Chi Wong. 2005. Decentralized intrusion detection in wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*. 16–23.

[21] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. 2018. LiteHAX: Lightweight Hardware-Assisted Attestation of Program Execution. In *2018 International Conference On Computer Aided Design (ICCAD'18)* (San Diego, California, US).

[22] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. 2016. Resource-efficient Byzantine fault tolerance. *IEEE Trans. Comput.* 65, 9 (2016), 2807–2819.

[23] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[24] Walaa Elsayed, Mohamed Elhoseny, Alaa Mohamed Riad, and Aboul Ella Hassanien. 2017. Autonomic self-healing approach to eliminate hardware faults in wireless sensor networks. In *International conference on advanced intelligent systems and informatics*. Springer, 151–160.

[25] Sriharsha Etigowni, Mehmet Cintuglu, Maryam Kazerooni, Shamina Hossain, Pengfei Sun, Katherine Davis, Osama Mohammed, and Saman Zonouz. 2017. Cyber-Air-Gapped Detection of Controller Attacks through Physical Interdependencies. In *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*.

[26] F-Secure Labs. 2016. BLACKENERGY and QUEDAGH: The convergence of crimeware and APT attacks.

[27] Kevin Fall. 2003. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 27–34.

[28] Nicolas Falliere, Liam O Murchu, and Eric Chien. 2011. W32.stuxnet dossier. *White paper, Symantec Corp., Security Response* 5, 6 (2011), 29.

[29] Aurélien Francillon and Claude Castelluccia. 2008. Code Injection Attacks on Harvard-architecture Devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '08)*. ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/1455770.1455775

[30] Aurélien Francillon, Quan Nguyen, Kasper B. Rasmussen, and Gene Tsudik. 2014. A Minimalist Approach to Remote Attestation. In *Design, Automation & Test in Europe*.

[31] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17*. ACM Press, 51–68. https://doi.org/10.1145/3132747.3132757

[32] Tal Grinshpoun, Amnon Meisels, and Eyal Felstaine. 2014. Avoidance of misbehaving nodes in wireless mesh networks. *Security and Communication Networks* 7, 7 (2014).

[33] Jason D. Hiser, Anh Nguyen-Tuong, Michele Co, Matthew Hall, and Jack W. Davidson. 2012. ILR: Where'd My Gadgets Go?. In *IEEE Symposium on Security and Privacy*.

[34] Shengtuo Hu, Qi Alfred Chen, Jiwon Joung, Can Carlak, Yiheng Feng, Z Morley Mao, and Henry X Liu. 2020. CVShield: Guarding Sensor Data in Connected Vehicle with Trusted Execution Environment. In *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*. 1–4.

[35] Yi-an Huang and Wenke Lee. 2003. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*. 135–147.

[36] Hyperledger. 2020. Hyperledger Sawtooth Documentation on Proof-of-Elapsed-Time. https://sawtooth.hyperledger.org/docs/core/nightly/0-8/introduction.html.

[37] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Gene Tsudik. 2016. DARPA: Device Attestation Resilient against Physical Attacks. In *ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*.

[38] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. 2017. SeED: secure non-interactive attestation for embedded devices. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 64–74.

[39] IEEE. 2010. IEEE 802.11p-2010: Extensions to IEEE Std 802.11 for wireless local area networks (WLANs) providing wireless communications while in a vehicular environment. https://standards.ieee.org/standard/802_11p-2010.html.

[40] Krontiris Ioannis, Tassos Dimitriou, and Felix C Freiling. 2007. Towards intrusion detection in wireless sensor networks. In *Proc. of the 13th European Wireless Conference*. Citeseer, 1–10.

[41] A. Jaeger and S. A. Huss. 2011. The weather hazard warning in simTD: A design for road weather related warnings in a large scale Car-to-X field operational test. In *11th International Conference on ITS Telecommunications*.

[42] Michel E. Kabay. 2010. Attacks on Power Systems: Hackers, Malware. https://www.networkworld.com/article/2217684/attacks-on-power-systems--hackers--malware.html.

[43] Oleg Kachirski and Ratan Guha. 2003. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. IEEE, 8–pp.

[44] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning. 2006. Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software. In *Annual Computer Security Applications Conference*.

[45] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*. 279–296.

[46] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. 2018. SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS)*. https://doi.org/10.1145/3196494.3196544

[47] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. 2010. Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy*.

[48] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of*

Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07). ACM, 45–58. https://doi.org/10.1145/1294261.1294267
[49] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. 2002. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless networks* 8, 2/3 (2002), 169–185.
[50] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.
[51] Seungho Lee, Wonsuk Choi, Hyo Jin Jo, and Dong Hoon Lee. 2019. T-Box: A forensics-enabled trusted automotive data recording method. *IEEE Access* 7 (2019), 49738–49755.
[52] Linaro, Inc. 2021. OP-TEE Documentation. https://optee.readthedocs.io/en/latest/.
[53] Fang Liu, Xiuzhen Cheng, and Dechang Chen. 2007. Insider attacker detection in wireless sensor networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 1937–1945.
[54] Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. 2018. Scalable Byzantine Consensus via Hardware-assisted Secret Sharing. *IEEE Trans. Comput.* (2018). https://doi.org/10.1109/TC.2018.2860009
[55] Renju Liu and Mani Srivastava. 2017. PROTC: PROTeCting drone's peripherals through ARM trustzone. In *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. 1–6.
[56] Prince Mahajan, Ramakrishna Kotla, Catherine C. Marshall, Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, and Ted Wobber. 2009. Effective and Efficient Compromise Recovery for Weakly Consistent Replication. In *Proceedings of the 4th ACM European Conference on Computer Systems*.
[57] Navamani Thandava Meganathan and Yogesh Palanichamy. 2014. Privacy Preserved and Secured Reliable Routing Protocol for Wireless Mesh Networks. *The Scientific World Journal* (2014).
[58] Charlie Miller and Christopher Valasek. 2014. A Survey of Remote Automotive Attack Surfaces. In *Blackhat USA*.
[59] OneWeb. 2021. OneWeb Home Page. https://www.oneweb.world/.
[60] OpenSim Ltd. 2021. OMNeT++ Discrete Event Simulator. http://omnetpp.org/.
[61] Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. 2012. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-Place Code Randomization. In *IEEE Symposium on Security and Privacy*.
[62] PaX Team. 2001. PaX address space layout randomization (ASLR). http://pax.grsecurity.net/docs/aslr.txt.
[63] Jonathan Pollet and Joe Cummins. 2010. Electricity for Free? The Dirty Underbelly of SCADA and Smart Meters. In *Blackhat USA*.
[64] Jerome Radcliffe. 2011. Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System. In *Blackhat USA*.
[65] Raspberry Pi Foundation. 2021. Raspberry Pi 3 Model B+. https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.
[66] Arvind Seshadri, Mark Luk, and Adrian Perrig. 2008. SAKE: Software Attestation for Key Establishment in Sensor Networks. In *Distributed Computing in Sensor Systems*.
[67] Sooyeon Shin, Taekyoung Kwon, Gil-Yong Jo, Youngman Park, and Haekyu Rhy. 2010. An experimental study of hierarchical intrusion detection for wireless industrial sensor networks. *IEEE Transactions on Industrial Informatics* 6, 4 (2010), 744–757.
[68] F. Shrouf, J. Ordieres, and G. Miragliotta. 2014. Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm. In *IEEE International Conference on Industrial Engineering and Engineering Management*.
[69] Meital Ben Sinai, Nimrod Partush, Shir Yadid, and Eran Yahav. 2015. Exploiting Social Navigation. In *Blackhat Asia*.
[70] Slash Gear. 2019. The 2020 VW Golf 8 could make smart cars mass-market. https://www.slashgear.com/2020-volkswagen-golf-8-car2x-v2v-smart-cars-mass-market-25597350/.
[71] SpaceX. 2021. Starlink Mission. https://www.starlink.com/.
[72] Daniel Sterne, Poornima Balasubramanyam, David Carman, Brett Wilson, Rajesh Talpade, Calvin Ko, Ravindra Balupari, C-Y Tseng, and T Bowen. 2005. A general cooperative intrusion detection architecture for MANETs. In *Third IEEE International Workshop on Information Assurance (IWIA'05)*. IEEE, 57–70.
[73] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*. Ieee, 526–545.
[74] The Conversation. 2018. Connected cars can lie, posing a new threat to smart cities. http://theconversation.com/connected-cars-can-lie-posing-a-new-threat-to-smart-cities-95339.
[75] The Guardian. 2011. DigiNotar SSL certificate hack amounts to cyberwar, says expert. https://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar.
[76] The Guardian. 2016. DDoS attack that disrupted internet was largest of its kind in history, experts say. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.
[77] Chai K Toh. 2001. *Ad hoc mobile wireless networks: protocols and systems*. Pearson Education.
[78] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo. 2013. Efficient Byzantine Fault-Tolerance. 62 (Jan 2013), 16–30. https://doi.org/10.1109/TC.2011.221
[79] Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin. 2012. Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code. In *ACM SIGSAC Conference on Computer and Communications Security*.
[80] ZDNet. 2016. How hackers attacked Ukraine's power grid: Implications for Industrial IoT security. https://www.zdnet.com/article/how-hackers-attacked-ukraines-power-grid-implications-for-industrial-iot-security/.
[81] Yongguang Zhang, Wenke Lee, and Yi-An Huang. 2003. Intrusion detection techniques for mobile wireless networks. *Wireless Networks* 9, 5 (2003), 545–556.
[82] Yang Zhang, Nirvana Meratnia, and Paul JM Havinga. 2010. Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Surveys and Tutorials* 12, 2 (2010), 159–170.

A APPENDIX

This section will give a thorough discussion on the identified options mentioned in Section 4.

A.1 Integrity Validation Options

Outlier Detection. Unsupervised outlier detection for sensor data [12, 53, 82], which is the prevalent in Wireless Sensor Networks, enables the validation of measurements reported by individual nodes. Hence, it allows to identify malicious nodes sending manipulated data. We outline the different types of anomalies and detection techniques in Section 9. The detected outliers as well as the required accompanying data (e.g., outlier reports of the blaming node's neighbors [12]) can be used as the disputed evidence in GrandDetAuto.

Intrusion Detection. Intrusion Detection Systems (IDS) monitor for anomalies in network traffic in order to discover intrusions. In GrandDetAuto, detected anomalies can serve as evidence. Instead of sending an *attack report* [20] or aggregated observations [67] to a central authority, this data can be used as evidence to trigger an investigation by a jury. Section 9 outlines several distributed approaches for IDS proposed in the literature.

A.2 Random Election Options

Algorand. In Algorand [31] a delegation group is randomly elected to propose new blocks. Here, each node draws a number based on the Verifiable Randomness Function (VRF). The lowest numbers win the election, and thus the delegation group is elected. The VRF works as a deterministic source of randomness and as such is publicly verifiable. Put simply, each node's individual random number is the hash of the concatenation of its identity, i.e., public key, and the last block's hash. This results in a random number that is verifiable by all participants, as only public information is necessary to calculate it. Algorand also needs to protect against Sybil attacks, as nodes may freely join the network. Each node has stake, i.e., the amount of money they own in the system, which is used to assign weight to their random number. Thus, the bigger a node's monetary stake the higher the chances to be elected and vice versa. However, if all participants are known, the election itself can be executed without requiring stake. It also implies a large message overhead, as all participants broadcast their election numbers virtually simultaneously. Further, Algrorand builds on a adapted

binary consensus algorithm adapted to transaction block selection, and thus is specifically designed for cryptocurrencies.

Byzcoin. For selecting the consensus group Byzcoin [45] requires to mine consensus blocks via Proof-of-Work (PoW). Then, a chosen number of the last successful miners emerges as the group executing Byzantine Fault Tolerance (BFT). This elected group will then propose a new transaction block together. The key issue when using PoW is that in a heterogeneous network some less powerful nodes have a significant disadvantage in the election. Further, an adversary may even use a powerful external machine to exceed the processing power of the entire network.

Deterministic Random Jury. Another simplified approach is to use a single verifiable source of randomness instead of many. Thus, the drawn number elects the whole jury. This way, instead of having all nodes announce their number individually, the whole network would deterministically know who is part of the next jury. For example, a counter of the GrandDetAuto round could be concatenated with all of the identities of the previous jury and subsequently hashed as the source of randomness. This approach has a very low overhead for the election phase; yet, one has to consider the possibility that an elected juror crashed, which leads to additional faults in the consensus phase.

A.3 Consensus Options

Simple Majority. If we assume to elect a new jury every round and every juror has a random number, we can extract an inherent order of requests. On two conflicting requests, there will be two separate elections with two separate juries. In such a case, the juries decide which request is executed first by comparing their election results. With the order being ensured, a simple majority vote among the jury suffices. While this requires little overhead for the consensus phase, it requires a new election each round.

BFT with Enhancements. In Section 6.2 we show that with increasing jury size, the probability to fail decreases significantly. However, a larger jury also implies a larger overhead due to the $O(n^2)$ message overhead of BFT [16]. To counter this, different enhancements of Practical Byzantine Fault Tolerance (PBFT) can be employed to reduce complexity. For example, the speculative case [48], skipping PBFT phases, or the optimistic case [22], halving the consensus group. However, both reduce overhead only for the benign case. Thus, if we expect Byzantine events to be rare, it may be feasible to consider larger jury sizes with inherently better security guarantees. Another approach is to involve a message aggregation scheme [45] to significantly reduce message complexity. If we have a trusted component available, we can also employ a trusted monotonic counter, which removes the need for BFT's prepare phase entirely as well as reducing the required quorum to half plus one nodes [54, 78].