# Malware Tolerant (Mesh-) Networks

Denzel, Michael; Ryan, Mark

*DOI:*
10.1007/978-3-030-00434-7_7



*Document Version*
Peer reviewed version

*Citation for published version (Harvard):*
Denzel, M & Ryan, M 2018, Malware Tolerant (Mesh-) Networks. in *Proceedings of the 17th International Conference on Cryptology And Network Security (CANS 2018)* . Lecture Notes in Computer Science, vol. 11124, Springer, pp. 133-153, 17th International Conference on Cryptology And Network Security (CANS 2018) , Naples, Italy, 30/09/18. https://doi.org/10.1007/978-3-030-00434-7_7

# Malware Tolerant (Mesh-)Networks

Michael Denzel and Mark Ryan

University of Birmingham, Birmingham B15 2TT, UK,
[m.denzel,m.d.ryan] -AT- cs.bham.ac.uk

**Abstract.** Mesh networks, like e.g. smart-homes, are networks where every node has routing capabilities. These networks are usually flat, which means that one compromised device can potentially overtake the whole infrastructure, especially considering clone attacks.
To counter attacks, we propose a network architecture which enhances flat networks, especially mesh networks, with isolation and automatic containment of malicious devices. Our approach consists of unprivileged devices, clustered into groups, and privileged "bridge" devices which can cooperatively apply filter rules like a distributed firewall. Since there is no ultimate authority (not even bridges) to control the whole network, our approach has no single point-of-failure – so-called intrusion or malware tolerance. That means, attacks on a single device will not compromise the whole infrastructure and are tolerated. Previous research on mesh networks [10, 3, 8, 9] relied on a single point-of-failure and is, thus, not intrusion or malware tolerant.
Our architecture is dynamic in the sense that bridge devices can change, misbehaving devices can be isolated by outvoting them, and cryptographic keys evolve. This effectively turns the entire network into a moving target.
We used the protocol verifier ProVerif to prove the security properties of our network architecture.

**Keywords:** mesh network, malware tolerance, self-management, network security

## 1 Introduction

Recently, smart-homes and Internet of Things (IoT) devices gain popularity through e.g. Google's Nest[1], Samsung SmartThings[2], Philips Hue[3], and Amazon Echo[4]. These smart-homes increasingly employ mesh routing, a technique where every device can forward messages. Google WiFi[5] routers already form mesh networks for performance reasons and to support non-uniformly shaped network areas (like a long stretched, narrow flat).

[1] https://nest.com/uk/about/
[2] http://www.samsung.com/uk/smartthings/
[3] http://www2.meethue.com/
[4] https://www.amazon.co.uk/Amazon-SK705DI-Echo-Black/dp/B01GAGVIE4
[5] https://madeby.google.com/wifi/

Mesh networks were originally used in isolated settings, e.g. by rescue teams in remote areas, and thus the main focus was on reliability and safety. Nowadays, these networks are more and more applied in consumer scenarios like smart-home networks, Wireless Ad-Hoc Networks (WANETs), Vehicular Ad-Hoc Networks (VANETs) and Wireless Sensor Networks (WSNs). However, with features and time-to-market influencing the strategy of the IoT industry, devices often lack security features. Various forms of attacks like black hole attacks, Denial of Service (DoS), routing table overflows, and impersonation [14] exist in addition to well-known attacks like buffer overflows and similar. As mesh networks are usually flat, compromised devices immediately supply an adversary with access to the entire network instead of part of it. This is in particular of concern when taking node capture attacks into account, where the adversary hijacks a device in wide area networks (e.g. a network of weather stations).

While some of the previous research already proposed self-healing techniques for mesh networks [8, 3], they rely on a single point-of-failure – usually a trusted base station – which could be attacked by an adversary to still successfully compromise the architecture.

Our aim is to ensure that most of the network still operates securely when parts of it are entirely controlled by an adversary and that the attack is contained and limited from spreading further. There shall be no ultimate authority with access to everything – an approach called intrusion tolerance [19] or malware tolerance [7]. Note that our method takes place *after* the initial infection where the adversary already has control over the first system.

In our research we focus on mesh networks since they are used in smart-homes.

*Contributions:*

1. We design the architecture of a malware-tolerant mesh network that isolates devices into groups. Groups are able to communicate with each other via a special bridge group that can enforce security properties and filtering like a firewall. The network also applies automatic containment based on voting to identify and quarantine threats, and is in this regard self-managing.
   It is malware-tolerant, that means none of the devices, bridge groups, components, software, or accounts alone is sufficient to take over the whole network, i.e. there is no single point-of-failure.
2. To prove our network architecture we utilise state-of-the-art protocol verifier ProVerif[6]. The proofs can be found online[7].

## 2 Overview

Let us consider a smart-home setting in the near future with smart light bulbs, smart fridge, smart door locks, entertainment zone, tablets, phones, laptops, etc. (see e.g. Fig. 1a). The WiFi of the gateway (the router) does not reach all devices

[6] `http://proverif.inria.fr`
[7] `https://github.com/mdenzel/malware-tolerant_mesh_network_proofs`

(e.g. due to the long stretched layout of the house and interference) and a flat mesh network similar to Google WiFi is used: All devices route packets. While the laptops and PCs in the network employ a firewall and an anti-virus, e.g. the light bulbs and fridge lack these defences and are easier to compromise. Thus, an attacker might get access to a light bulb and start infecting the network.

To protect the other devices from attacks (e.g. the data on the laptop), we suggest to (1) isolate different device types. We propose to setup e.g. a group for the light bulbs, one for the entertainment devices, one for the work devices, etc. This already limits the adversary to the group of the compromised device.

If an old light bulb is compromised and starts attacking the newer ones, we (2) automatically contain the attack when we detect this behaviour. All devices are allowed to apply detection and will then vote against dishonest devices. A configurable threshold of votes (we used 2) is enough to trigger a quarantine. The method of detection is hereby interchangeable, every device could employ its own detection technique or even rely on the other devices.

Lastly, there should also be (3) no single point-of-failure (malware tolerance) because this would only shift the target. E.g. if we used the PC to setup the entire network, the adversary would only have to compromise the PC to succeed. This must not be the case for our architecture.

Let us illustrate the detection in an example: Suppose the adversary compromised a light bulb in the hallway and tries to scan the ports of the PC in the living room. The traffic is e.g. routed via the dishwasher in the kitchen and two light bulbs to the PC (see red path in Fig. 1a). In addition, all the light bulbs in the hallway see the traffic because it is wireless traffic. Let us say the light bulbs have old signatures and do not detect the attack. PC and dishwasher detect the attack and vote to quarantine the compromised light bulb. These two votes satisfy our threshold and a bridge device (e.g. the TV) would isolate the compromised light bulb and trigger a renewal of the light bulb key. Note that no device would give evidence for its vote. This could be optionally possible but the result is triggered by the amount of votes not by convincing others. It relies on the assumption that most of the nodes are honest.

### 2.1 Assumptions

Since the Dolev-Yao attacker alone is insufficient for certain types of attacks [17], especially in mesh networks with e.g. node capture attacks, we extended the Dolev-Yao attacker with further capabilities.

We make the following assumptions for our architecture:

1. Attacker Model: We are looking at the state where the adversary compromised the first device (e.g. through a buffer overflow). The adversary has access to this one device on the network, including all its cryptographic keys, but we are initially not aware which device this is. Additionally, the adversary has Dolev-Yao capabilities through e.g. an arp spoof or similar.
   However, the attacker does not (yet) have any other keys and can only access devices outside of his group through plaintext messages, fake encryption, etc. This behaviour is likely noticed by the honest devices around him and

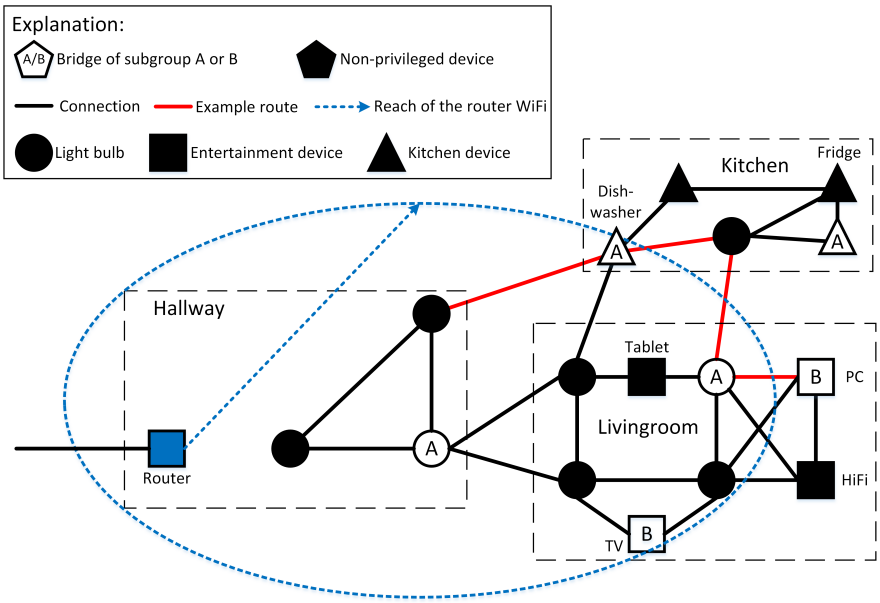
detected as suspicious activities (an IDS is not even necessary to detect this). The attacker can try to compromise further devices but a careful attacker is limited to his own group.
2. Every device is able to execute cryptography. We expect applicable CPUs to become so small in the near future that they fit into e.g. a smart light-bulb.
3. We assume the network has a high connectivity and it is not partitioned. Low connectivity enables the adversary to compromise a device that is situated at interconnections enabling him to block messages completely. As light-bulbs are normally in every room of a flat or house, this assumption is justifiable in a smart-home setting with smart light-bulbs (and further smart devices).
4. Attacks on the cryptographic algorithms itself are out of scope. I.e. we assume that e.g. AES is still secure. If it becomes insecure, the algorithm can be swapped out with the new standard.

### 2.2 Proposed Architecture

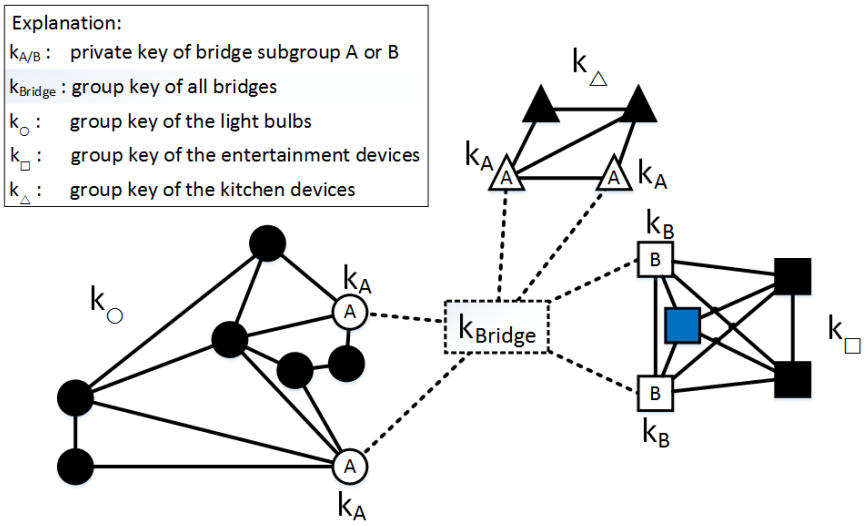
Our network architecture consists of two virtual types of devices: non-privileged devices and privileged "bridge" devices. Non-privileged devices are clustered into groups with every group having one symmetric group key – like a standalone WiFi network. All messages are encrypted (as it is also common nowadays in WiFi networks) and devices can only communicate with devices in their group. To enable groups to talk to each other, we promote a small amount of devices of each group ($\geq 2$) to bridge devices.

These bridges have two tasks:
- First, they enable communication between the device groups. A bridge device of one group and a bridge device of another group can re-encrypt and forward messages between those two groups. Filtering, similar to a firewall, works in a distributed fashion, namely at bridges.
- Second, bridges certify keys of non-privileged devices. For this, bridges are divided into two subgroups: bridge group A and B, each of which has an asymmetric key. A key of a non-privileged device is valid, if it was certified by both bridge groups A and B (i.e. by at least two different bridge devices).

Through this, we created a virtual overlay over the geographical layout of the network. An example is shown in Fig. 1 where the network is divided into three groups: light bulbs (circles), kitchen devices (triangles), and entertainment devices (squares). Non-privileged group devices are black shapes while bridge devices are white with their subgroup (A or B) written inside the shape. Connections are shown as black lines and as blue dotted circle for the router. Not every device is connected to any other one because walls and electronic currents limit connectivity. E.g. the router cannot directly reach the HiFi stereo system. Fig. 1 (b) displays the virtual layout of this network with the particular cryptographic keys.




(a) Geographic network layout showing actual data links




(b) Virtual network layout showing the links in each of the groups and the cryptographic keys. Additionally to the displayed keys, each device has its own asymmetric key pair (identity key).

Fig. 1: Example of the same mesh network with its geographic and virtual network layout

In order to react to attacks, the proposed architecture should be dynamic. Any device can *vote to promote* another device in its group to a bridge device and can *vote to exclude* any device in *any* group from the network. We do not restrict how to detect malicious behaviour; honest devices could e.g. notice unencrypted messages, protocol errors, ongoing DoS attacks or a malicious device which is dropping packages. An Intrusion Detection System (IDS), anti-virus, or honeypot could identify malware delivered from a certain source. In these cases the honest device would vote to exclude that particular dishonest device. We would like to emphasise that not every device needs an IDS. Low capability systems could rely on others to detect attacks and forego detection or only detect simple misbehaviour like unencrypted messages or protocol errors.

If there is a certain number of votes against a device (minimum two to avoid attacks), the distrusted device is isolated by placing it into a separate new group and revoking its permission to join any other group. If the isolated device was a bridge, it is also removed from the bridges (the keys evolve) and a new bridge is elected. To forbid quarantined devices to promote themselves, at least two group votes are needed to promote a device and devices are only allowed to vote to promote in their group (note that vote to exclude is possible against every device). Thus, we create a dynamic network architecture that applies a moving target defence.

A potential adversary who compromised one device in the network is, therefore, limited to the group of the device. He or she would have to compromise multiple bridge devices at the same time, also those of other groups which the attacker cannot access (yet). Additionally, bridge devices only expose a reduced interface.

Our network architecture focuses on access control. Similar to a WiFi network where the router controls which devices can join the network and who can communicate with whom. The cryptography we apply in our approach is mainly to create and supply the architecture, not to secure the requests themselves as this is already implemented by application protocols such as HTTPS. However, our architecture provides every device with an asymmetric key pair which could be used to setup a symmetric key via a Diffie-Hellman Key Exchange if the application protocol does not already supply this.

We adjusted the definition of malware tolerance to incorporate these ideas and make it applicable to networks:

> *Malware tolerance (for networks) distributes trust over several independent components in a way that an individual component infected with malware cannot gain access (spreading), deny access, or control access (MITM) of devices on the network it did not control before. No part of the network is assumed invulnerable to attacks.*

In the literature, security is usually defined in a binary fashion: a system can be secure or compromised. Malware tolerance, in contrast, has flexible trust assumptions: a compromised system can still work securely.

To achieve our self-managing, malware-tolerant network architecture, six operations are necessary. At the beginning, the network has to be bootstrapped:

1. SETUP

For the non-privileged groups there are three operations:

2. GET TICKET / JOIN GROUP
3. SEND
4. VOTE TO EXCLUDE / LEAVE GROUP

The bridges have two operations:

5. VOTE TO PROMOTE / BRIDGE JOIN
6. VOTE TO EXCLUDE / BRIDGE LEAVE

We will now introduce the underlying cryptography and these six operations.

**Basic Cryptography:** The architecture utilises four different types of keys as already mentioned in Fig. 1 (b):

1. Each device has its own asymmetric key pair – the identity key (e.g. $[k^{pub}_{fridge}, k^{priv}_{fridge}]$).
2. Each device is part of one group and has this particular symmetric group key (e.g. the light bulb key $k_\circ$).
3. Bridge devices are additionally part of the bridge group with the symmetric bridge key $k_{AB}$.
4. All bridge devices of one group have exactly one of two private keys, $k_A$ or $k_B$. E.g. both bridge devices of the kitchen group (Fig. 1 (b)) have key $k_A$, but not key $k_B$. The combination of $k_A$ and $k_B$ is used to certify the identity key of each device. For this, we use the Intrusion Resilient Signature (IRS) scheme of Itkis et al. [16, 15] A second similar cryptosystem, Intrusion Resilient Encryption (IRE), was proposed by Dodis et al. [11, 12]

The basis of IRS (and IRE) is a static public key and an evolving private key. In IRS, messages can be verified with the public key and the time period in which the signature was created. The private key material consists of the evolving private key and an evolving base secret which is kept separate at a different device. To evolve the private key, a contribution of the base secret is necessary. There are two ways to modify the private key which Itkis et al. called *update* and *refresh*. An update changes to the next time period while a refresh only changes the private secrets and leaves public key and time period unaffected. If an attacker compromised the secret key, he can read messages of the corresponding time period but does not get further keys. The attacker would have to compromise both the base and the secret key, at the same time in order to succeed. We refer the interested reader to the original papers [16, 15, 11, 12] and a summary by Franklin [13] for more information.

Our architecture uses two bridge IRS keys ($k_A$ and $k_B$) who both run an IRS scheme with the base secret being managed by the corresponding other bridge group. That means bridge key management is securely interleaved between the two bridge groups due to the properties of IRS. Certificates need to be signed with both keys to be considered trustworthy.

IRS is limited to $N = 2^t$ time periods with $t$ being the bit length of the time variable. That means for a 32-bit time variable, we would have $N = 2^{32}$ or

roughly 4 billion time periods. Assuming the network is used 100 years this corresponds to roughly 1 time period every second. Updating the secrets will likely take longer than this, limiting even DoS attacks. At the point this might run out, a re-setup of the architecture is recommended. If a DoS manages to exhaust all these time periods regardless of the aforementioned restriction, human intervention is necessary.

In the following, $E(k, m)$ refers to an authenticated symmetric encryption of message $m$ with key $k$ e.g. AES-GCM, $E(k^{pub}, m)$ denotes an asymmetric encryption, $S(k^{priv}, m)$ is a digital signature, $H(k, m)$ is a secure keyed hash function, and $IRS(k^{priv}, m)$ is the Intrusion Resilient Signature scheme. $m_0|m_1$ represents a concatenation of two messages. Also, by *time* we mean time-periods of IRS.

**SETUP:** Initially, four devices $A_1, A_2, B_1, B_2$ are selected to become bridges, each generating an asymmetric identity key-pair. Either the user sets these devices up with all of the other public keys (a simple option to establish these channels would be QR codes or a USB stick), or the devices send the keys to each other with Diffie-Hellman Key Exchange and the user verifies the fingerprints of the keys.

Afterwards, $A_1$ and $B_1$ generate $k_A^{priv}$ stored at $A_1$ and $base_A$ stored at $B_1$ of IRS (see Distributed Key Generation of Itkis et al. [16]). Similarly, $A_2$ and $B_2$ generate $k_B^{priv}$ at $B_2$ and $base_B$ at $A_2$. $A_1$ and $A_2$ share their secrets with each other, i.e. $k_A^{priv}$ and $base_B$, and similarly do $B_1$ and $B_2$. This establishes the initially needed keys. Key management is interleaved between groups A and B; A has the base secret of B and vice versa.

Two bridge IRS keys can support any number $n \geq 2$ of non-privileged groups. For additional security, three bridge IRS groups (A, B, and C) could be similarly setup. However, this comes at the cost of performance and convenience hence we suggest two bridge groups.

**GET TICKET / JOIN:** First, we have to define how devices join the network. A base of four bridge devices was already setup, thus we can rely on them.

If the user wants to connect a new device $d$ (e.g. a smart light bulb) to group $g$, he or she logs in at two devices $d_1, d_2$ (e.g. phone and PC) both different from $d$. Then, the user would request a new one-time password via $d_1$ and $d_2$ to sign up the new device $d$ to group $g$. Half of the password is displayed on $d_1$ and the other half on $d_2$.

The total password consists of four shares ($pw_{A1}$, $pw_{A2}$, $pw_{B1}$, $pw_{B2}$). Two of the shares identify to one bridge group: e.g. half-password $pw_A = pw_{A1}|pw_{A2}$ is needed to authenticate to group $A$ (similar for $B$). Each bridge IRS group generates the shares by hashing $k_A^{priv}$ or $k_B^{priv}$, the requesting device ($d_1$ or $d_2$), $g$, and the time. This way, any bridge device in the particular bridge IRS group can re-calculate the shares.

To avoid attacks of $d_1$ or $d_2$, the half-passwords are sent interleaved. E.g. $d_1$ would receive $pw_1 = pw_{A1}|pw_{B1}$ and $d_2$ receives $pw_2 = pw_{A2}|pw_{B2}$.

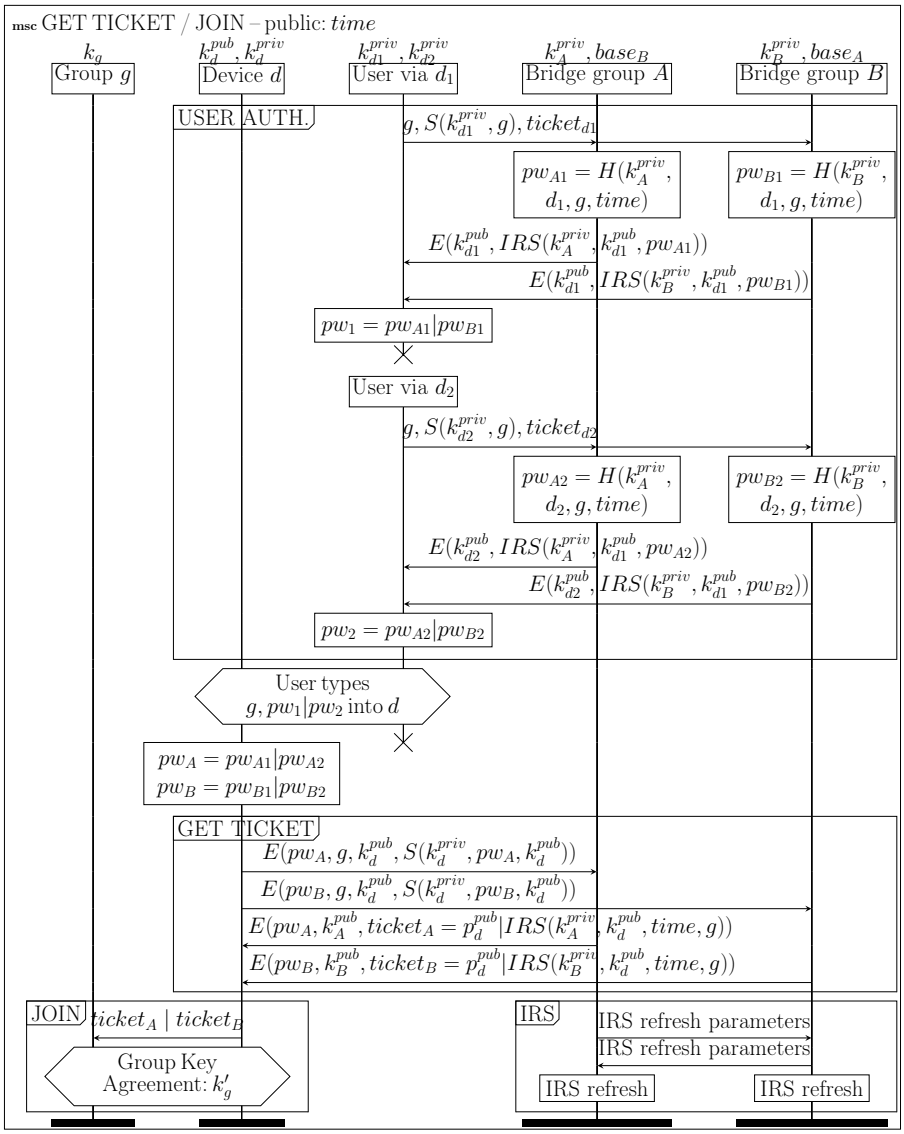



Fig. 2: GET TICKET / JOIN protocol: Device $d$ gets a ticket and joins group $g$. The user logs in with two devices $d_1$ and $d_2$.

The result is that:

- Only the user has the full password.
- No device in the entire network has the full password (not even the bridges).
- Only $A$ and $B$ have the half-password needed for authentication.
- $d_1$ has shares of the password which, alone, are worthless (similar $d_2$). Thus, neither $d_1$ nor $d_2$ can sign up.

The two devices $d_1$ and $d_2$ each show their shares to the user who then has the full password. The user concatenates all shares (displayed to him as two parts) and types the full password into $d$. With this combined one-time password the new device $d$ can identify itself to both bridge IRS groups and receives a certificate for its public key $k_d^{pub}$. We call this certificate a *ticket*. Tickets include the time-period(s) in which they are valid as well as the group to which $d$ belongs (here: $g$). This ticket must be signed by both IRS keys, i.e. by two different bridges, to be valid.

With the ticket, the device $d$ can authenticate to group $g$ and take part in a Group Key Agreement to establish a shared symmetric key with the group. Applicable are e.g. Group Diffie-Hellman (GDH), Tree-Based Group Diffie-Hellman (TGDH), Burmester-Desmedt Group Key Agreement Protocol (BD), and Skinny Tree Key Agreement Protocol (STR). Centralised Key Distribution (CKD) relies on a trusted central system and can, therefore, not be used. [1] We suggest to use BD as it is stateless but the other Group Key Agreement protocols are also possible.

The full GET TICKET / JOIN protocol is also shown in Fig. 2. We sign before encrypting because $d$ has to prove that its key $k_d^{pub}$ is connected to $pw_A$ or $pw_B$, and that $d$ is in possession of $k_d^{priv}$. Since the one-time password identifies the receiver as well, attacks are prevented through a naming repair. [6]

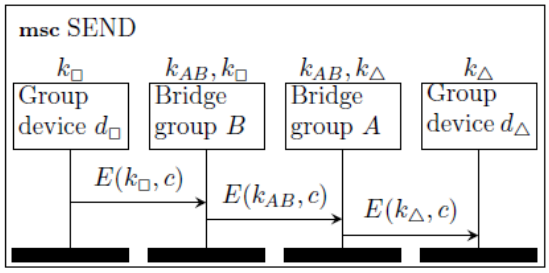



Fig. 3: SEND protocol

**SEND:** Each group device can communicate via the group key with devices in its group. However, there might be exceptional cases where two devices have to communicate over two groups. E.g. the user wants to check cooking recipes on a tablet (entertainment group) for the content of the fridge (kitchen group).

Sending messages to another group requires that bridges re-encrypt the packet from one group key ($k_\square$) to another group key ($k_\triangle$). The sending device $d_\square$ (e.g. the tablet) would send the packet to a bridge of group $\square$ which re-encrypts the message with the bridge key $k_{AB}$ and sends it to a bridge of group $\triangle$. The second bridge re-encrypts the message from $k_{AB}$ to $k_\triangle$ and sends it to the recipient (e.g. the fridge).

This re-encryption realises filtered communication between the groups. To setup a more efficient connection after the initial SEND protocol, an application protocol (e.g. HTTPS) could provide a session key. Alternatively, such a tunnel could be setup via the certified identity keys of the two devices, i.e. running an authenticated Diffie-Hellman Key Exchange via SEND.

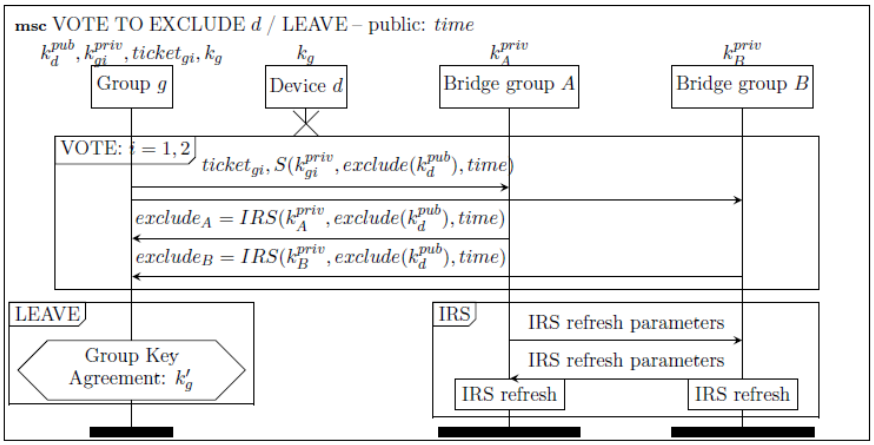



Fig. 4: VOTE TO EXCLUDE / LEAVE protocol

**VOTE TO EXCLUDE / LEAVE:** If devices are misbehaving, bridges must be able to isolate these devices. We force misbehaving devices out of a group by re-establishing the group key – so-called leave event in Group Key Agreement protocols.

Misbehaving devices can be identified by any other device which then casts a vote to quarantine the misbehaving device(s) to all bridges. If a certain threshold of votes is received (Fig. 4 indicates this by $i$ votes), the misbehaving device is forced to leave the group and the user is informed about the attack. Devices without group are quarantined and cannot communicate with any other device. This way, malware cannot spread further and is automatically contained. Optionally, one could allow quarantined devices to communicate with bridge devices in order to access the internet.

**VOTE TO PROMOTE / BRIDGE JOIN:** When a group is introduced or updated, new bridge devices are necessary. The group members directly elect these devices which are picked depending on their resources, geographic location, and/or trustworthiness. We propose that every device sends a signed message with a device for promotion but this can be replaced by any other voting algorithm that tolerates malicious nodes. This *promote* message is signed with

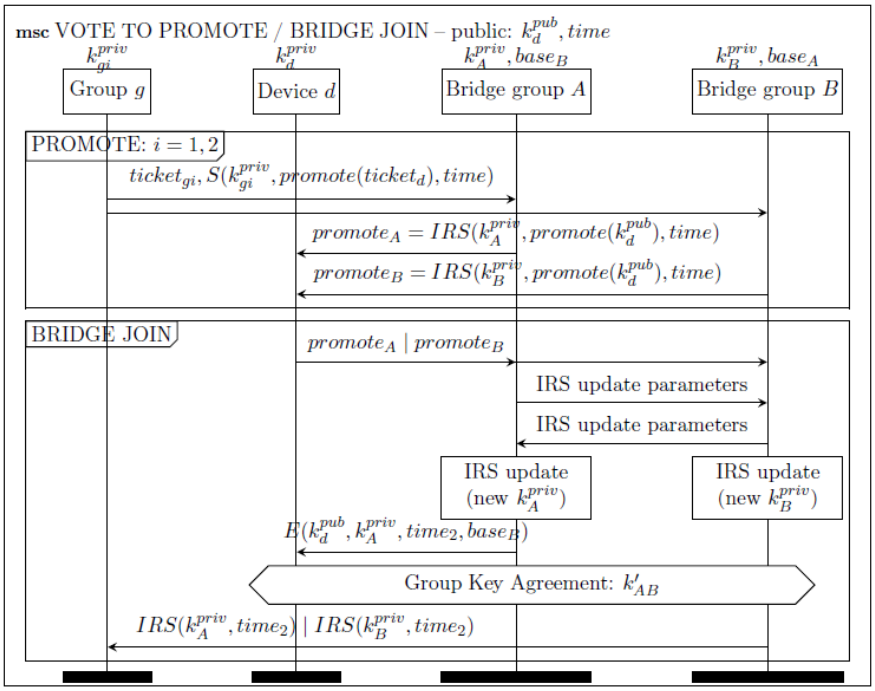



Fig. 5: VOTE TO PROMOTE / BRIDGE JOIN protocol

private key of the voter and sent to both bridge IRS groups. Bridge devices can verify the signature and record the vote. Devices which do not have the capabilities (e.g. battery), can opt-out by adding this to their vote. Self-election is possible but at least two votes are necessary to elect a bridge. At the end of the election, both bridge groups issue a promotion ticket which is sent to the promoted device to verify that it agrees to being elected. Promotion tickets must expire in the next time period to avoid devices joining twice. The bridge group which the new device should join ($A$ or $B$, not both), must be fixed and reproducible. We suggest to use a hash like $H(k_d^{pub}, time, k_{AB})$.

Promoted devices can join the bridge group similar to the JOIN operation. The new device $d$ sends the promotion ticket to the bridge group and participates in a Group Key Agreement to get the bridge key $k_{AB}$. The difference to JOIN is that bridge devices also share the asymmetric key and base secret of IRS in order to certify other keys in the GET TICKET protocol. This key material needs to be updated and shared with the new bridge device; i.e. to join bridge group $A$, $d$ gets $k_A^{priv}$ and $base_B$ (see Fig. 5). Optionally, this message could be signed with the key of the device sharing the secrets $k_{A1}^{priv}$ and include $k_d^{pub}$ as naming repair (see also [6]). This detects attacks of bridge device $A_1$.

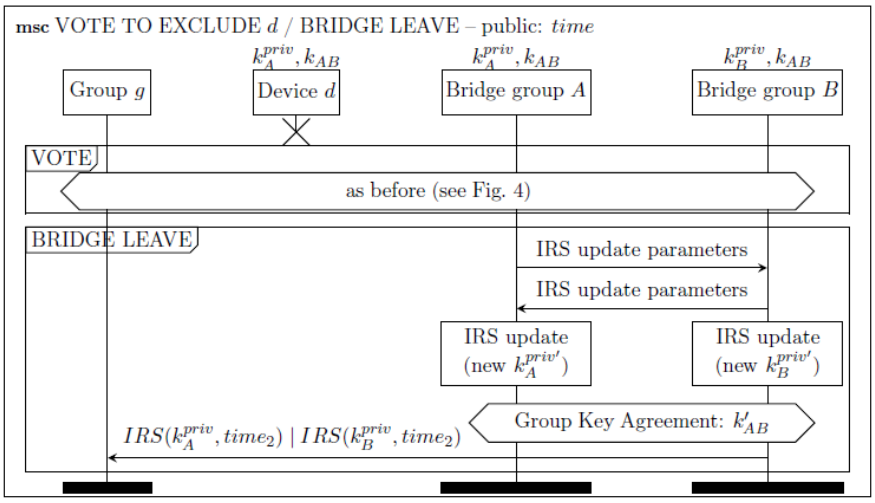



Fig. 6: VOTE TO EXCLUDE / BRIDGE LEAVE protocol

**VOTE TO EXCLUDE / BRIDGE LEAVE:** Analogous to VOTE TO EXCLUDE / LEAVE, devices can also vote against a bridge device. While the vote phase works exactly as before (see Fig. 4), BRIDGE LEAVE requires that bridge secrets (for bridge group $A$: $k_A^{priv}$ and $base_B$) and bridge group key $k_{AB}$ are also updated. Additionally, a new bridge needs to be promoted. Fig. 6 shows the details of BRIDGE LEAVE. Depending on the used Group Key Agreement, it might be more efficient to run a Group Key Agreement leave protocol first and use the renewed bridge group key to run IRS update.

## 3 Security Analysis

We evaluated our architecture using state-of-the-art protocol verifier ProVerif. The proofs are available online[8]. Each proof one by one assumes every combination of devices compromised and analyses four properties:

1. if the main purpose of the protocol is guaranteed,
2. if a second group cannot interfere with the protocol,
3. if at least one bridge IRS key is secure,
4. if the proof script ran through.

Property 3 and 4 are the same for all proofs. We will introduce them first and explain the other properties for each protocol below.

Property 3: Since we have two IRS bridge groups, it is sufficient when one of them stays secure. With bridge devices changing from time to time, the compromised bridge group can recover from the attack. This is impossible if both

[8] `https://github.com/mdenzel/malware-tolerant_mesh_network_proofs`

IRS groups are compromised at the same time. We tested for each protocol, if the confidentiality of the secret bridge keys is guaranteed.

Property 4: ProVerif does not indicate whether a proof script ran entirely through or aborted early. To test that the proof finished, we leaked an artificial secret at the end of the protocol. If the attacker is in possession of the secret, we know that the proof script reached the end of the protocol.

Table 1: ProVerif Results: GET TICKET / JOIN

| No | Compromised Devices | Authentication | Group 2 join | $k_A/k_B$ secure | End reached |
|---|---|---|---|---|---|
| 1 | None | ✓ | ✓ | ✓ | ✓ |
| 2 | $d$ | ✓ | ✓ | ✓ | ✓ |
| 3 | $d_1$ | ✓ | ✓ | ✓ | ✓ |
| 4 | $d_2$ | ✓ | ✓ | ✓ | ✓ |
| 5 | $A$ | ✓ | ✓ | ✓ | ✓ |
| 6 | $B$ | ✓ | ✓ | ✓ | ✓ |
| 7 | $d, A$ | ✓ | ✓ | ✓ | ✓ |
| 8 | $d, B$ | ✓ | ✓ | ✓ | ✓ |
| 9 | $d, d_1$ | ✓ | ✓ | ✓ | ✓ |
| 10 | $d_1, A$ | ✓ | ✓ | ✓ | ✓ |
| 11 | $d_1, B$ | ✓ | ✓ | ✓ | ✓ |
| 12 | $d_1, d_2$ | | ✓ | ✓ | ✓ |
| 13 | $A, B$ | | | | ✓ |
| 14 | $d, d_1, A$ | ✓ | ✓ | ✓ | ✓ |
| 15 | $d, d_2, A$ | ✓ | ✓ | ✓ | ✓ |
| 16 | $d, A, B$ | | | | ✓ |
| 17 | $d, d_1, d_2$ | | ✓ | ✓ | ✓ |
| 18 | $d_1, d_2, A$ | | ✓ | ✓ | ✓ |
| 19 | $d_1, A, B$ | | | | ✓ |
| 20 | All | | | | ✓ |

GET TICKET / JOIN proves that (1) authentication via the user and the two devices is correct, and that (2) only the right group can be joined with a ticket. Table 1 shows the results of the proofs with $d_1$ and $d_2$ being the two devices for authentication. Important are cases 1 to 6 where the attacker controls up to one device. The protocol is even much stronger than this requirement and only fails if either both bridge IRS groups $A$ and $B$, or $d_1$ and $d_2$ are compromised.

For VOTE TO EXCLUDE / LEAVE, we proved that (1) a minimum of two votes is needed and that (2) the exclusion will only take place in the correct group. This is true for all cases apart from the ones where both bridge IRS groups are compromised (see Table 2).

The SEND protocol is trivial and therefore we did not formulate a proof in ProVerif. If device $d_\square$ only accepts messages signed with $k_\square$ and device $d_\triangle$ is not in possession of the key, isolation is achieved.

VOTE TO PROMOTE / BRIDGE JOIN proves that (1) one honest device voted for $d$ and at least two promote messages were received. All devices are allowed to vote, that means also potentially malicious devices cast votes. However, there must be at least one honest device agreeing to proof the absence of

Table 2: ProVerif Results: VOTE TO EXCLUDE / LEAVE

| No | Compromised Devices | Vote | Group 2 leave | $k_A/k_B$ secure | End reached |
|---|---|---|---|---|---|
| 1 | None | ✓ | ✓ | ✓ | ✓ |
| 2 | $d$ | ✓ | ✓ | ✓ | ✓ |
| 3 | $A$ | ✓ | ✓ | ✓ | ✓ |
| 4 | $B$ | ✓ | ✓ | ✓ | ✓ |
| 5 | $d, A$ | ✓ | ✓ | ✓ | ✓ |
| 6 | $d, B$ | ✓ | ✓ | ✓ | ✓ |
| 7 | $A, B$ | | | | ✓ |
| 8 | All | | | | ✓ |

Table 3: ProVerif Results: VOTE TO PROMOTE / BRIDGE JOIN

| No | Compromised Devices | Promote | Group 2 vote | $k_A/k_B$ secure | End reached |
|---|---|---|---|---|---|
| 1 | None | ✓ | ✓ | ✓ | ✓ |
| 2 | $d$ | ✓ | ✓ | ✓ | ✓ |
| 3 | $A$ | ✓ | ✓ | ✓ | ✓ |
| 4 | $B$ | ✓ | ✓ | ✓ | ✓ |
| 5 | $d, A$ | | ✓ | ✓ | ✓ |
| 6 | $d, B$ | ✓ | | | ✓ |
| 7 | $A, B$ | | | | ✓ |
| 8 | All | | | | ✓ |

attacks. Furthermore, we proved that (2) devices of another group cannot promote devices of the original group. This protocol only holds for the basic requirement (one compromised device) as displayed in Table 3. The reason is that a compromised $A$ can always leak its secrets to $d$ (case 5) and that also a compromised $d$ correctly receives the secrets of $A$ at the end of the protocol. It can then leak them to compromised $B$ (case 6). However, malware tolerance is still guaranteed because the protocol holds for one compromised device (case 1 to 4).

Since Group Key Agreement and IRS cryptography scheme are already proven by the original papers, VOTE TO EXCLUDE / BRIDGE LEAVE is equivalent to VOTE TO EXCLUDE / LEAVE and, therefore, does not need additional proofs.

All in all, our architecture is secure for one compromised device. As expected, none of the protocols holds if both bridge IRS groups are compromised.

## 4 Discussion

Our security analysis proves that our architecture is tolerant against attacks (malware-tolerant) and can even recover from infected bridge devices. Devices are isolated in their groups and can only communicate to another group if bridge devices allow this – similar to a firewall. This also enables the architecture to proxy vulnerable or outdated devices by simply moving them into an empty group. Services to the network of the device can be made available or turned off per connection.

### 4.1 Attacks on Multiple Devices

In order to control the whole network, an attacker would have to compromise the two bridge IRS groups $A$ and $B$ in the same time-period. The adversary could either compromise a device of each of the two IRS groups or compromise enough nodes to promote him or herself in multiple groups. The former is the worst case where the architecture is tolerant against only one compromised device in total. In the latter case, a maximum of one compromised device *per group* can be tolerated and half of those can be bridge devices (if they are in the same IRS group). If the threshold for elections is greater than two, even more compromised (non-privileged) devices can be tolerated. However, we assume a powerful attacker and thus the worst case is more realistic.

### 4.2 Attacks on one Device

Noisy attack: Let us assume the adversary compromised one light bulb with a buffer overflow or similar. The identity key of that light bulb and the group key of the light bulbs is compromised. The attacker can now participate (incl. read/write access) inside the light bulb group and he or she can communicate with the bridge group. Let us further assume the adversary then launches some attacks without focusing on stealth – like network scans, a spambot, crypto-miner, black hole attacks dropping all network traffic, or communication to a known Command-and-Control server. For simplicity we will now only discuss the network scan, the other attacks would be similar.

A network scan can be detected by the victim and all devices that route the traffic or are in the proximity of the adversary since WiFi is not limited through cables. For example there could be two devices around the compromised light bulb, three devices routing traffic, and the victim (as seen before by the red path in Fig. 1a). Not all of these six devices might notice the attack but if we assume a threshold of two votes to exclude, only 33% of those devices need to detect and report the attack through a vote to exclude to the bridges. All the rest of the network does not know of the attack but this is not necessary.

After receiving the two votes to exclude, the compromised light bulb will be quarantined through revoking the identity key of it and renewing the light bulb group key. If the light bulb was a bridge device, the bridge keys are also renewed.

Stealthy attack: If the adversary compromises a light bulb but stays hidden, he or she can eavesdrop, manipulate and attack the light bulb group and attack the limited interface of the bridge group.

Our architecture will not automatically defend against this, however, an adversary is restricted to the own group. A compromised bridge light bulb would give access to two groups (the light bulbs and the bridges) but cannot interfere with other groups. We estimate that the system will converge to a trusted state in the long run, because as soon as any attack becomes visible (e.g. through signature updates) the device will be automatically isolated. The infrastructure becomes a moving target for the adversary which is harder to compromise.

Physical attacks (clone attacks): An adversary who physically attacks a device and clones it, only gets access to one identity key. Since we incorporated the user into the GET TICKET protocol, we defend against this kind of attack.

Apart from this, it does not matter for the architecture, if the attacker compromises nodes physically or through e.g. a buffer overflow. A compromised node gives the attacker all the secrets of that node. Thus, physical attacks are equal to the attacks discussed above.

Denial of Service: DoS attacks targeting the cryptographic routines are unavoidable but they are easily identified, resulting in the misbehaving device being quickly removed from the network (similar to noisy attacks). If the DoS focuses on the compromised device (e.g. a CryptoLocker), that device is not available any more but the attack can only spread in that one group.

Phishing: Our architecture gives the user full access. By employing two devices in the authentication process, phishing attacks are more complicated. However, the user does have full control over the architecture and phishing could leverage this.

### 4.3 Performance and Adoption

A major concern for all types of network is performance. Our inter-group communication requires three symmetric encryptions until the session key is setup. Hence, devices which communicate a lot with each other should be organised in the same group, such that communication between the groups is an exception. With the expensive operations JOIN and LEAVE occurring seldom – during setup, failures, and attacks – the performance is approximately similar to symmetric key encryption. Adding devices to multiple groups would improve performance, but bypasses the bridge devices and undermines the distributed firewall. Non-privileged devices only need public key cryptography during setup and for voting. However, resource-constrained devices could rely on other devices to some point. Bridges rely on public key cryptography to add or remove devices but only one bridge of each IRS group needs to answer requests. Hence, these operations can be distributed. If overhead is a concern for bridges, we could promote more bridges to distribute computations more and lower the overhead at the single bridge.

Another concern is that one bridge device of group $A$ and $B$ has to stay online. Considering that fridge, smartphone, router, smart-meters, and smart thermostats are already online around-the-clock we expect this to be less of an issue when smart devices are widespread.

Lastly, also old devices which do not support the protocol could be assigned to a (possibly static) group. They can communicate with the rest of the network without supporting the protocol because the SEND operation is transparent to the sender and receiver. While these old devices do not gain all the benefits, they can rely on other devices to verify and run the network.

Setting devices up works in the same fashion as with normal WiFi networks apart from our password being longer. So, if a device (e.g. a smart light-bulb) has no interface, we would setup the password as before. For light bulbs this

usually includes connecting to it via a default WiFi and adding WiFi name and password of the home user. In our case, these are the group identifier instead of the WiFi name and a one-time password instead of the WiFi password. For less technically aware users, devices could be automatically allocated to groups depending on their identifier.

### 4.4 Improvements

We see two areas for future work. Firstly, the more expensive cryptography routines for GET TICKET, BRIDGE JOIN, and BRIDGE LEAVE could be improved. We imagine a combination of IRS and mediated RSA (mRSA) [5, 4]. In mRSA the private key is split into two additive shares, which could be split between bridge group $A$ and $B$ removing the need for two public keys. But, a new cryptographic scheme was beyond our scope.

Secondly, we would like to improve the usability of the GET TICKET procedure. Instead of employing two different devices, we could allow GET TICKET from one device (e.g. a smartphone) and store or re-compute old one-time passwords. If a password is used twice, either the setup device (the smartphone) or the new device (e.g. the new smart light bulb) misbehaved. At the moment, this would require to store all one-time passwords.

## 5 Related Work and Comparison

In the literature, mesh networks are usually analysed in their applied forms, i.e. Wireless Sensor Networks, Wireless Ad-Hoc Networks, Mobile Ad-Hoc Networks, and Vehicular Ad-Hoc Networks.

While there are more mesh architectures with hierarchies or clustered devices, usually their cryptographic key or the base station is the single point-of-failure. For brevity we only included a few.

Diop et al. [10] deployed isolation to WSNs by utilising secret sharing and a network-wide symmetric key. The base station uses secret sharing to setup keys for clusters. Each head of a cluster derives the keys for the sensor nodes. Through this key infrastructure, sensor data is delivered via the cluster heads to the base station and isolation is achieved between the clusters. The authors assume a trusted base station with an IDS, unlimited resources, and table of all nodes in the network. The base station is a single point-of-failure and the architecture is, thus, not malware-tolerant.

The SASHA architecture [3] proposes a self-healing approach for sensor networks that is inspired by the immune system. The objectives of this work are automatic fault recognition, adaptive network monitoring, and a coordinated response. To detect faults, the network has a definition of itself in the form of a neural network. Base station, Thymus system, and Lymph system are single points-of-failure making the architecture vulnerable to targeted attacks.

Posh [8] is a proactive self-healing mechanism for WSNs. The sink periodically re-initialises sensor nodes with a new key and a secret seed. The nodes then

also share some randomness with their neighbours to make it more difficult for the adversary to derive keys if nodes are compromised. Since the sink is in possession of all secrets, it can recompute the keys and read the data. However, the sink is a trusted entity and a single point-of-failure.

The authors [9] later improved their architecture with a moving target defence system. Data is moved either once or continuously, and then re-encrypted with one of three cryptographic schemes: symmetric encryption, symmetric encryption with key evolution, and asymmetric encryption. This way an adversary cannot easily delete data as the location of the data is unknown. Also in the extended architecture, the sink is assumed trustworthy which is not malware-tolerant.

The concept of intrusion-tolerant systems was already proposed and theoretically analysed by Verissimo et al. [19] They also created a self-healing, intrusion-tolerant system aimed at Industrial Control Systems (ICSs) [2]. A slightly different self-healing, malware-tolerant ICS was proposed in [7].

Sousa et al. [18] designed together with Verissimo an intrusion-tolerant, distributed replication system built on top of *critical utility infrastructural resilience information switches*, a firewall device which they previously developed. The approach uses proactive and reactive recovery to self-heal failed replicas. The devices are rejuvenated periodically and upon detection of malicious or faulty behaviour. To agree on firewall decisions, the replicas communicate through a synchronous, trusted channel between the replicas called wormhole. If a majority of replicas approve a network message, it is signed with a key unknown to the replicas in the wormhole subsystem. Local machines behind the firewall can verify the signature to identify trusted packages. The approach is similar to ours but, instead of deploying protection mechanisms in front of the network, we spread our firewall over the network and enforce it with cryptography. On the other side, our approach can only automatically quarantine devices. Self-healing would be optionally possible since it is orthogonal to our infrastructure but we doubt there is a general purpose solution to securely rejuvenate consumer devices of different manufacturers.

## 6 Conclusion

All in all, our architecture can provide isolation for flat, interconnected networks and networks where geographic and virtual layout differ. It enables them to automatically contain compromised devices while distributing trust over the entire architecture without single point-of-failure, i.e. malware tolerance.

## References


1. Amir, Y., Kim, Y., Nita-Rotaru, C., Tsudik, G.: On the performance of group key agreement protocols. ACM Transactions on Information and System Security (TISSEC) 7(3), 457–488 (2004)
2. Bessani, A.N., Sousa, P., Correia, M., Neves, N.F., Verissimo, P.: The crutial way of critical infrastructure protection. IEEE Security & Privacy 6(6), 44–51 (2008)

3. Bokareva, T., Bulusu, N., Jha, S.: Sasha: Toward a self-healing hybrid sensor network architecture. In: The Second IEEE Workshop on Embedded Networked Sensors. pp. 71–78. Citeseer (2005)
4. Boneh, D., Ding, X., Tsudik, G.: Identity-based mediated rsa. In: 3rd Workshop on Information Security Application [5]
5. Boneh, D., Ding, X., Tsudik, G., Wong, C.M.: A method for fast revocation of public key certificates and security capabilities. In: USENIX Security Symposium (2001)
6. Davis, D.: Defective sign & encrypt in s/mime, pkcs# 7, moss, pem, pgp, and xml. In: USENIX Annual Technical Conf., General Track. pp. 65–78 (2001)
7. Denzel, M., Ryan, M., Ritter, E.: A malware-tolerant, self-healing industrial control system framework. In: IFIP Advances in Information and Communication Technology ICT Systems Security and Privacy Protection. vol. 502. Springer (2017)
8. Di Pietro, R., Ma, D., Soriente, C., Tsudik, G.: Posh: Proactive co-operative self-healing in unattended wireless sensor networks. In: IEEE Symposium on Reliable Distributed Systems. pp. 185–194. IEEE (2008)
9. Di Pietro, R., Mancini, L.V., Soriente, C., Spognardi, A., Tsudik, G.: Playing hide-and-seek with a focused mobile adversary in unattended wireless sensor networks. Ad Hoc Networks (Elsevier) 7(8), 1463–1475 (2009)
10. Diop, A., Qi, Y., Wang, Q.: Efficient group key management using symmetric key and threshold cryptography for cluster based wireless sensor networks. Intl. Journal of Computer Network and Information Security 6(8), 9 (2014)
11. Dodis, Y., Franklin, M., Katz, J., Miyaji, A., Yung, M.: Intrusion-resilient public-key encryption. In: Cryptographers' Track at the RSA Conf. pp. 19–32. Springer (2003)
12. Dodis, Y., Franklin, M., Katz, J., Miyaji, A., Yung, M.: A generic construction for intrusion-resilient public-key encryption. In: Cryptographers' Track at the RSA Conf. [11], pp. 81–98
13. Franklin, M.: A survey of key evolving cryptosystems. Intl. Journal of Security and Networks 1(1-2), 46–53 (2006)
14. Hu, Y.C., Perrig, A.: A survey of secure wireless ad hoc routing. IEEE Security & Privacy 2(3), 28–39 (2004)
15. Itkis, G.: Intrusion-resilient signatures: generic constructions, or defeating strong adversary with minimal assumptions. In: Intl. Conf. on Security in Communication Networks [16], pp. 102–118
16. Itkis, G., Reyzin, L.: Sibir: Signer-base intrusion-resilient signatures. Advances in Cryptology–Crypto 2002 pp. 101–116 (2002)
17. Parno, B., Perrig, A., Gligor, V.: Distributed detection of node replication attacks in sensor networks. In: IEEE Symposium on Security and Privacy. pp. 49–63. IEEE (2005)
18. Sousa, P., Bessani, A.N., Correia, M., Neves, N.F., Verissimo, P.: Highly available intrusion-tolerant services with proactive-reactive recovery. IEEE Transactions on Parallel and Distributed Systems 21(4), 452–465 (2010)
19. Veríssimo, P., Neves, N., Correia, M.: Intrusion-tolerant architectures: Concepts and design. Architecting Dependable Systems pp. 3–36 (2003)