CrypTopology:
Plug, Play and Recover Key Management

Amir Herzberg and Yehonatan Kfir

Dept. of Computer Science Bar-Ilan University, Israel
herzbea@cs.biu.ac.il,yehonatank@gmail.com

Abstract. Research on establishing and maintaining secure communication, has two distinct categories: using *cryptography*, with *pre-shared or certified keys*, and using *known, redundant network topology*. We present the *CrypTopology model*, combining cryptography with topology, with benefits over the pure-crypto and pure-topology approaches. The model also considers *deployment* challenges, by taking into account legacy devices and routing, an aspect which is very relevant (but so far ignored) in topology-based protocols.

We use the CrypTopology model to study *key setup and management*. We present the CrypTop protocol, that allows easy *plug and play* key setup, between new devices and a trusted *authentication server* (whose public key is known). Furthermore, CrypTop limits the impact of key exposures: it ensures *proactive key refresh*, re-establishing security after exposure. In addition, CrypTop supports *incremental deployment*, and is effective even for partial deployment.

We analyze the properties of the CrypTop protocol and show sufficient topology conditions for its applicability. We prove its security against an attacker that is able control some of the devices in the network. We further present AR-CrypTop, an improvement of CrypTop that is secure even for *Adversarial Routing*.

1 Introduction

Cryptography is the main tool for protecting communication against strong eavesdropping and MitM adversaries. However, cryptography requires a party to have a key for each peer, obtained and authenticated directly or with the help of a trusted third party such as a certification-authority or key distribution center. This approach relies on *computational assumptions*, essentially, that the computational resources of the attacker are limited and cannot 'break cryptography'.

An alternative, first proposed in the seminal work of Dolev et al. [5], is to secure communication using knowledge of the *network topology*, instead of trusted third party or directly-shared keys (as in cryptographic solutions). This approach relies on different type of assumptions, namely, that the network is sufficiently redundant to prevent attacker from controlling 'too many' links or nodes.

It is interesting - and, we believe, useful - to study what can be achieved by *combining cryptography and topology.* Namely, we propose the *CrypTopology model*, combining cryptography with topology. This assumes computational-limitations of the attacker (and hardness of cryptographic functions), *together* with knowledge of the topology and limitation on the presence of the adversary (to a limited subset of the links and nodes).

The CrypTopology model applies to important practical scenarios - and allows to provide properties not achievable using only cryptography or only topology. Specifically, we present the CrypTopology-based key setup protocol, providing *'plug-and-play' key setup and refresh*, between 'unkeyed' devices and a special *authentication server* (whose public key is known), assuming known topology. CrypTopology-based key setup allows recovery from key exposure, ensuring *proactive security.* The use of (known, redundant) topology allows CrypTopology-based key setup to improve security compared to the use of 'only' cryptography; and the use of cryptography allows us to significantly reduce the overhead and redundancy-requirements compared to topology-only solutions.

The CrypTopology model also considers *deployment* and *routing* challenges. Namely, we take into account that the network is likely to contain *legacy devices* which do not follow the proposed protocol (in our case, CrypTopology-based key setup); such devices only provide specific routing functionality. This is in contrast to existing topology-based works, including [5], which naively assume 'source-routing', where senders can control the path of messages they send. Source routing is simple and easy to use but rarely available in practical networks. In practice, networks usually use shortest-path routing; furthermore, routing protocols may be vulnerable to attacks. We consider routing as part of the CrypTopology model, and in particular, consider both shortest-path routing as well as adversarial routing.

Support for incremental deployment and realistic routing is critical for practical applications, and has significant impact on the design and analysis of topology-based protocols.

Our model and protocols are applicable to practical scenarios, since in many networks, topology is known and quite stable, with considerable redundancy (e.g., for resiliency). *Key setup* is a significant challenge in deploying cryptography within such organizational networks. Each device that supports cryptography needs to be securely initialized with keys. This large-scale initialization is a challenging operation to manage.

Possibly even more significant, after deployment, the keys need to be re-initialized if key exposure is suspected. Many systems support *forward security*, which protects the confidentiality of past traffic from later exposure of keys; but *proactive security*, i.e., automated recovery of security for communication *after* key compromise, is a harder challenge, addressed only by the (theoretical) solution of [4], which relies on alerting by devices - an unrealistic assumption for many types of devices. Our method exploits the topology to provide a more practical solution, allowing completely-automated secure key refresh, and where detection and alert are done by the authentication server.

Another important requirement we introduce and support is *incremental deployment*. Namely, it is perfectly possible to deploy the protocols in different devices at different times, i.e., gradually. This is another aspect of the *plug and play* functionality, making our results applicable to deployment of security into existing networks, with acceptable startup challenge.

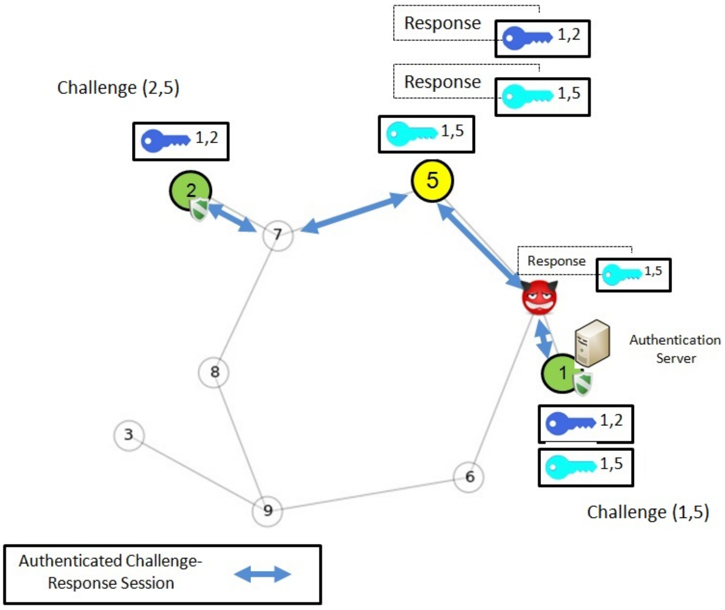

Fig. 1. A simple example: the IEEE 9-bus model [14] of a small power communication network. The CrypTopology-based key setup protocol provides plug-and-play initial key setup and proactive key refresh. The protocol uses multiple authenticated challenge-response sessions, between a client (Device 5) and several upgraded devices (Devices 1 and 2). In this way, the network provides security from an attacker that controls part of the routes between the client and the upgraded devices

One important example for relevant networks, are SCADA and Industrial Control networks. Those networks present stable and known topology [1], from one hand, and from the other hand, they include many geographically distributed, unmanned devices, making it hard to rely on manual process for key setup and refresh.

For example, consider the network in Figure 1, which is based on the IEEE 9-bus model [14] - a known topology of power networks. This system is built from 9 communication devices, with each one of them representing a communication device at a power site. For this example, Devices 1 and 2 are assumed to be upgraded to support the protocol, and Device 1 is the authentication server. The server knows the topology, as shown in the figure. When Device 5 is upgraded, the server will ask it for evidence that it is connected with a path of length 2, to Devices 1 and 2. Only after receiving this evidence, will the server authenticate Device 5 and its cryptographic keys.

Contributions. This work makes the following contributions:

- We present the CrypTopology model, allowing the use of topology properties to improve deployability and/or security of cryptographic systems.
- Our model supports *incremental deployment* with legacy devices, and different routing models, including *shortest path* and *adversarial* routing. The model also supports the TTL mechanism, deployed in IP devices.
- We present the CrypTopology-based key setup plug-and-play key setup and refresh protocol, allowing setup of cryptographic keys between an authentication-server and unkeyed, unmanned devices. CrypTopology-based key setup works for the typical case of shortest-path routing.
- We extend CrypTopology-based key setup to support also adversarial routing, taking advantage of the TTL mechanism of IP devices.
- Both protocols ensure *proactive recovery*, i.e., the system remains secure even if all components may be corrupted by the attacker, as long as the number of concurrently-corrupted components is appropriately limited.

1.1 Related Work

Several works [2, 6, 7, 11, 15–17] study the use of network topology to create a shared secret between parties. However, these works assume 'source routing', i.e., senders have complete control over the route, and do not support legacy devices and realistic routing mechanisms (shortest path and adversarial), and do not handle key exposures (cf. to our proactive recovery property).

On the other hand, existing proposals for plug-and-play key setup has with weaker security guarantees, since they do not utilize the topology. Such proposals include BTNS and other 'leap of faith' designs [12], and the use of anonymity network to detect attacks [8].

Our work also presents a formal model for key-setup, following the seminal paper of Bellare et al. [3]. We extend their model to include the network topology and routing method.

2 Model

2.1 Network Model

We model the communication network as an undirected hyper-graph, $G = (V, E)$, where $N = |V|$ denotes the numbers of devices (nodes), and E is a set of hyper edges representing the connections between devices. Some edges are simple edges representing point-to-point communication, and some are hyper-edges representing a connection to multiple devices on the same interface.

A device can send messages to other devices. The message may pass through several intermediary devices that act as routers, before reaching its destination. Every device can block, pass, or change messages that pass through it.

Every device in the network has an identifier that uniquely represents it. An example of such an identifier can be a combination of the device IP and MAC address. For simplicity, we denote the identifier of device $v \in V$, by v.

Devices and Adversary In every network, we assume there is a group of devices that do not support cryptographic modules. We call this group *legacy devices* and denote it by $L \subset V$.

Another group in the network contains the *upgraded devices*, $U \subset V - L$, which support cryptographic modules and need to be initialized with keys. These devices must also have a recovery plan to receive a new key in case of suspected key leakage.

One of the devices in the network, is the authentication server s, which has a known public key, $s.pu$. It also has a shared secret key with each of the upgraded devices. Using these keys, the server can send and receive encrypted and authenticated messages. It is assumed that the authentication is the only device that can not be *compromised device* [1].

On each graph $G = (V, E)$, we define a coloring function $\phi : V \times V \to \{Legacy, Upgraded, Compromised\}$, which defines the type for each device in the network. Using this definition, a *legacy* or *upgraded* device that was compromised will change its color to show it is *compromised*.

We consider an attacker $\mathcal{A}$, who controls all *compromised devices*[2]. The attacker tries to: disrupt the key setup for an upgraded device, register its own key for some device, or learn the key setup in the server for some upgraded device.

We define n_A*-nodes attacker* as an attacker that controls n_A devices. From the coloring function definition, it is clear that $n_A = |A| = |\{v \in V \ s.t. \ \phi(v) = Compromised\}|$ devices.

The attacker is able to initiate, delay, block, or manipulate messages that pass through its devices. In addition, the attacker can eavesdrop on messages in the entire network, including messages that do not pass through its devices.

Routing Model The *routing method* defines the way each device forwards incoming messages. We consider the following routing methods:

Source-routing: Each device can set the route in the network for messages that it initiates. The route contains the sequence of devices that relay the message until it reaches its destination.

Shortest-path routing: Messages are sent on the shortest-path between the source and destination device. Routing in a shortest-path network is formulated as a function $\Re : V \times V \to V$, which receives the current device and the destination, and returns the neighbor of the current device, to which the message is forwarded $\Re(current, destination)$, such that the sequence of forwarding

[1] Compromised server is discussed in future works. One trivial way to handle that is to use redundant servers.

[2] Controlling a device effectively controls all of its links. For simplicity, we do not discuss an attacker that is able to control only specific links.

from source to destination is always the shortest path. If there is more than one shortest-path route between the source and destination, the shortest-path routing function consistently chooses the same route.

Adversarial routing: The routes *should have been* shortest-path, but they may have been changed by an attacker. In this network, in addition to the default shortest-path routing function $\Re$, there is an adversarial routing function $\Re_{\mathcal{A}}$.

Legacy and *upgraded* devices always send messages according to the routing method; this may be the routing list in source-routing networks, the $\Re$ function in shortest-path network, or $\Re_{\mathcal{A}}$ in adversarial routing networks. In contrast, *compromised* devices are not bound by the routing method and can freely select the edge from which to forward each message.

2.2 Protocol and Asynchronous Execution Model

Our model is based on that of Bellare et al. [3] for message-driven-protocols. For simplicity, we use a more specific definition for the key-setup protocol and extend their model to support several modes of the protocol on different parties.

A *CrypTopology-based key setup* protocol π is a message-driven-protocol [3] that has three types of participants in its execution:

Server - A non-compromised device that is initialized with a public key $s.pu$ and correlated private key $s.pr$. The server is also initialized with the network topology $G = (V, E)$, the location of the upgraded devices, the routing method $\rho \in \{source, shortest\text{-}path\}$, the security parameter 1^l, and $\Re$ for non-source routing networks. At the end of the protocol execution, the server has three possible outputs: *Alert*; and *Success* with a pair (k_s^{OUT},c) of key k_s^{OUT} of device c;

Client - An *upgraded* device that is initialized with the server's public key $s.pu$, the routing method $\rho \in \{source, shortest\text{-}path\}$, and the security parameter 1^l. This is the only upgraded device that does not have a shared secret key with the server. At the end of a successful execution, this device will create and register such a secret key, k_c^{OUT}.

Collaborator - An *upgraded* device that has a shared secret key with the server $k_{s,i}$; this key is different for each collaborator.

The goal of the protocol is to set the same secret key at the server and client: $k_s^{OUT} = k_c^{OUT}$. The ability to securely set such a shared key with a device $v \in V$, depends on the topology of the network $G = (V, E)$, the type of each device ϕ, the routing method ρ, and the routing function $\Re$. Let $P(v, G, \phi, \rho, \Re)$ be a *topology availability predicate* that returns 1 if several topology conditions are met for device $v \in V$.

We define the *availability* of protocol π with respect to predicate P, as the fraction of all devices in the network that have $P(v, G, \phi, \rho, \Re) = 1$.

Execution of a CrypTopology-based key setup protocol depends on the network properties and on the attacker capabilities. As input, the execution receives the attacker algorithm $\mathcal{A}$, a CrypTopology-based key setup protocol π, a topology predicate P, and a random key $k \xleftarrow{\$} \{0,1\}^l$. We denote this execution by $\mathbf{EXEC}(\mathcal{A}, \pi, P, k)$.

We describe now the main ideas of the asynchronous execution process. The details are described in the full version of the paper [9]. In Section 5 we describe a synchronous execution model in order to describe additional time-depended properties.

The execution process is *adversarial* in the sense that the attacker $\mathcal{A}$ chooses all the network parameters: ρ, ϕ and the topology $G = (V, E)$. In addition, the attacker chooses one *upgraded* device as the client c and one *non-compromised* device as the server.

The output of the execution is one of the following *results*: (1) *"Failure"* - if the server registers a key that is not the same as the client key (probably because of an attacker); (2) *"Alert"* - if the server detects an attacker that prevented the key setup; (3) *("Success", k_c^{OUT}, $\sigma_{\mathcal{A}}$)*, where $\sigma_{\mathcal{A}}$ is the attacker state - if the server registers the same key as the client, k_c^{OUT} or if $P(c, G, \phi, \rho, \Re) \neq 1$.

2.3 Protocol Properties

We define the following properties of CrypTopology-based key setup protocols.

Secrecy. A key-setup protocol ensures *secrecy* if no PPT attacker can retrieve any information about the key from the protocol messages. In other words, there is no probabilistic polynomial-time attacker that can distinguish the key from a randomly-generated string of the same length. Formally:

Definition 1 ***[Secrecy]***

Protocol π ensures secrecy *with respect to predicate P, if* $|Pr\left[IND_{\mathcal{A},\mathcal{D},\pi,P}(l) = 1\right] - \frac{1}{2}|$ *is a negligible function (in security parameter l), for all PPT attacker $\mathcal{A}$ and PPT distinguisher $\mathcal{D}$, and where $IND_{\mathcal{A},\mathcal{D},\pi,P}(l)$ is defined in Algorithm 1.*

```
Indistinguishability Experiment $IND_{\mathcal{A},\mathcal{D},\pi,P}(l)$

1. $\mathcal{D}$ chooses $k_0, k_1 \xleftarrow{\$} \{0,1\}^l$.
2. A random bit is chosen, $b \xleftarrow{\$} \{0,1\}$
3. A successful execution, with key $k_b$, is chosen randomly,
   $(\text{"}Success\text{"}, k_c^{OUT}, \sigma_{\mathcal{A}}) \xleftarrow{\$} \mathbf{EXEC}(\mathcal{A}, \pi, P, k)$  $where\ k = k_b$
4. Return 1 if $\mathcal{D}(k_c^{OUT}, \sigma_{\mathcal{A}}) = b$, and 0 otherwise.
```

Algorithm 1: Indistinguishability experiment

Correctness. A key-setup protocol ensures *correctness* if, whenever the server outputs a key k_c^{OUT} for specific client c, then, with overwhelming probability, c outputs the same key k_c^{OUT}. In addition, if the server outputs Alert, then, with overwhelming probability, there is an attacker in the network (i.e., no false alerts).

Formally, we require that any polynomial-limited time-attacker will have a negligible probability of preventing key setup, without being detected. In other words, we require a negligible probability of the execution's output being Failure.

Definition 2 ***[Correctness]*** *Protocol π ensures correctness, with respect to predicate P, if for all PPT attackers $\mathcal{A}$, there exists a negligible function negl s.t.:*

$Pr(\textbf{\textit{EXEC}}(\mathcal{A}, \pi, P, k) = \text{"}Failure\text{"}) < negl(l)$,

where the probability is taken over the random coins used by $\mathcal{A}$ and $\textbf{\textit{EXEC}}(\mathcal{A}, \pi, P, k)$.

Guaranteed Key-Setup. A key-setup protocol ensures *guaranteed key-setup* with respect to predicate P, if, with overwhelming probability, executions terminate successfully (and correctly) - even in the presence of an attacker. In an asynchronous model, the adversary schedules the message delivery. Hence, it can prevent completion of the protocol simply by delaying messages ('forever'). Thus, for this property we must make an assumption about the message delivery.

Definition 3 ***[Eventually Delivering]*** *Attacker $\mathcal{A}$ is eventually delivering if it delivers all the messages between non-compromised parties. This attacker can only permanently block messages that pass through compromised devices.*

Using these definitions, we can define the following property:

Definition 4 ***[Guaranteed Key-Setup]*** *Protocol π ensures guaranteed key-setup, with respect to predicate P, if for all security parameters 1^l and eventually delivering attackers $\mathcal{A}$:*

$\textbf{\textit{EXEC}}(\mathcal{A},\pi,P,k) = \{\text{"}Success\text{"}, k_c^{OUT}, \sigma_{\mathcal{A}}\}$.

3 Crypto-Topology Key-Setup Protocol (CrypTop)

In this section, we present an implementation for a Crypto-Topology protocol, called CrypTop.

The goal of CrypTop is to provide cryptographic keys to upgraded devices, without requiring manual installation. The protocol is executed whenever new keys are needed, and specifically upon device upgrade or to recover from possible compromise of the device's secret keys.

The protocol uses a public key encryption scheme ξ and a MAC scheme $\mathcal{M}$, as defined by Katz and Lindell [10]. We denote a CrypTop that uses these schemes as $CrypTop^{\xi,\mathcal{M}}$. We will omit $\mathcal{M}$ when we want to emphasize that it is not relevant for the discussion.

After creating the symmetric key k, the server authenticates the key holder's identity. Using challenge-response sessions, the server validates that the key holder is the client that it claims to be. The challenge-response sessions validate the topological location of the key holder. In this way, the key holder is authenticated under several conditions, which will be discussed further.

3.1 Protocol Design

Before initiating the protocol, the client $c \in V$ is loaded with the server's public key $s.pu$ and the security parameter 1^l. To initiate the protocol, the client generates a random key $k \overset{\$}{\leftarrow} \{0,1\}^l$.

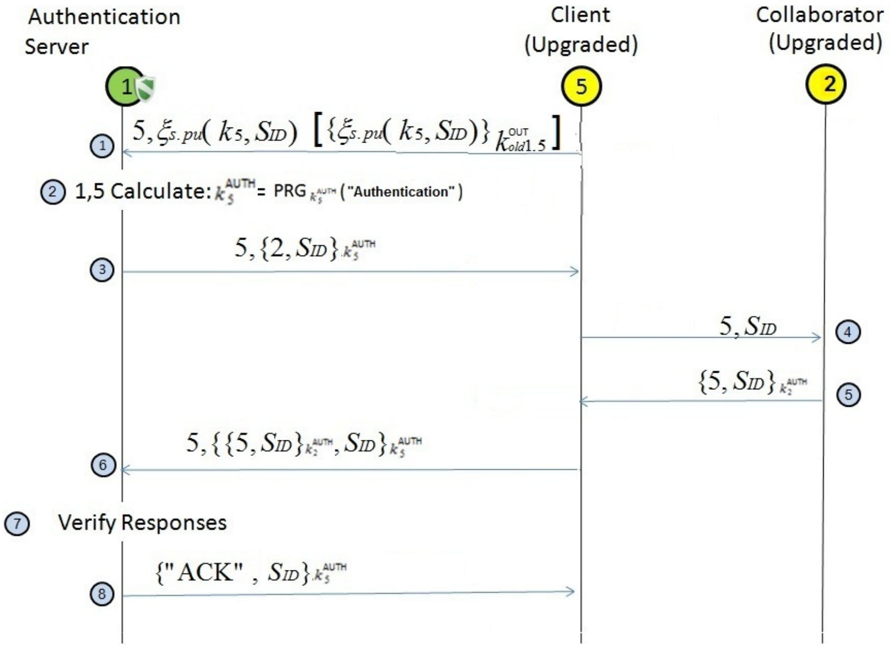
Authentication Server
Client (Upgraded)
Collaborator (Upgraded)
1
5
2
1 $5, \xi_{s.pu}(k_5, S_{ID})\ [\{\xi_{s.pu}(k_5, S_{ID})\}_{k^{OUT}_{old1,5}}]$
2 1,5 Calculate: k_5^{AUTH} = PRG $_{k_5^{AUTH}}$ ("Authentication")
3 $5, \{2, S_{ID}\}_{k_5^{AUTH}}$
4 $5, S_{ID}$
5 $\{5, S_{ID}\}_{k_2^{AUTH}}$
6 $5, \{\{5, S_{ID}\}_{k_2^{AUTH}}, S_{ID}\}_{k_5^{AUTH}}$
7 Verify Responses
8 $\{\text{"ACK"}, S_{ID}\}_{k_5^{AUTH}}$

Fig. 2. Example for a successful key-setup session, for the network in Figure 1. Device 5 receives keys, after two challenge-response sessions, with Device 1 and Device 2. Device 1 is the authentication server. In square brackets, proactive fields, which is described in Section 5

In addition, the server is loaded with the network graph $G = (V, E)$, the device-type (coloring) function ϕ , the routing $\Re$, and the number of compromised devices that it should handle, n_A.

The client activates the protocol session by sending an *activation message*, m_{Init}, to the server. The activation message contains a randomly chosen session ID S_{ID} and the device random key k. The session ID does not consider secure/secret from an attacker. It is used by the server to detect all the messages from the same authentication session.

All these values are sent encrypted using ξ with the server public key $s.pu$. The client also sends its identification (e.g., its IP address, as described in Section 2.1), unencrypted.

Using the device identifications, the server finds the location of the device in the topology. These identifications are not considered trusted and the server will have to validate the device's claimed location.

Using the generated key k, and a pseudo-random-function (PRF), the server and client derive a new symmetric authentication key $k_c^{AUTH} = PRF_k(''Authentication'')$, as well as a shared secret key $k_c^{OUT} = PRF_k(''Client\_Key'')$. k_c^{AUTH} will be used with the MAC scheme $\mathcal{M}$ to authenticate all the messages between the server and the client during the key setup protocol. k_c^{OUT} will be used as the registered key of the client.

After generating k_c^{AUTH}, the server selects a group of $2n_A + 1$ collaborators from the upgraded devices, $U_c \subset U$, that are placed on disjoint routes between the server and the client.

If there are no $2n_A + 1$ such collaborators, the server will simply select a group of $n_A + 1$ collaborators. One of the collaborators can be the authentication server itself.

If the number of available collaborators is below $n_A + 1$, the server will not continue with the protocol execution and will neglect the client request for key setup.

In order to validate the client location, the server sends the client the identification of the chosen collaborators U_c. For each collaborator, the client sends the session ID as a challenge. In response, each collaborator sends back the session ID and the client identification, authenticated with the key the collaborator has with the server.

After the client receives all the responses, it sends them to the server.

A challenge-response session is defined as *successful*, if (1) the session ID and the client ID in the responses are as expected; (2) all the messages between the server and the client are authenticated with the key k_c^{AUTH}; (3) each response is authenticated with the appropriate collaborator key k_i^{AUTH}.

The protocol behavior is defined relative to the upper bound of devices that can be under control of the attacker n_A. Using this parameter, the protocol defines three behaviors:

Key Registry at the Server: If $n_A + 1$ sessions succeeded, the server will register the key k_c^{OUT} for device c, and will send an *ACK message* to the client, through all of the collaborators. The ACK message contains the string "ACK" and the session ID S_{ID}.

Key Registry at the Client: When one ACK message is received from the server and that ACK message is authenticated with the key k_c^{AUTH}, the client will register its long time key k_c^{OUT}.

We denote the maximal network delay between two devices as T_{delay}. For simplicity, we assume that the processing time of messages in the devices is zero.

Alerting: The server will wait up to $4T_{max}$ time after sending the challenges to the client. The server output will be Alert if there are no $n_A + 1$ successful challenge-response sessions.

An example for a successful key setup session for $n_A = 1$ can be seen in Figure 2.

3.2 Protocol Properties

In this section we present the topology conditions necessary to ensure that the CrypTop protocol has the properties defined in Section 2.3. Formal proofs are available in Appendix A.

The properties of the protocol are defined with respect to the network topology. For n_A-node attacker, we define the following topology-predicates:

Detection-based predicate: $P^{DET}(v, G, \phi, \rho, \Re) = 1$ only if (1) $\rho \in \{source, shortest-path\}$, (2) and v has n_A+1 disjoint routes to *upgraded* devices. In this section, we explain that if P^{DET} holds for device v, then either v will set up keys successfully or the server will raise an alert (correctly detecting at least one compromised device in the network).

Full-availability predicate: $P^{FULL}(v, G, \phi, \rho, \Re) = 1$ only if (1) $\rho = source$ and v has $2n_A+1$ disjoint routes to the security server; or (2) $\rho = shortest-path$ and v has $2n_A+1$ disjoint routes that are built from routes to collaborators and from the collaborators to the authentication server. In this section we explain that if P^{FULL} holds for device v, then v will always set up keys successfully.

Using these predicates we discuss the properties of the protocol.

The secrecy of the protocol is based on the security of the public encryption scheme ξ and the MAC scheme $\mathcal{M}$. If ξ is secure against an adversary who tries to retrieve the key from its encrypted messages, then the CrypTop protocol ensures that the key k_c will not be leaked to the attacker.

The correctness property is based on the topology properties of the client devices in the network. Devices that have $n_A + 1$ disjoint routes to upgraded devices can be authenticated securely. The reason is that the attacker has only n_A devices, which do not allow him to be on all of the routes. Thus, at least one message from the original client will be received at the server. If an attacker attempts to register such a device, the server will detect different attempts for key registrations for the same client. Different keys for the same device will cause the server to raise an alert.

Clients that have $2n_A + 1$ disjoint routes to the server will always be able to receive keys. This guaranteed key-setup property is based on the topology properties. An attacker with n_A devices will be able to fulfil only n_A challenge-response sessions and to block the same number of sessions of the client, even in adversarial-routing. Under the same conditions, the client will be able to fulfil at least $n_A + 1$ sessions. The authentication server will be able to detect the "real" client as it will have an additional successful challenge-response session.

Following this discussion, we phrase several theorems. The detailed proofs are in Appendix A:

Theorem 1. ***[Secrecy]****: For every CPA-secure public-key scheme ξ (and every MAC scheme $\mathcal{M}$), protocol CrypTop$^{\xi,\mathcal{M}}$ ensures secrecy, as defined in Definition 1.*

Theorem 2. ***[Correctness]****: For every CPA-secure public-key scheme ξ, and MAC-secure scheme $\mathcal{M}$, with respect to predicate P^{DET}, protocol CrypTop$^{\xi,\mathcal{M}}$ ensures correctness as defined by Definition 2.*

Theorem 3. ***[Guaranteed Key-Setup]****: Protocol CrypTop, with predicates P^{FULL}, ensures the* guaranteed key-setup *property, as defined in Definition 4.*

Notice that *correctness* and *guaranteed key-setup* depend on the topology predicates; thus, they are not ensured in adversarial networks. We will address this in the next section.

4 Adversarial Routing CrypTop (AR-CrypTop)

The CrypTop protocol presented in Section 3 does not ensure correctness and guaranteed key-setup in networks with adversarial routing. This is because the authentication server does not loaded with the adversarial routing and hence, cannot ensure disjoint routes.

For example, consider an adversarial routing in the network of Figure 1. An attacker that controls Device 9 will be able to authenticate as Device 5, simply by routing the responses from Devices 1 and 2 to pass through Device 9. In this way, Device 9 can provide the needed authenticated responses and authenticate as Device 5.

In this section we present AR-CrypTop, an enhancement of CrypTop for networks with adversarial routing.

For AR-CrypTop discussion, we are assuming that the network is an IP network, and we use the TTL field [13] of IP packets. We define a *TTL-Network* as one in which devices obey the 'TTL rules' as follows:

Definition 5 ***[TTL-Network]*** *A network is a* TTL-network *if: (1) Every non-compromised device decreases the TTL field of packets that pass through it by 1, and discards them if TTL=0, and (2) when a non-corrupt device initiates a message, it uses the initial TTL of 255.*

Following the model of the attacker from Section 2.1, it is clear that in *TTL-network*, the attacker is able to change the TTL-field of messages that pass through Compromised devices.

4.1 Protocol Design

The goal of AR-CrypTop is to provide cryptographic keys to upgraded devices, without requiring manual installation. AR-CrypTop ensures correctness and proactive security in networks with adversarial routing.

AR-CrypTop assumes a TTL-network as in Definition 5. The topology authentication is based on authenticated TTL field in the responses received from the collaborators, and comparing their values to the ones expected based on the topology. As we describe, comparing the authenticated TTL field to the expected value limits the capabilities of an attacker in networks with adversarial routing.

The protocol messages and design are based on CrypTop , and the protocol activation and sequence are similar to CrypTop. In this section we describe only the differences.

The first difference is that all the messages sent from the client include the maximal value of TTL, ttl_{MAX}. These messages are routed through the network to the collaborators or to the authentication server. During the routing, the TTL fields are decreased, based on the TTL-network assumption. We use ttl_i to denote the TTL value of the message that reaches collaborator i.

A collaborator in AR-CrypTop is an *upgraded device* that already has a shared-secret key with the server.

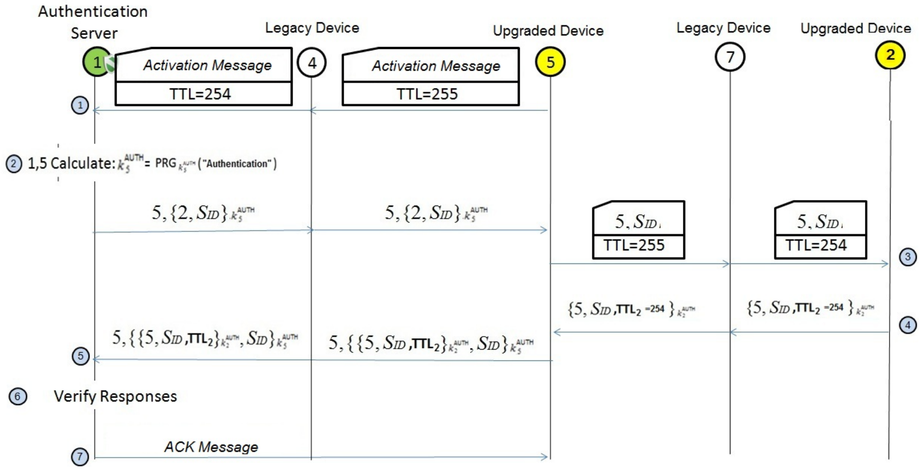
Authentication Server
Legacy Device
Upgraded Device
Legacy Device
Upgraded Device
1
4
5
7
2
Activation Message
TTL=254
Activation Message
TTL=255
1
2 1,5 Calculate: k_5^{AUTH} = PRG$_{k_5^{AUTH}}$("Authentication")
$5, \{2, S_{ID}\}_{k_5^{AUTH}}$
$5, \{2, S_{ID}\}_{k_5^{AUTH}}$
$5, S_{ID}$
TTL=255
$5, S_{ID}$
TTL=254
3
$\{5, S_{ID}, \mathrm{TTL}_2 = 254\}_{k_7^{AUTH}}$
$\{5, S_{ID}, \mathrm{TTL}_2 = 254\}_{k_2^{AUTH}}$
4
$5, \{\{5, S_{ID}, \mathrm{TTL}_2\}_{k_2^{AUTH}}, S_{ID}\}_{k_5^{AUTH}}$
$5, \{\{5, S_{ID}, \mathrm{TTL}_2\}_{k_2^{AUTH}}, S_{ID}\}_{k_5^{AUTH}}$
5
6 Verify Responses
7
ACK Message

Fig. 3. AR-CrypTop message sequence with the additional TTL field, based on the network in Figure 1. *Activation* and *ACK messages* are similar to CrypTop.

When collaborator i receives a challenge, it checks the value of the TTL field, ttl_i. The collaborator responds with an authenticated messages containing the TTL value of the received message, in addition to the client identification and the session ID S_{ID}). The response is validated at the authentication server using the key shared with the collaborator.

After the client receives all the responses, it sends them to the authentication server.

An AR-CrypTop challenge-response session is defined *successful*, if: (1) it is a successful session in CrypTop as defined in Subsection 3.1, and (2) the TTL-fields in all the authenticated responses are aligned with the distance between the collaborator and the client.

The protocol behavior is defined relative to the number of devices that can be under the control of the attacker n_A; this is similar to CrypTop.

4.2 Protocol Properties

In this section we present the topology conditions necessary to ensure that the AR-CrypTop protocol has the properties defined in Section 2.3. Formal proofs are available in the full version of the paper [9].

Let $c_1, ..., c_{N_c}$ be the group of collaborators (which are *upgraded* devices).

We denote the group of devices that are located at a distance smaller than or equal to l from collaborator c_i as V_{l,c_i}. We denote the length of the shortest path between the client c to collaborator c_i as l_{c,c_i}.

Lemma 1. *A response from collaborator c_i that has a TTL of ttl_i, indicates that the sender of the challenge (the client device or a compromised device) is not farther than $255 - ttl_i + 1$ hops from c_i.*

The proof is straightforward from the TTL definition, and it is provided at Appendix A.

According to Lemma 1, an attacker can produce the same TTL as the client, only if there is a compromised device that is located at a distance that is equal to or less than the distance between the client and the collaborator.

Hence, for given network properties $G, \phi, \rho, \Re$, and for each collaborator c_i, the group V_{l_{c,c_i},c_i} can provide the same TTL as the client c. We require that there will be no more than $n_A - 1$ devices, that are part of V_{l_{c,c_i},c_i}, for all the collaborators.

We denote by A_v the minimal number of devices that can provide the same TTL fields as a device v, for a given group of upgraded devices.

For attacker $n_A - node$, we define the following topology-predicate:

Detection-based adversarial-route predicate: $P^{DET-ROUTE}(v, G, \phi, \rho, \Re) = 1$ only if (1) $\rho \in \{adversarial\}$, (2) and $|A_v| > n_A$

The correctness property is based on the topology properties of the client devices in the network. Every device $v \in V$ that has $|A_v| > n_A$, can be authenticated securely. The reason is that the attacker has only n_A devices, which do not allow him to provide all the required TTL fields. Thus, at least one message from the original client will be received at the server. If an attacker attempts to register such a device, the server will detect different attempts for key registrations for the same client. Different keys for the same device will cause the server to raise an alert.

Theorem 4. *[Correctness]: For every CPA-secure public-key scheme ξ and MAC-secure scheme $\mathcal{M}$, with predicate $P^{DET-ROUTE}$, protocol $AR - CrypTop^{\xi,\mathcal{M}}$ ensures correctness as defined by Definition 2.*

Notice that guaranteed key-setup is not ensured in adversarial routing, even with AR-CrypTop. In such routing, the attacker can always change the routes of the messages from the client, so they appear unsuccessful for the server. For example, it can increase the route those messages take, causing them to have a TTL of a much more distant device than the client.

5 Preserve Security and Proactive Recovery

5.1 Synchronous Execution Model and Security Definitions

In this section we define important properties that depends on the time parameter in the execution. Hence, we add a short description of a synchronous model of the protocol execution that was described in Section 2.2.

In the synchronous model, every sender measures the time that has passed since sending the message. When that time reaches a timeout value, the sender will no longer wait for a response for that message, and will output *Timeout.* All the parties in this model have synchronized clocks that proceed at the same rate.

We denote a synchronous execution $\mathbf{EXEC^{SYN}}$ $(\mathcal{A},\pi,P,k,t_0)$, where t_0 denotes the starting time of the execution. The rest of the parameters are as the same as in the asynchronous execution that was described in Section 2.2.

In the synchronous model, we divide the network's coloring function ϕ into time slots. Each time slot is T_P long. We denote the coloring function at time t as ϕ_t.

We define a n_A*-active-attacker* as follows. During each T_P period, the attacker is able to control a maximum of n_A devices. While controlling a device, the attacker can retrieve this device's key; the key information remains known to the attacker, even after he releases control of the device.

5.2 Proactive Security

A key-setup protocol ensures proactive security if it has a mechanism for periodically recovering from key compromise, without the need to detect an attacker in the network.

Using proactive mechanism, the protocol is able to limits the maximal number of devices/keys that can be compromised by n_A-active-attacker, and to ensure the correctness of the protocol over time. That, even in the presence of a n_A-active-attacker.

First property we define is *to preserve the security of the device*:

If at some time a device succeed in receiving a key, than in the future, as long as the attacker did not compromise that device, the server will be able to provide new key to the device or to detect an attack on that device.

Definition 6 ***[Preserve Security]***
Protocol π *ensures* preserving security with respect to predicate P, *if for every time t and for every device v that was not compromised and fulfil:* $\boldsymbol{EXEC^{SYN}}$ $(\mathcal{A},\pi,P,k,t_0)$*=("Success",*$k_c^{OUT}$*,*$\sigma_{\mathcal{A}}$ *) it holds:*
$Pr(\boldsymbol{EXEC^{SYN}}$ $(\mathcal{A},\pi,P,k,t_0+T_P)$*="Failure")* $<$ *negl(l)*

Second property we define is *proactively recovering* from a compromise of a device, and, under several topology conditions, providing new secret key to that device.

Definition 7 ***[Proactive Recovery]***
Protocol π *ensures* proactive recovery with respect to predicate P, *if for every time* t_0*, it ensures Correctness and Guaranteed Key Setup as defined at 1 at time* t_0+T_P *, for every device v that fulfil:*
$\phi_t(v) = Compromise$*,* $\phi_{t+T_P}(v) = Upgraded$
and $P(v) = 1$ *for all of the period between* t_0 *to* t_0+T_P*.*

Notice that a single execution of CrypTop and AR-CrypTop for key setup, does not ensure security against n_A-active-attacker. In presence of n_A-active-attacker, the identity of the compromised devices can be changed over time, while the key of every device that was compromised will be known to the attacker even after he release the compromised device. After enough time, the attacker will be able to retrieve the keys of all the upgraded devices in the network.

5.3 Proactive CrypTop and AR-CrypTop

In order to handle n_A-active-attacker and ensure proactive security, we describe a proactive variant of both protocols. The proactive variant is different in the following aspects:

Periodical activation - The protocol is re-activated in every device every T_P time-units. We assume that the execution duration time of the protocol is negligible with respect to T_P.

Message authentication of activation message At every re-activation of the protocol, the activation message sent from the device to the server is authenticated using the shared authentication key exchanged during the previous successful run. If the activation message received does not include a valid message authentication code (MAC), then the recipient detects an attack ('Alert' event).

If a device received a key which was not compromised at time t, i.e., a successful run, then it will always be able to refresh its key at time $t + T_P$ or it will create an alert. Using the previous key $k^{OUT}_{Old,c}$, the device will be able to identify itself in front of the authentication server. If it will not be able to provide enough successful sessions (probably because of an attacker, since at time t the device was able to provide enough successful sessions), the server will raise an Alert.

Refreshing the key (whether using the topology or the previous key) provides proactive security. The new key is always chosen randomly, with no deterministic way to relate it to the old key. In addition, periodically refreshing the key ensures that the network will recover from a compromised key, even without detecting the specific key that was leaked.

Following this discussion, we can phrase the following theorem:

Theorem 5. ***[Preserve Security and Proactive Security]****: Protocols CrypTop and AR-CrypTop, with respect to predicate P^{FULL}, ensures* preserve security *and* proactive security *properties, as defined in Definitions 6 and 7.*

6 Conclusions

We present the CrypTopology model, combining cryptography with topology. The model considers deployment challenges, by taking into account legacy devices and routing. We use this model to define the CrypTopology-based key setup task. We then present the CrypTop protocol, that facilitates easy plug and play key setup and refresh. In addition, we present AR-CrypTop, an improvement of CrypTop that is secure even against adversarial routing. We analyse the security of CrypTop, for different routing models.

References

1. Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. A first look into scada network traffic. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 518–521. IEEE, 2012.
2. Paolo Barsocchi, Stefano Chessa, Ivan Martinovic, and Gabriele Oligeri. A cyber-physical approach to secret key generation in smart environments. *Journal of Ambient Intelligence and Humanized Computing*, 4(1):1–16, 2013.
3. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 419–428. ACM, 1998.
4. Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.
5. Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.
6. Matthias Fitzi, Matthew Franklin, Juan Garay, and S Harsha Vardhan. Towards optimal and efficient perfectly secure message transmission. In *Theory of Cryptography*, pages 311–322. Springer, 2007.
7. Jokin Garay, Clint Givens, and Rafail Ostrovsky. Secure message transmission with small public discussion. *Information Theory, IEEE Transactions on*, 60(4):2373–2390, 2014.
8. Yossi Gilad and Amir Herzberg. Plug-and-play IP security. In *Computer Security–ESORICS 2013*, pages 255–272. Springer, 2013.
9. Amir Herzberg and Yehonatan Kfir. Topology-based plug-and-play key-setup ,https://eprint.iacr.org/2016/060. 2015.
10. Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC Press, 2014.
11. Kaoru Kurosawa and Kazuhiro Suzuki. Almost secure (1-round, n-channel) message transmission scheme. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 92(1):105–112, 2009.
12. Viet Pham and Tuomas Aura. Security analysis of leap-of-faith protocols. In *SecureComm*, volume 96, pages 337–355. Springer, 2011.
13. Jon Postel. Rfc 791: Internet protocol, september 1981. *Darpa Internet Protocol Specification*, 1990.
14. A. B. Smith. IEEE std c37. 1-1994, IEEE standard definition, specification, and analysis of systems used for supervisory control, data acquisition, and automatic control. *IEEE Power Engineering Society*, 1994.
15. K Srinathan, Arvind Narayanan, and C Pandu Rangan. Optimal perfectly secure message transmission. In *Advances in Cryptology–CRYPTO 2004*, pages 545–561. Springer, 2004.
16. Matthias Wilhelm, Ivan Martinovic, and Jens B Schmitt. Secure key generation in sensor networks based on frequency-selective channels. *Selected Areas in Communications, IEEE Journal on*, 31(9):1779–1790, 2013.
17. Sencun Zhu, Shouhuai Xu, Sanjeev Setia, and Sushil Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. In *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, pages 326–335. IEEE, 2003.

A Detailed Proofs

Theorem 1

Proof. Assume to the contrary that exists $\mathcal{A}_1, \mathcal{A}$, s.t. for all negligible function $negl(l)$:

$|Pr\,[IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l) = 1] - \frac{1}{2}| \ > \ negl(l)$

Using those attackers, we define an attacker on the encryption scheme ξ, $\mathcal{A}_\xi$. Using this attacker, we will prove a contradiction to the CPA-secure property of ξ.

We denote the PRF that is being used by π as $PRF : \{0,1\}^l \to \{0,1\}^{l'}$, $l < l'$.

Assume that PRF is an ideal random function U, $U : \{0,1\}^l \to \{0,1\}^{l'}$, $l < l'$. If the property holds for U and not for PRF, then it is easy to build a distinguisher between PRF and U - which is a contradiction to the assumption that PRF is a PRF. Thus, it is sufficient to assume that PRF is ideal.

Following the CPA experiment $PubK^{cpa}_{\mathcal{A}_\xi,\xi}(l)$ [10] :

1. Keys $s.pu, s.pr$ are chosen randomly.

2. The adversary $\mathcal{A}_\xi$ is given as input 1^l, the public key $s.pu$ and oracle access to the encryption scheme. $\mathcal{A}_\xi$ will choose two activation messages m_0, m_1.

3. A random bit $b \xleftarrow{\$} \{0,1\}^l$ is chosen, and then a ciphertext $c = \xi_{s.pu}(m_b)$ is computed and given to the adversary $\mathcal{A}_\xi$.

4. The adversary $\mathcal{A}_\xi$ executes the process $EXEC^{\mathcal{A}_\xi}$ $(\mathcal{A}, \pi, P, k_b)$ until the output is "Success". $\mathcal{A}_\xi$ returns $b' = \mathcal{D}(k, \sigma_{\mathcal{A}})$.

5. The output of the experiment is 1 if b=b', and 0 otherwise.

According to the assumption:

$Pr\,[\,\mathcal{D}(k_c^{OUT}, \sigma_{\mathcal{A}}) = b\,] \ > \ negl(l) \Rightarrow Pr\,[\,PubK^{cpa}_{k_c^{OUT},\mathcal{A}_\xi,\xi}(l) = 1\,] \ > \ negl(l)$

and this is contradiction to the assumption that ξ is CCA secure. Thus, if ξ is CPA-secure then $|Pr\,[IND_{\mathcal{A},\mathcal{D},\pi,P}(l) = 1] - \frac{1}{2}| \ < \ negl(l)$.

Thus, for every CPA-secure ξ, protocol CrypTop ensures secrecy.

Theorem 2

Proof. According to the predicate P^{DET}, there are at least $n_A + 1$ disjoint routes between the client c and the server.

According to the protocol design **Key Registry at the Server**, if the server registers a key, it sends the authenticated ACK message through all of the collaborators, through at least $n_A + 1$ disjoint routes.

Thus, if the server registers a key k_s^{OUT}, at least one ACK message had reached the client, and the client registers a key k_c^{OUT}. If not, the attacker was able to block all the ACK messages from $n_A + 1$ disjoint routes with its n_A devices. Hence, at least one attacker device was on more than one route. This contradicts the disjoint routes assumption.

According to the protocol design **Key Registry at the Client**, the client registers a key if it receives even one authenticated ACK message from the server.

Assume in negative that there exists a PPT attacker $\mathcal{A}$ s.t. $Pr(\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l) = "Failure") > negl(l)$.

If $\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l)$=Failure, then $k_c^{OUT} \neq k_s^{OUT}$. In this case, the server did not register the same key as the client. Thus, the attacker $\mathcal{A}$ was able to (1) create and authenticate the ACK message without knowing the key k_c^{OUT} or (2) change and authenticate the key agreement message, or (3) retrieve the key k_c^{OUT}. Each operation is done with a non-negligible probability greater than $negl(x)$.

Assuming (1) or (2): The attacker $\mathcal{A}$ was able to create message m' and tag tag' for a message he did not see before. This is in contradiction to the assumption that $\mathcal{M}$ is MAC secure.

Assuming (3): The attacker was able to retrieve the key k_c^{OUT} from the activation message. This contradicts the secrecy property.

Thus, for all PPT attackers $\mathcal{A}$, there exists a negligible function $negl(l)$ s.t. $Pr(\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l) = "Failure") < negl(l)$.

Definition 8 ***[Bounded Termination]*** *CrypTopology-based key setup π is a bounded termination protocol if, for each topology $G = (V, E)$, there exists a number $M(G)$ s.t. for all protocol executions, and the number of messages that will be sent is less than $M(G)$.*

Theorem 6. ***[Bounded Termination]****: Protocol CrypTop ensures* Bounded Termination *property, as defined at Definition 9.*

Proof. Let t_0 be the time the execution process started, and let T_{delay} be the maximal delay of message between neighbor devices. We will prove that the execution time is bounded by $T_{bounded} = 6\,|V|\,T_{delay}$.

The longest route a message can pass is a route that includes all the devices in the network. Thus, the maximal time delay of a message from a sender to a receiver device is $|V|\,T_{delay}$. We assume that the processing time of a message, at each device, is zero. For simplicity, we will assume that every device in the network waits that period for receiving response, even if the routes for its message's destination is shorter that the maximal route.

According to the CrypTop design there are 6 synchronous transactions (can be seen in Fig. 2). Each transaction, as explained, is bounded by $|V|\,T_{delay}$.

Thus, the maximal time that it takes to the protocol to execute is: $6\,|V|\,T_{delay}$

Theorem 3

Proof. First, we will prove that CrypTop is bounded termination protocol by the number: $2 + 4\,|V|$.

Let $G = (V, E)$ be the network graph. According to the protocol details, the client sends a key request message. Than, the server sends the client list of collaborators. The number of collaborator is limited by the number of devices in the network, $|V|$. The client initiate challenge-response session with each collaborator. and the number of messages is twice (for challenge and response messages with each device).

The client send all the responses through all its neighbours, and hence, this limit the number of messages to $|V|$. Upon successful authentication at the server, the server initiate ACK message through all the routes to the device. Those messages number is also bounded by the number of devices in the network.

Summarizing the upper limit for messages: $1+1+2|V|+|V|+|V| = 2+4|V|$.

Thus, CrypTop is a Bounded Protocol. We will now prove that CrypTop with predicate P^{FUL} is always finished with key setup.

If $P^{FULL}(c, G, \Phi, \rho, \Re) = 1$, then there are at least $2n_A + 1$ disjoint routes between the client c and the server s.

Because of the disjoint routes, attacker that control n_A devices will be able to manipulate or complete maximum n_A challenge-response sessions. There will still be $n_A + 1$ routes between the server and client, without any attacker node. Using the $n_A + 1$ responses, the server will be able to distinguish the client from the attacker, and to authenticate the client.

Now, it is left to prove that the server will receive the $n_A + 1$ responses. Since the attacker is eventually delivering, all the messages on the routes that it does not control, will eventually reach their destination. This includes the challenge-responses sessions, and the ACK from the server to the client.

This complete the proof.

Theorem 5

Proof. Let $\mathcal{A}$ be an attacker that is able to compromise n_A devices at T_P time. Let $lifetime < T_P$ be the time that was set by CrypTop that after it the client must request for new key. We will prove that can not be more than n_A compromised devices in the system.

Assume in negative that at time t_0 there were more than n_A compromised devices. Since the attacker is able to compromise maximum n_A devices at T_P time, than there is at least one device v' that was compromise more than T_P time earlier than t_0. Before compromised, v' could be Legacy or Upgraded device.

If v' was Legacy than the attacker did not compromise any key. By the attacker definition, it can not control more than n_A devices, and hence v' is no longer compromised.

If v' was Upgraded: According to CrypTop design, the server set a key for v' with a lifetime. After that lifetime, v' will have to request for a new key.

Since $lifetime < T_P$, v' requested for a new key at time t_R, where $t_0 - T_P < t_R < t_0$. Thus, v' is no longer Compromised, not its keys are known to the attacker.

Thus, the maximal number of compromised devices (or their keys) is not greater than n_A.

Lemma 1

Proof. Every device in the network that processes the packet must decrease the TTL received from the sender by at least one, $\triangle_i \geq 1$. The maximal TTL for a packet is 255. Hence,

$$ttl = 255 - \sum_{i=1}^{N} \triangle_i + 1 \leq 255 - N + 1 \Longrightarrow N \leq 255 - ttl + 1$$

where N is the distance of the sender in hop counts.

B Execution Model

B.1 Asynchronous Execution

For the detailed execution model, we define additional device type, which called *trusted device*. Those devices are highly secure, and assumed to be not-compromised. In addition, *trusted devices* do not have to obey the routing function, and allowed to send messages according to their inner protocol. We exclude this type of devices from the main paper due to page limitations.

On each graph $G = (V, E)$, we define a coloring function $\phi : V \times V \rightarrow \{Legacy, Upgraded, Trusted, Compromised\}$, which defines the type for each device in the network.

The details of the topology-based execution are in Algorithm 2.

For each of the defined properties, the attacker's goal is to create an execution process whose output contradicts one of the protocol requirements. To achieve that, the attacker $\mathcal{A}$ is allowed to choose all the network conditions: the network graph, the coloring function, the routing method, the client device c, and the server device s.

Let $G = (V, E), \phi$ be the network topology and the coloring function chosen by the attacker.

In addition to these conditions, the attacker chooses the client device $c \in V - T$, such that $P(c, G, \phi, \rho, \Re)$=1. It also chooses the server s to be one of the trusted devices in the network.

The routing function $\Re$ is chosen by the attacker, according to the network routing method.

If the network routing is *shortest-path routing*, $\mathcal{A}$ will choose a shortest-path tree for the message routing $\Re = \Re_0$.

If the network routing is *adversarial*, then the attacker will provide its desired adversarial routing function $\Re = \Re_A$, in addition to providing a shortest-path tree routing $\Re_0$.

During the initialization phase (Algorithm 3), the server receives the network properties with the non-adversarial routing function $\Re_0$, the security parameter 1^l, and the topology predicate P. It creates a pair of private and public keys, $s.pr, s.pu$. The public key $s.pu$ is given to the client for its initialization process, along with the routing method and the security parameter 1^l.

The key-setup execution process consists of a sequence of activations of π within different devices - which include the client c and the server s.

The activations are controlled and scheduled by the attacker, who also decides which incoming messages or external requests the activated party will receive.

Every message m that is sent by a party contains the sender device IP, the destination device IP, the next hop device IP, the TTL field, and a random string *payload* that should reach the destination device. For a source-routing network, each message also contains the route of the message.

In order to send a random string *payload* to device d, a party s adds the message m to a set of pending messages M. The message's next hop will be to the next device that should receive the message.

Whenever $\mathcal{A}$ activates a party v on some incoming message m, it must be that m is in the set M and that v is the device in the next hop field of message m. Upon activation, the party adds a group of messages M' to M.

Furthermore, m is now deleted from M. If v is not the destination device of the message m, then a new message m' will be added to M (Algorithm 4). The next hop field of m' will be the neighbor of v that should receive that message, according to the routing method. The payload, the source, and the destination device of m' will be identical to m. In TTL networks, the TTL field will be decreased by 1, and the message will be added only if the TTL field is greater than 0.

$\mathcal{A}$ is not required to maintain the order of the messages, nor is it bound by any fairness requirement on the activation of parties. By definition 3, an *Eventually Delivering* attacker is required to deliver all the messages between non-compromised parties, and it can only block messages that pass through compromised devices.

Beyond activating parties, the adversary $\mathcal{A}$ can also corrupt parties. Upon corruption, $\mathcal{A}$ learns the entire current state of the corrupted party. From this point on, $\mathcal{A}$ can add to M any (fake) messages from the corrupted party. $\mathcal{A}$ can block or change messages that pass through corrupted devices.

B.2 Synchronous Model

We denote the maximal network delay between two devices as T_{delay}. For simplicity, we assume that the processing time of messages in the devices is bounded by this value.

The longest route a message can pass through is a route that includes all the devices in the network. Thus, the maximal time delay of a message from a sending to a receiving device is $|V|\,T_{delay}$. We assume that the processing time of a message at each device is zero. Thus, the maximal time that will be passed from sending a message till receiving a response is $T_{max} = 2\,|V|\,T_{delay}$. For simplicity, we assume that every device in the network waits that period to receive a response, even if the route for its message's destination is shorter that the maximal route.

Using the synchronous model we define the bounded termination property:

Bounded Termination. - A key-setup protocol ensures bounded termination if the protocol's execution time is bounded, possibly as a function of the network topology $G = (V, E)$.

Definition 9 ***[Bounded Termination]*** *Let t_0 be the time the execution game started and $G = (V, E)$ the network topology.*

Protocol π ensures bounded termination, if there exists $T_{EXEC}(G)$ s.t. for every attacker $\mathcal{A}$, and the execution is finished after time $t_0 + T_{EXEC}(G)$:

$\boldsymbol{EXEC^{SYN}}(\mathcal{A},\pi,P,1^l,t_0)) \in \{"Alert","Success","Failure","Timeout"\}$

```
EXEC($\mathcal{A}$,$\pi$,$P$,$1^l$)
// Initialization - see Algo. 3
$\{G, \phi, \rho, \Re, A, c, s, \sigma_{\mathcal{A}}, \sigma_c, \sigma_s, m_{Init}, k_c^{OUT}\}$ = Init_Execution($\mathcal{A}$,$\pi$,$P$,$1^l$)
$M = \emptyset$
Add_Message($m_{Init}$,$M$) // see Algo. 4
while True do
    $M' = \emptyset$
    $\hat{m} = \mathcal{A}(\sigma_{\mathcal{A}})$
    switch $\phi(M[\hat{m}].next\_hop)$ do
        case Compromised do
            $\{M', \sigma_{\mathcal{A}}\} = \mathcal{A}(\sigma_{\mathcal{A}}, M[\hat{m}])$ $M[\hat{m}] = \emptyset$
        end
        case Trusted OR Upgraded do
            $v = M[\hat{m}].next\_hop$
            $\{M', \sigma_v, k_s^{OUT}, c', isAlert\} = \pi(\sigma_v, M[\hat{m}])$
            if isAlert then
                Return ("Alert",$\sigma_{\mathcal{A}}$)
            end
            if $v = s$ AND $k_s^{OUT} \neq NULL$ AND $c = c'$ then
                if $k_s^{OUT} = k_c^{OUT}$ then
                    Return ("Success",$k_c^{OUT}$,$\sigma_{\mathcal{A}}$)
                else
                    Return ("Failure",$\sigma_{\mathcal{A}}$)
                end
            end
            $M[\hat{m}] = \emptyset$
        end
        case Legacy do
            Decrement $M[\hat{m}].ttl$ by 1
            if $M[\hat{m}].ttl = 0$ OR $M[\hat{m}].next\_hop = M[\hat{m}].destination$ then
                $M[\hat{m}] = \emptyset$
            else
                // The message should be routed to the next device
                if $\rho = source$ then
                    // In source-routing the messages are route according to the
                       routing list. See section 2.1
                    $M[\hat{m}].next\_hop = M[\hat{m}].route[next\_hop]$
                else
                    // adversarial or shortest-path routing
                    $M[\hat{m}].next\_hop = \Re(M[\hat{m}].next\_hop, M[\hat{m}].destination)$
                end
            end
        end
    end
    foreach $m \in M'$ do
        Add_Message($m$,$M$)
        $\sigma_{\mathcal{A}} = \mathcal{A}(\sigma_{\mathcal{A}}, m)$
    end
end
```

Algorithm 2: Execution Process

C Experimental Evaluation

In this section we evaluate several practical aspects related to CrypTop. Since there is no public information on ICS control networks, we assumed that the topology of the control network is similar to the power system topology. This is a reasonable assumption, since the communication lines often pass through the

```
Init_Execution($\mathcal{A}$,$\pi$,$P$,$1^l$):

$k \xleftarrow{\$} \$\{0,1\}^l$ OR receives as an input (for the Secrecy property only).

/* The attacker chooses the network properties, the client, and the server.          */
$\{G = (V, E), \phi, \rho, c \in V, s \in V, \sigma_{\mathcal{A}}\} \leftarrow \mathcal{A}(P, 1^l))$
s.t.:
$\phi : V \to \{Legacy, Trusted, Compromised, Upgraded\}$
$\rho \in \{source, shortest\_path, adversarial\}$
$\phi(s) = $ Trusted
$P(c, G, \phi, \rho) = 0$ AND $\phi(c) = $ Upgraded
IF one of the above does not holds, RETURN ("Success",$k$,$\sigma_{\mathcal{A}}$)

$\{\sigma_s, s.pr, s.pu\} \leftarrow \pi.Init\_Server(G, \phi, \rho, P, 1^l)$

$S_{ID} \xleftarrow{\$} \{0,1\}^l$
$\{\sigma_c, k_c^{OUT}\} \leftarrow \pi.Init\_Client(s.pu, k, \rho, 1^l)$

/* The server public key is known to the attacker                                    */
$\sigma_{\mathcal{A}} \leftarrow \sigma_{\mathcal{A}} \cup \{s.pu\}$

/* Shared keys with the server are loaded on each trusted device                     */
foreach $\{t \in V | \phi(t) = Trusted, t \neq s\}$ do
    $k_t^{AUTH} \xleftarrow{\$} \{0,1\}^l$
    $\sigma_t \leftarrow k_t^{AUTH}$
    $\sigma_s = \sigma_s \cup k_t^{AUTH}$
end

if $\rho \in \{shortest\_path, adversarial\}$ then
    $\mathcal{A} \to \Re_0 : V \times V \to V$
    Validate $\Re_0$ is shortest path routing. If not, return 0.
    $\sigma_{\mathcal{A}} = \sigma_{\mathcal{A}} \cup \Re_0$ $\sigma_s = \sigma_s \cup \Re_0$
    if $\rho = adversarial$ then
        $\Re_A \leftarrow \mathcal{A}, s.t. \Re_A : V \times V \to V$
        $\sigma_{\mathcal{A}} = \sigma_{\mathcal{A}} \cup \Re_A$
        $\Re = \Re_A$
    else
        $\Re = \Re_0$
    end
end

Return $\{G, \phi, \rho, \Re, A, c, s, \sigma_{\mathcal{A}}, \sigma_c, \sigma_s, m_{Init}, k_c^{OUT}\}$
```

Algorithm 3: Initializing Process Execution

same infrastructure as the power lines. For the power system, we used several IEEE benchmark topologies: IEEE 300, IEEE 118, and IEEE 57 bus systems.

Initially we present the methodology we used to model the communication network from the power system.

Next, we evaluate the ways to choose the first devices to upgrade to use the protocol. Since CrypTop assumes collaborators, it is better to choose the first devices to upgrade as devices that will allow key-setup for as many other devices as possible. We will evaluate two methods for choosing those devices: choosing devices with the higher edge degree and choosing randomly.

At last, we evaluate the availability of CrypTop on different network sizes and different routing methods. We will show that in the tested networks, more than 60% of the devices in the network can use CrypTop for receiving keys, in the presence of an attacker that controls a single device in the network.

```
Procedure Add_Message($m$,$M$)
    Add to $M$ element with:
        $source = m.source$
        $destination = m.destination$
        $payload = m.payload$
        $ttl = m.ttl$
        if $\rho = source$ then
            $route = m.route$
            $next\_hop = m.route[m.source]$
        else
            $next\_hop = \Re(m.source, m.destination)$
        end
```

Algorithm 4: Message Handling

C.1 Methodology

A power system is built from a number of electrical buses that requires monitoring and control. Each bus is operated by several field devices (such as PLCs) that rare esponsible for the physical measurements and physical state of the bus. Those field devices operation is controlled and monitored from a centralized server called "control server".

In order to communicate with a field device (e.g. sending command or receiving measurements) the control server uses a communication network, that is built from communication devices, namely routers.

We assume that each group of field devices, that controls a single bus, is connected the same router. By this assumption, every electrical bus has a single router in the communication network.

There are several ways to implement connection between routers. A frequently used method is to pass the communication cables on the same pols as the electrical cables. In that way, to reduce the cost of another pols set for the communication cables.

Following this implementation method, we assume that for buses that are electrically connected, their routers are also directly connected.

Using those assumptions modeled the communication network, based on the electrical network. Let $G = (V, E)$ be the control network, where V is the set of devices (e.g. routers), and E are the edges. Device v_i is responsible on managing electrical bus i (e.g. connecting it to other buses, measuring its voltage), those, there is a vertex for each bus. v_i and v_j are connected in the graph, only if there is a connection between bus i and bus j in the electrical system.

As an example for this modeling method, consider the IEEE 9 bus case, shown on Figure 4. Each bus represent a vertex in the graph. Edges are communication connections between buses. In that way, the communication network graph $G = (V, E)$ for IEEE9 was created.

The routing method ρ of the networks chosen to be shortest-path or source-route. In the shortest-path routing, each edge considered to be with weight of 1.

For each network G, routing method ρ and method for choosing trusted devices we evaluate the number of devices that have $n \in \{2, 3, 4, 5\}$ disjoint routes to trusted devices. Those values represent detection-based availability against attacker with {1,2,3,4} devices respectively.

The number of devices is presented as a percentage from the total number of devices in the network.

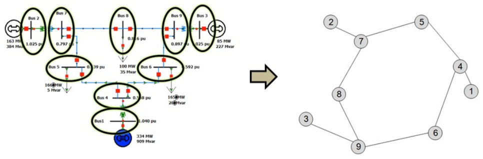

Fig. 4. Extracting the control network from the power network

C.2 Availability

We defined the CrypTop properties with respect to topology predicates P^{DET} and P^{FULL}. These predicates require that the number of disjoint routes be greater than the number of compromised devices. In this section we evaluate the number of devices with $P^{DET}(v, G, \phi, \rho, \Re) = 1$. In other words, these are devices with $n_A + 1$ disjoint routes, where n_A is the number of compromised devices in the network.

The routing method ρ of the networks was chosen to be shortest-path or source-route. In the shortest-path routing, each edge is considered to have a weight of 1.

The coloring function ϕ required that we choose a group of trusted devices and n_A compromised devices. The group of trusted devices we chose were the devices with the higher degree of edges. The size of the trusted devices group was from 1 to 5.

For each device v, we assumed that the n_A compromised devices are located in the worst case, with respect to v's location. In other words, the compromised devices are located on n_A disjoint routes between v and the group of trusted devices (if v has at least n_A such disjoint routes). The number of compromised devices was chosen to be from 1 to 4, $n_A \in \{1, 2, 3, 4\}$

The results are shown in Figures 5 and 6.

From the results, it can be seen that even with a small number of trusted devices (about 5 devices), most of the devices in the network can use CrypTop in the presence of one compromised device or one compromised link. In IEEE 118 with source routing, more than 80% of the devices can use CrypTop, while in shortest-path routing, more than 65% of the network can use CrypTop.

Another conclusion from the results is that the networks present significantly more percentage of devices that available for CrypTop in the presence of one compromised device, $n_A = 1$, than for a higher number of compromised devices. Motivated by this conclusion, we compared the availability for CrypTop between

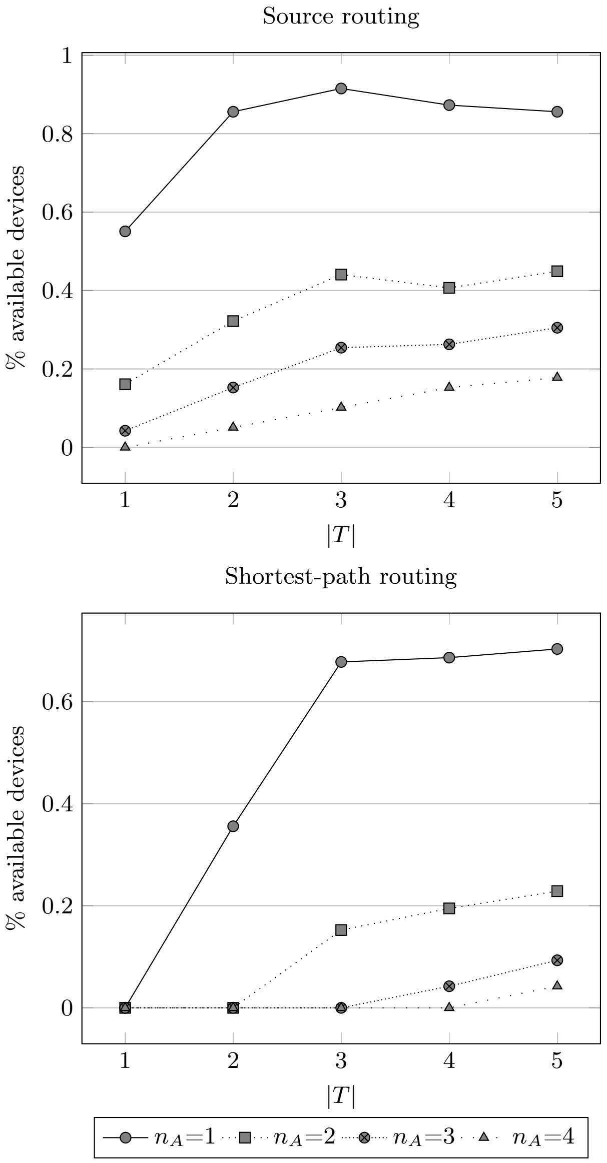

Fig. 5. Percentage of devices available for secure ToBKES in IEEE 118. Compromised devices were chosen to be at the worst-case position.

different networks. The results can be seen in Figure 9. These results indicate that for most of the evaluated network topologies, even 5 trusted devices are sufficient to have an availability of more than half of the network, in the presence of a single compromised device.

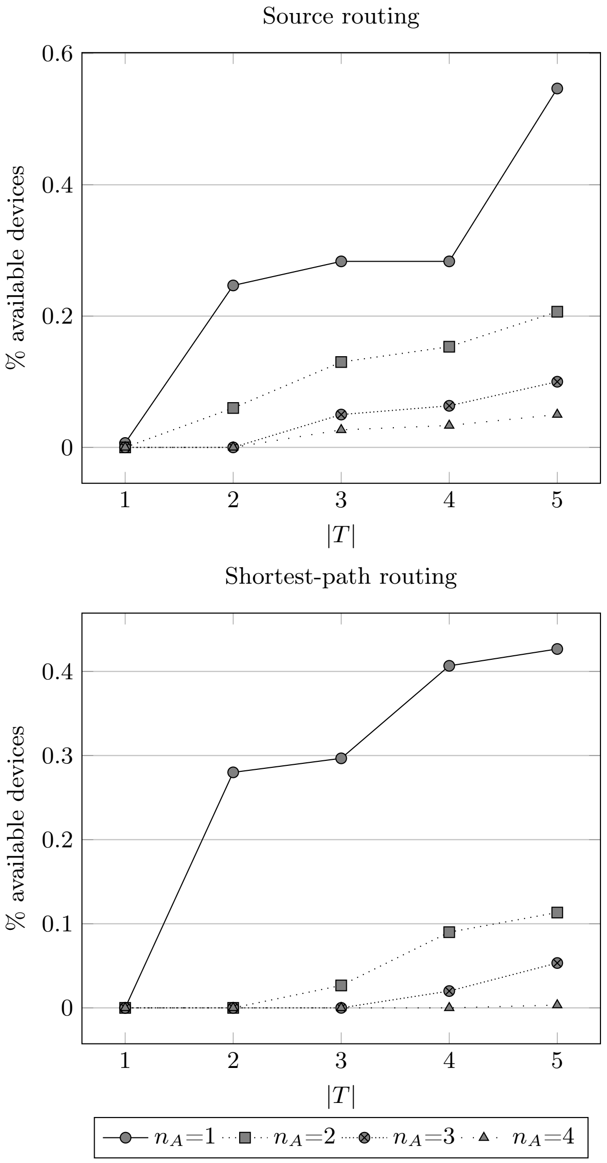

Fig. 6. Percentage of devices available for secure ToBKES in IEEE 300. Compromised devices were chosen to be at the worst-case position.

C.3 Proactive Security

In this section we evaluate the applicability of the proactive mechanism. For the networks IEEE 300 and IEEE 118, we assumed a large number of upgraded devices that support CrypTop: 30%,40% and 50% of the devices in the network.

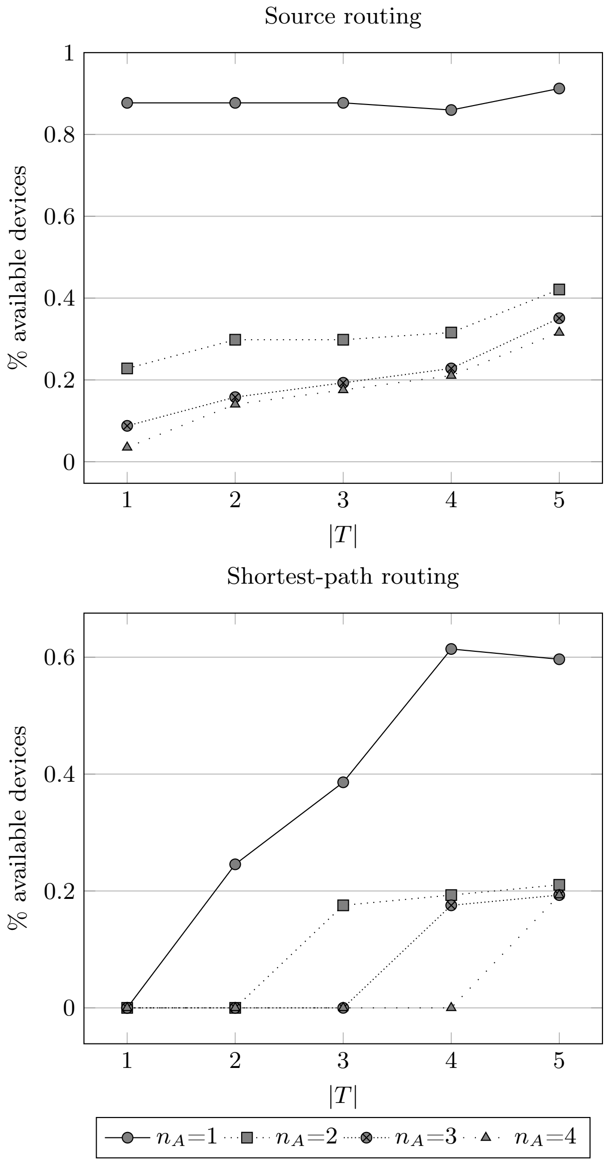

Fig. 7. Percentage of devices available for secure ToBKES in IEEE 57. Compromised devices were chosen to be at the worst-case position.

We also assumed that one of those devices is trusted and considered to be the authentication server.

For each device in the network, we evaluated the number of compromised devices that are needed in order to prevent it from receiving a refresh key. Similar

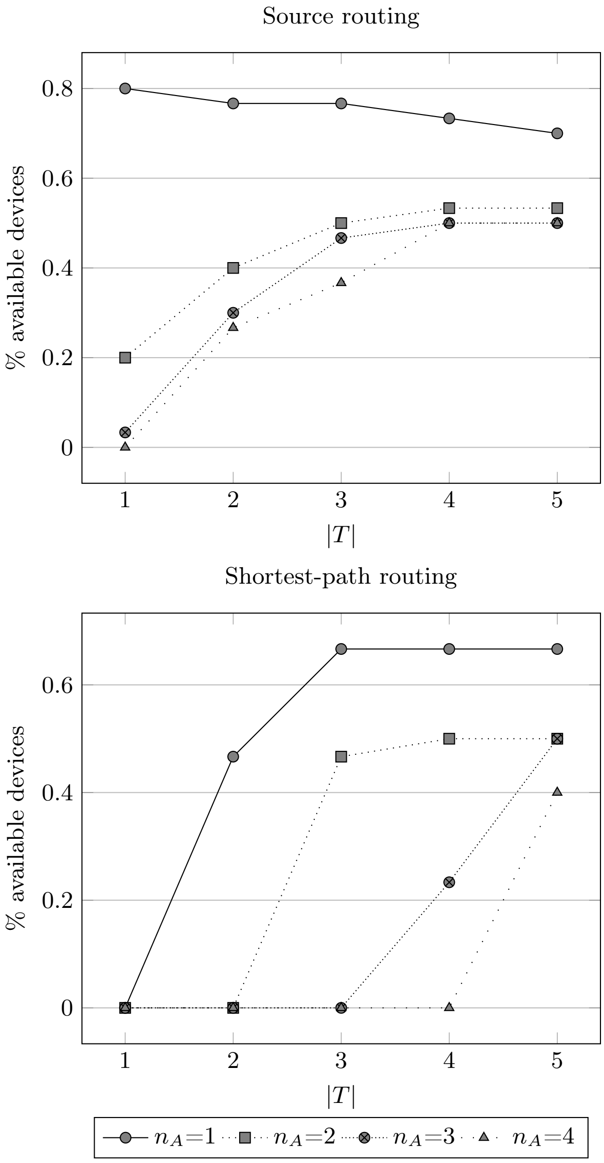

Fig. 8. Percentage of devices available for secure ToBKES in IEEE 30. Compromised devices were chosen to be at the worst-case position.

to the previous section, we assume that the compromised devices are located at the worst place, with respect to the device location.

We evaluated the number of devices in the network that will be able to receive a refreshed key for a number of compromised devices from 1 to 5, $n_A \in \{1, 2, 3, 4, 5\}$.

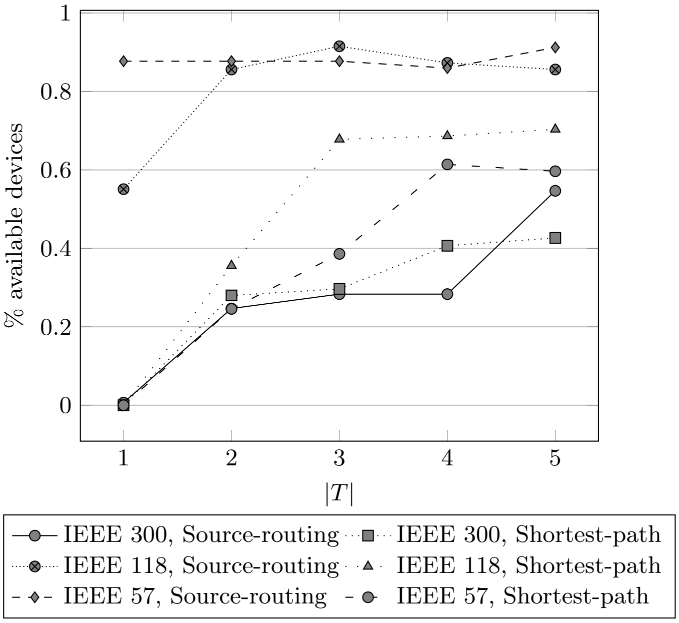

Fig. 9. Comparison between different network topologies showing devices available for secure ToBKES in the presence of a compromised device, $n_A = 1$.

The results are shown in Figure 10. From the results, it can be seen that even with partial deployment of CrypTop for only 30% of the network, more than 85% of the devices will be able to receive a refresh key.

Moreover, the results show that at 50% deployment of CrypTop, the proactive key refresh is available for more than 80% of the devices, even in the case where there are 5 compromised devices (and keys). This result strengthens CrypTop as a proactive method for plug-and-play key setup in ICS networks.

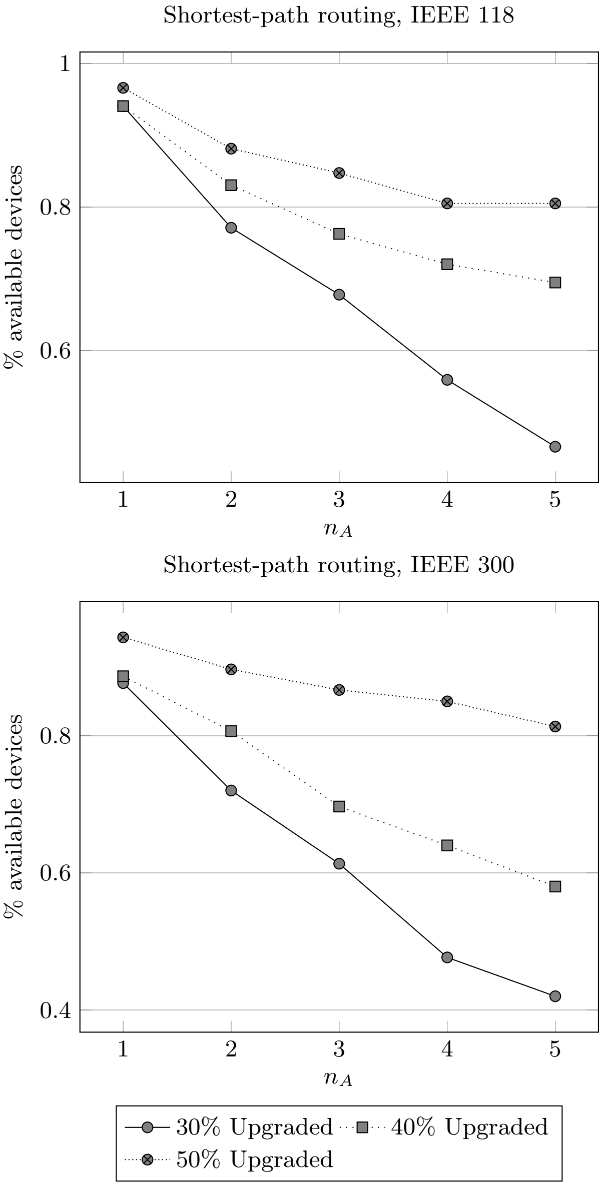

Fig. 10. Percentage of devices that can securely receive a refreshed key in the presence of n_A compromised devices. For large scale deployment of ToBKES, most of the network is secured against a small number of compromised devices.