Automated key setup and recovery from key exposure for power networks

Abstract—

Power networks are built with inherent redundancy, including in the communication channels. In this work, we show that this redundancy can increase the cyber security resiliency to key exposure. Specifically, we present CrypTop, a novel protocol for the *key setup and refresh* problem in power networks. CrypTop uses multipath communication and already-keyed devices, to setup keys in other devices, and to recover from key exposures.

We evaluate CrypTop in realistic power network topologies. Our results show a significant to dramatic improvement compared to previous works, in both recovery from key exposure and ability to setup keys automatically. For example, in IEEE 300 network, and for a single corruption, CrypTop can recover security in 100% of the devices compared to 1% for previous works.

I. Introduction

Power networks are usually distributed over large geographical areas, with devices placed at remote unmanned sites. Communication between the remote sites is prone to cyber threats, such as man-in-the-middle (MiTM), eavesdropping and data manipulation.

Cryptography is the primary tool used to protect communication against eavesdropping and MitM adversaries. Using cryptography means that a party must 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. Bellare et al. [4] present formal analysis of cryptogrpahic-based protocols. Those protocols typically rely on *initialization assumptions*, such as pre-shared keys, and on *computational assumptions*, such as poly-time attackers.

Using cryptographic protocols requires that each device will be manually initialized with keys, such as shared keys with specific peers, public keys of peers and a private key (for itself). Those initial keys are initialized in a secure environment (e.g., installed manually on each device, via secure channel [3] or in a 'secure zone' [1]), and may later be used to setup additional keys (e.g., use public keys to exchange shared keys).

In power networks, devices are often located at remote unmanned sites, which makes initialization a challenge. Furthermore, keys of devices in power networks are vulnerable to exposure by physical break-in, due to the remote location and to often insufficient physical security [16]. This is in addition to private-key exposure due to weak cryptography [16] and software vulnerabilities [1], problems which exist in other environments and applications too.

When key exposure is suspected, keys need to be re-initialized. Due to the large number of remote unmanned sites, and the geographically distance between sites, manual key recovery is difficult and undesirable. One alternative to manual key recovery is to distribute new keys using safely-stored private-public key pairs, however, this is inapplicable when even the private keys may be exposed (as due to physical break-in).

An alternative to manual key setup and recovery, first proposed in the seminal work of Dolev et al. [8], is to secure communication using knowledge of the *network topology*; most significantly, this eliminates the need for initialization of pre-shared keys. This approach relies on assumptions regarding the network topology; namely, the topology is known and is sufficiently redundant to prevent an attacker from controlling 'too many' links or nodes. Under such topology conditions, a pair of devices can set up a shared secret key, without assuming any pre-shared keys.

Protocols for sharing a secret key by using the topology, require a large number of disjoint paths that are not under the control of an attacker [8]. In addition, they require that communicating devices will control the route of the messages between them. However, controlling the message route is impractical in real networks, which are mostly based on shortest-path routing. Moreover, in shortest-path networks there is usually only one route between every two device, which is not enough for a secure secret sharing.

In this work, we present CrypTop, a protocol which combines cryptographic assumptions, *together* with topological assumptions. This protocol aim to solve the problem of *'plug-and-play' key setup and refresh*. The main goals of this problem are (1) to ease deployment of cryptographic security using *automated key establishment* and (2) to *recover from key-exposure* through periodic, automated, key refresh. We focus on the variant of the problem with a trusted, non-compromised *authentication server* device, whose public key is known; this assumption can be weakened using known results [6].

Previously suggested protocols, require disjoint routes merely between the client and the server. In contrast, CrypTop execution is based on disjoint routes between a client device and other devices that already have registered keys; we refer to these devices as *helpers* (Figure 1). The client sends a challenge to each helper, and collects the helper's signed response. The authentication server registers the client's key, only if the client provides enough authentication responses that came from disjoint paths. In that way, CrypTop is available to more network topologies than previous protocols.

We present CrypTop for both source-routing networks and shortest-path networks. For shortest-path routing, we present a novel approach to increase the number of shortest-path routes between helpers and client. We do this by allowing the helpers to send packets to their neighbors, even if this neighbor is not

on the shortest path between the helper and the client.

We evaluate the usage of CrypTop in power network. Those networks have a large number of routers, distributed over a large area, with low physical security. The concern of key exposure, caused by unauthorized physical access to a remote site, is a major concern. At the same time, manual recovery of the key requires significant efforts, e.g. reaching the remote sites. Our evaluation shows that using CrypTop, it is possible to set keys remotely and to proactively recover keys of more than 80% of the devices. In addition, the evaluation shows a significant improvement compared to previously suggested protocols.

CONTRIBUTIONS

We make the following contributions:

- We present CrypTop, a plug-and-play key setup and refresh protocol, which allows the setup of cryptographic keys (Section IV).
- We extend the model of Bellare et al. [4] to support the network topology and different routing models, including *source* and *shortest path* routing (Section A-E).
- We prove that CrypTop satisfy the security requirements for secrecy and for the correctness of the key (Section A-E).
- We evaluate CrypTop for power networks, and show its applicability to key setup and recovery (Section VI).

II. RELATED WORKS

A formal cryptographic model for message-driven key-exchange protocols was presented in the seminal paper of Bellare et al. [4]. The secure key-exchange protocols in their work, as in most of the works on key-exchange protocols, require that the parties pre-share some secret (e.g., message-authentication-code (MAC) key, signature key, or a verification key of a third trusted party). This pre-shared secret is assumed to be installed manually at each party or delivered via secure channel.

Our work extends the model developed by Bellare et al. [4] to account for the corruption of the pre-shared key. We assume that the key shared by the parties was exposed, and all of its bits are known to the adversary. To formally support that, we add a definition of *exposed* devices. These are devices that are not under the control of the attacker, but the attacker knows (and even set) their internal state.

Instead of using a pre-shared key, we use multi-path disjoint routes in the network to authenticate parties. This allows our model to provide keys without manual procedures in cases of key setup and key exposure. To formulate this benefit, we extend the execution model of Bellare et al. [4] to consider the topology of the network and the method by which messages are routed. This allowed us to formally analyze protocols that use cryptography *and* topology as part of their execution.

Using multi-path disjoint routes for secure transmission of message was first presented by Dolev et al. [20]. Their work deal with sending a secure message between parties, in case a man-in-the-middle attacker is presence in the network. Their work prove a fundamental requirement: in order to share a secret, the parties should send the message through $n_A + 1$ disjoint routes, where n_A is the number of routes that under the attacker control.

Following Dolev et al. [20] there was a wide range of research on using disjoint multi-path for sharing a cryptographic key [9], [10], [20], [22]. Recent work of Costea et al. [7] presents a Secure Multipath Key Exchange (SMKEX) protocol which allows key exchange between parties based on diffie-helman, on disjoint routes. Comparing to Costea et al. [7], our work ensures that the key exchange session is done between two *trusted* parties. This allows our work to be applicable for the case of *recovery from key exposure*; an application which non of the previous works support.

In similar to our work, several previous works suggest using multi-path disjoint routes for authenticating devices. This authentication method was evaluated for wireless networks [14] and for the internet routing [7], [21]. However, our work is the first done on using disjoint routes for authenticating devices in power networks.

In addition, previous works [7], [14], [21] achieved disjoint routes by sending information using different types of networks (e.g. cellular and wifi) or taking advantage of the large and dynamic connectivity of the Internet. However, in power networks, the parties use a single type of communication network and usually a non dynamic communication routes. Those properties of the network degrade the applicability of methods suggested by previous works [2], [7], [9], [10], [14], [20]–[22].

For enabling multi-path routes in power networks, our work presents a protocol that takes advantage of devices which already have a registered key, called *helpers*. Previous works [2], [7], [9], [10], [14], [20]–[22] uses multi-path routes between a client and an authentication server. In contrast, our work uses multi-path routes between the client and the helpers. This increases resiliency and applicability allowing CrypTop to operate in the presence of a larger number of compromised devices, and in power networks.

III. MODEL

In this section we present a model which formulates the terms we use, including the network routing and the adversary capabilities.

A. Network Model

We model the communication network as an undirected graph, $G = (V, E)$, where V represents the devices (nodes) and E is a set of edges, representing point-to-point communication between devices.

In this work, we focus on IP [19] networks and the router devices. We model every device $v \in V$ in the network as a router. Every router has an address v that uniquely represents it (e.g. its management IP address), and it can send (initiate) a message and it can forward messages initiated by other device. The algorithm that devices use to forward messages initiated by other devices, is called the *routing method*. For the routing method, we explore both source-routing and shortest-path. In source routing [12], each message contains the list of the devices on the route. In shortest-path [17], each message contains

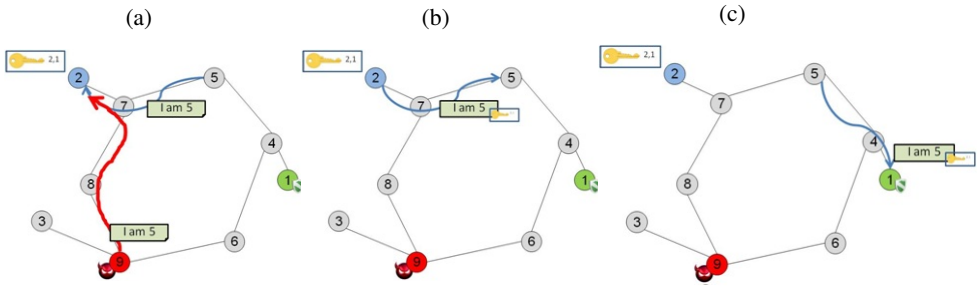


Figure 1: CrypTop Example: (a) Compromised device 9 tries to register key as device 5. CrypTop requires to receive a signed response from a *helper*, in this example, device 2. Device 2 signs on the challenges from device 9 and device 5 and sends them back. (b) Because of the network routing, both signed responses will be routed only to the *real* device 5. (c) Only the real device 5 will be able to send helper's signed response to the authentication server, located at device 1.

the destination device, and the router needs to calculate the next hop device.

Devices in the network must forward messages according to routing method. There are two exceptions to this rule: the first, are devices that initiate a message. Those devices allowed to pass their message to a neighbor device, not according to the routing method. The second exception are compromised devices, as will discuss in the next subsection.

In addition to initiating and routing messages, devices contain a computationally-efficient unit for processing messages and a clock that is proceeding at the same rate. Devices also contains a cryptographic signature scheme [13] module; the same scheme for all of the devices. We denote by $\mathcal{S}$ the signature scheme used by the devices.

For completeness, we provide informal definition of signature scheme, based on Katz et al. [13] definition. We refer the reader to their work for the formal definition. A Signature scheme $\mathcal{S}$ is a tuple of probabilistic polynomial-time algorithms $\mathcal{S}$ = (*KeyGen,Sign,Verify*), where:

- The key generation algorithm *KeyGen* receives as input a security parameter 1^l, and returns a pair of signing and verification keys, (σ, ν).
- The signing algorithm *Sign* receives as input a message m and a signing key, σ, and returns the signature of that message, $Sign_\sigma(m)$.
- The verification algorithm *Verify* receives as input a public verification key, ν, a message m, and a candidate signature sig. It returns *Accepted* if the signature of msg is sig, and returns *Rejected* otherwise.

B. Adversary Model

We consider an attacker $\mathcal{A}$ that can control devices. When controlling a device, the attacker receives all messages sent to this device, whether it is the final destination or still needs to be forwarded to its destination. The attacker determines which messages the device will send to its neighbors. Namely, it can delay, block and manipulate messages that a controlled device received for forwarding.

The attacker is also allowed to forward messages not according to the routing method. However, the attacker is not allowed to change the routing method of devices that are not under its control.

In addition, when controlling a device, the attacker has access to its storage, including its secret keys. The exposed information remains known to the attacker, even after it stops controlling the device. The main goal of our protocol is to recover those exposed devices, by setting new secret keys, which are unknown to the attacker.

Formally, the devices in the network are divided into three groups, with respect to the level controlled by the adversary:

- *Honest* - devices that are not under the control of the attacker and their internal state (e.g., private keys) is not known to the attacker.
- *Exposed* - devices that are not under the control of the attacker, but their internal state is known to the attacker, including private keys. We will show that under several conditions, exposed devices become honest (namely, can recover), during a single execution of the protocol.
- *Compromised* - devices that are under the control of the attacker, as described earlier.

Note, the groups *Honest, Exposed*, and *Compromised* are disjoint separations of V. We denote the number of exposed and compromised devices by n_A.

The attacker may have an arbitrary, 'byzantine' strategy. The main threat we consider is an attacker that tries to authenticate a compromised device as an honest device. In other words, an attacker that tries to 'spoof' the identity of an honest device.

IV. CrypTop Protocol

In this section, we introduce CrypTop, a key setup and refresh protocol. CrypTop protocol is a message-driven-protocol [4] involving an authentication server s, a client device and helpers, as we explain in the next sections.

A. Keys registration and Helpers

CrypTop protocol aim to register a signature-verification keys. We focus on registering signature-verification keys, since these can be used to establish keys for encryption and other goals, using known key-exchange protocols (e.g., [4]).

In typical use, CrypTop protocol would be invoked repeatedly, periodically, or on demand. However, for simplicity, our definitions and analyses are for a *single execution* of the protocol, from its activation until its termination.

At the beginning of the protocol execution, some devices may already have a registered key; due to manual installation or due to a previous CrypTop executions. we denote by $c.\sigma_{old}, c.\nu_{old}, c^s.\sigma_{old}$ the keys that were already registered at the beginning of the protocol execution. Those keys will need to be refreshed during the protocol execution.

We define a group of devices, called *helpers*, $\mathcal{H} \subset V$ as devices whose public keys were already registered by the server (or known otherwise, e.g., manually):

$\mathcal{H} = \{h \in V \ s.t. \ h^s.\nu_{old} \neq \perp\}$.

At the evaluation section (Section VI), we show that the size of the helpers group can be very small (about 5 helpers) in order to set keys for most of the devices in the network.

Upon activation of the protocol, client devices generate new signing and verification keys. By using CrypTop, clients request to register the new keys they generated. We denote by $c.\sigma, c.\nu$ the signing and verification keys that device c generated at the beginning of the execution.

We denote the signing key that the server registers at the end of the execution by $c^s.\sigma$. The execution may terminate without succeeding to register a new key, e.g., due to insufficient connectivity; in case of such failure, we set $c^s.\sigma$ to the special value $\perp$. Details of the cases where the registration fails are described at Section A-E.

B. Initialization.

The *server* s is initialized with a security parameter 1^l, the network topology $G = (V, E)$, the verification keys of the helpers ,$\{h.\nu_{old} \mid h \in \mathcal{H}\}$, and a pair of public-private keys for signatures $s.\nu, s.\sigma$. As in other works dealing with multiple compromised systems [6], we assume a known bound n_A on the total number of possible compromised and exposed devices.

Every client $c \in V$ is initialized with the server's public verification key $s.\nu$ and a security parameter 1^l. Both the server and the clients are using the same signature scheme $\mathcal{S}$.

C. Protocol Messages

A successful session between a single client c and the server s can be seen in Algorithm 1. All the messages in the protocol are signed and includes timestamp. Parties discard messages with an invalid timestamp or signature. The protocol has three phases of operation:

a) ACTIVATION*:* To initiate the protocol, the server sends a *START* message to a client c. Upon receiving the START message, the client generates a (new) random pair of private and public keys $(c.\nu, c.\sigma) \leftarrow KeyGen(1^l)$. The client then sends the new public key $c.\nu$, signed with its already-registered key $c.\sigma_{old}$ (Message 3, at Algorithm 1).

In key setup and recovery, the client c does not have a previous registered key. In those cases, $c.\nu_{old} = \perp$ and the client sends merely $c.\sigma$, without signing it.

Notations:

$c.\nu, c.\sigma$	The verification and signing keys that client c requested to register.
$c.\nu_{old}, c.\sigma_{old}$	The verification and signing *preset keys* of device c.
$[m]_{c.\sigma}$	The message m and its signature, using key $c.\sigma$.
$h \overset{\mathfrak{R}}{\rightsquigarrow} c$	h sends a message to c through the route $\mathfrak{R}$.

Protocol Messages:

		ACTIVATION
[1]	$s \rightsquigarrow c$	$[START, timestamp, c,]_{s.\sigma}$
[2]	c	$(c.\sigma, c.\nu) \leftarrow KeyGen(1^l)$
[3]	$c \rightsquigarrow s$	$[timestamp, c, c.\nu]_{c.\sigma_{old}}$
		COLLECTING SIGNED RESPONSES
[4]	s:	Chooses a group of helpers $\mathcal{H}_c$, with $n_A + 1$ vertex-disjoint routes from c to $h \in \mathcal{H}_c$. Terminates if no such group of helpers.
[5]	For each helper $h \in \mathcal{H}_c$:	
[5.1]	$s \rightsquigarrow c$	$[timestamp, h, \ \mathfrak{R}_h \ , c.\nu, h.\nu_{old}]_{s.\sigma}$
[5.2]	$c \rightsquigarrow h$	$[timestamp, h, \ \mathfrak{R}_h \ , c.\nu, h.\nu_{old}]_{s.\sigma}$
[5.3]	$h \overset{\mathfrak{R}_h}{\rightsquigarrow} c$	$[timestamp, c, h, \mathfrak{R}_h, c.\nu]_{h.\sigma_{old}}$
[5.4]	$c \rightsquigarrow s$	$[timestamp, c, h, \mathfrak{R}_h, c.\nu]_{h.\sigma_{old}}$
		VALIDATION
[6]	s	If there are $n_A + 1$ authenticated responses:
[6.1]	$s \rightsquigarrow c$	$[timestamp, ACK, c, c.\nu]_{s.\sigma}$
[6.2]	c	Outputs $c.\nu$.
[6.3]	$c \overset{\mathfrak{R}}{\rightsquigarrow} s$	$[timestamp, FINAL]_{c.\sigma}$
[6.4]	c	Outputs $c.\nu$
[6.5]	s	Outputs $c^s.\nu$

Algorithm 1: CrypTop session with a client c

It is important to note that the last message we described does not ensure that indeed device c sent it (even if the message is signed). The reason is that device c may be exposed. In that case, attacker can use device c old key, to signed on a new key generated by the attacker. Therefore, the server must authenticate that indeed device c sent the message. This authentication is done by using the helpers, in the second phase of the protocol.

b) COLLECTING SIGNED RESPONSES*:* In order to authenticate device c (e.g. the attacker did not spoofed device c identity), the server s sends to c challenges. Those challenges must be sent to and signed by a subgroup of helpers, $\mathcal{H}_c \subset \mathcal{H}$. The mandatory requirement from $\mathcal{H}_c$ is that there are $n_A + 1$ disjoint routes between the helpers and the client c. If there is not such a group of helpers, namely, the network topology does not provide enough redundancy, the server terminates the protocol, and does not register any key, i.e., sets $c^s.\sigma$ to the special value $\perp$.

Every challenge is built from the addresses of the chosen helper, $h \in \mathcal{H}_c$, the helpers' verification key $h.\nu_{old}$ and the route the helper should use in order to send its signed response to the client c.

The server is the only device that knows the network topology. Therefore, for ensuring disjoint routes, the server must embed in the challenges the route by which the helper

h should use for sending back the signed response. Formally, for each helper $h \in \mathcal{H}_c$, the server defines the route $\mathfrak{R}_h$ as the list of devices, from the helper h to the client c, that the helper should use.

For every helper $h \in \mathcal{H}_c$, the client sends the challenge it received from the server, to the helper h. Every helper h, upon receiving a challenge, verifies that the server signed on it. If the verification passes, the helper responds with signing on the challenge, using its already-registered signing key, $h.\sigma_{old}$. The helper routes the signed response according to the route $\mathfrak{R}_h$ it received in the challenge message.

Note that in **source routing**, the helper is able to send the response through the exact route defined in $\mathfrak{R}_h$. However, in the common case of **shortest-path** routing, the route cannot be chosen by the helper. We explain how CrypTop overcome this limitation in the next subsection.

When the client receives a signed response from the helper, it verifies the response signature and forwards it to the server.

c) VALIDATION *:* If the server receives n_A+1 correct and properly-signed responses, then it sends a signed *ACK message* to the client. At section V-C we prove that if there are n_A+1 such responses, than the client and the server registered the same key.

In some cases, there may not be enough signed responses - whether it is due to the topology that does not allow enough redundant disjoint paths, or due to an adversary that block some responses. In that case, the server will not register a key for the device, and if there was a key, the server will un-register it (setting $c.\sigma = \perp$).

This ACK message contains the client address and the public key received from the client, signed by the server $[c, c^s.\nu]_{s.\sigma}$. When the client receives this message, it responds by sending a *FINAL message*, signed by its private key to the server. In addition, the client outputs its new registered key $c.\nu$, $c.\nu \neq \perp$. The server validates the signature, registers the key $c^s.\nu$ for c, and outputs $c^s.\nu \neq \perp$. If the client does not receive an *ACK message* within T_{max} time, it will output $c.\nu = \perp$. If the server does not receive a *FINAL message* within T_{max} time, it will output $c^s.\nu = \perp$.

When all goes well, the key $c.\nu$ is secret (e.g. an attacker can no use it to sign on messages) and it is equal to the key registered by the server $c^s.\nu$. In the next section, we formally define those properties and prove that upon validation, CrypTop ensures them.

D. Key Recovery

Let c be an exposed device. By definition, the secret key $c.\sigma_{old} \neq bot$ is known to the attacker. While device c is exposed, the attacker is able to imitate (spoof) device c messages and to read its secret encrypted information. However, as we will explain, after one CrypTop session, the attacker is no longer knows the device secret key, and device c changed its state from exposed to hones. We define this process as *key recovery*, and it includes two phases: at first, the client needs to register a new key. Second, the new key must be secret from an attacker who knows the old key.

At the first phase of the recovery, device c must generate a new secret key and to register that new key $c.\sigma$. Following CrypTop session, the key will be register only if c provides n_A+1 correct and properly-signed responses. Second, we need to prove that the new key is secret and unknown to the attacker. Intuitively, the new key is secret since it was sent encrypted by the server public encryption key. Messages encrypted with the server public key can be decrypted only by the authentication server. Formally, we will prove this secrecy property at the next section.

E. CrypTop for shortest-path routing

In shortest-path routing, each device chooses to pass the message to its neighbor which is located between the device and the recipient of the message. Therefore, without innervation in the routing method, between every two devices in the network there is only one possible route. This single-path creates a challenge for multi-path protocols in shortest-path networks.

To overcome this challenge, the helpers in CrypTop are sending their signed responses through their neighbor that was defined in the route $\mathfrak{R}_h$; this, even if the neighbor is not on the shortest-path between the helper and the client. Note that this is not violating the routing-method, since the helpers are the initiators of the signed response message. The helper's neighbor that received the signed response must forward the message, through the shortest-path to the client device.

Note that this method to enrich the number of disjoint routes may not always increase the number of those routes. Some of the helpers neighbors may return the message to the sender helper, for example if the shortest path between the neighbor and the client goes through the helper. Surprisingly, as we will show in the evaluation section VI, in many cases, the path $v \rightsquigarrow c$ does not pass through h.

V. PROTOCOL ANALYSIS

In this section we analyze the security and correctness of CrypTop. Those properties depends on the security of the signature scheme $\mathcal{S}$. We denote by $\pi_{\mathcal{S}}$ a CrypTop execution were the devices are using $\mathcal{S}$ as their signature scheme.

SEC$(\{c.\nu, c^s.\nu \mid c \in V\}, Compromised \subset V)$
// The test fails if an attacker with access to signing oracle $\mathcal{O}_{\mathcal{S}}$, was able to create a new pair of message msg_A and signature sig_A for device c, s.t. $\mathcal{S}.Verify(c.\nu, msg_A, sig_A)$ =$Accepted$.
$msg_A, sig_A, c \leftarrow \mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$
if $c \notin Compromised$ **then**
  if $\mathcal{S}.Verify(c.\nu, msg_A, sig_A)$ =Accepted ***AND*** msg_A *was never given to* $\mathcal{S}.Sign$ **then**
    Return FAIL
  end
end
Return SUCCESS

Algorithm 2: Secure signing test

A. Protocol Execution

To facilitate provable security, we adopted a precise execution model. The protocol execution we present is an extension to Bellare et al. [4] model for message-driven-protocols. Due to page limitations, we describe informally mainly our extension

COR($\{c.\nu, c^s.\nu \mid c \in V\}, G = (V, E), s, \mathcal{H}, Compromised \subset V$)

// The test Fails if there exists device c, with $n_A + 1$ disjoint routes from the helpers, and its keys that were registered in the server are not the identical to the one the device c registered.

foreach $c \in V - \{s\}$ *s.t.* $P^{n_A+1}(\mathcal{H}, c) = 1$ ***AND*** $c \notin Compromised$ **do**
  if $c^s.\nu \notin \{\perp, c.\nu\}$ **then**
    Return FAIL
  end
end
Return SUCCESS

Algorithm 3: Correctness test

to Bellare et at. [4] model. Further details and extended formal analysis can be found at Appendix A.

The execution is a process $\mathbf{EXEC}^{TEST}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)$ that receives as input: a probabilistic polynomial time (PPT) algorithm as the adversary $\mathcal{A}$; CrypTop protocol that uses the signature scheme $\mathcal{S}$, $\pi_{\mathcal{S}}$; and a security parameter 1^l. At the beginning of the execution, the adversary $\mathcal{A}$ chooses the network topology $G = (V, E)$, an honest device as an authentication server s and up to n_A devices as compromised or exposed.

Devices in the execution are initialized in the following way: internal state of compromised devices is initialized by the adversary; exposed and honest devices are initialized by the protocol. Exposed devices provide their private keys to the adversary.

After initialization, devices are activated by the adversary. Upon activation, the device generates a message and adds it to a shared message pool. During the execution, the adversary is able to choose which devices to activate, and which message to send and on which order.

In contrast to Bellare et al. [4] model, we limit the adversary to modify and observe only messages that pass through its devices. For messages that do not pass through its device, the adversary knows only the sender device identity and the desired destination device. This limitation extends Bellare et al. [4] model to compromise also the network topology and the routing method.

During the protocol execution, the server and clients output the key they registered (as explained at Section IV). The *global output* (as defined at [4]) of running the execution is the concatenation of the cumulative outputs of all the devices, together with the output of the adversary. In our case, the global output is $\{c.\nu, c^s.\nu \mid c \in V\}$.

In order to formally define and to prove the properties of the protocol, we use *test algorithms*. Each test, receives as an input the global output of the execution. Test algorithms may receive additional inputs, according to the property they check. The test algorithms returns FAIL or SUCCESS.

We denote by $\mathbf{EXEC}^{TEST}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)$ a protocol execution were the execution's global output is given as an input to the algorithm TEST. For simplicity of notation, we omit the TEST's inputs from the notation.

We next describe the properties of $CrypTop$, by presenting the corresponding test algorithms. We present at this paper two fundamental properties: secure signing and correctness. At Appendix B, we present additional properties: Bounded Termination, Guaranteed key setup and Correctness resiliency to Man-in-the-Middle.

B. Secure Signing

A fundamental requirement from a key setup protocol is to preserve the secrecy of the registered keys. The formal test for this property can be seen in Algorithm 2. We now formally define this property and prove that CrypTop ensures secure signing if $\mathcal{S}$ is a chosen-message-attack secure (CMA-Secure) signature scheme [13].

Theorem 1. ***[Secure Signing]****: For every CMA-secure signature scheme [13]* $\mathcal{S}$ = (KeyGen,Sign,Verify) *and for all PPT attackers* $\mathcal{A}$, $\gamma > 0$, $\exists l_0$ *s.t.* $\forall l > l_0$*:*

$$Pr(\textbf{\textit{EXEC}}^{SEC}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)\text{=FAIL}) \; < \; l^{-\gamma},$$

where the probability is taken over the random coins used by $\mathcal{A}$ *and* $\textbf{\textit{EXEC}}^{SEC}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)$*.*

Proof:

The proof is based on the well known chosen-message-attack secure signature game; due to page limitation, we refer the reader to [13] for details on this game.

Let $\mathcal{S}$ be a CMA-secure signature scheme [13]. Assume to the contrary that exists a PPT attacker $\mathcal{A}_\pi$ on CrypTop and probability $\epsilon > 0$ s.t.:

$$Pr(\mathbf{EXEC}^{SEC}(\mathcal{A}_\pi, \pi_{\mathcal{S}}, 1^l) \text{=FAIL}) \geq \epsilon.$$

At the beginning of the CMA-secure game, the game randomly chooses a pair of private and public keys. Let PR, PK be the private and public keys generated by the CMA-secure experiment [13], and let $\mathcal{O}_{\mathcal{S}}$ be a signing oracle, which receives as input a message and returns the signature on it. We will build an attacker $\mathcal{A}^{\mathcal{O}_S}(PK)$ for $\mathcal{S}$ in the following way.

For any given public key PK, $\mathcal{A}^{\mathcal{O}_S}$ executes $\mathbf{EXEC}^{SEC}(\mathcal{A}_\pi, \pi_{\mathcal{S}}, 1^l)$. At the beginning of the execution, $\mathcal{A}^{\mathcal{O}_S}$ randomly chooses an honest device $c*$; Whenever $c*$ calls $KeyGen$, $\mathcal{A}^{\mathcal{O}_S}$ will return PK as the public key, and $\perp$ as the private key. Whenever device c calls $Sign(PK, msg)$, $\mathcal{A}^{\mathcal{O}_S}$ will use the oracle $\mathcal{O}_{\mathcal{S}}$ to sign the message msg.

Let msg_A and $sign_A$ be the output of $\mathcal{A}_\pi$ that failed SEC test. These pair of message and signature can be a pair for $c*$ or for other devices in the network. In both cases, $\mathcal{A}^{\mathcal{O}_S}$ will return msg_A and $sign_A$ to the CMA-secure game.

The CMA-secure game uses the private key PR to verify whether $sign_A$ is the signature of a message that has not seen before by the oracle, msg_A. This verification will succeed (and the attacker will 'win' the game) only if $\mathcal{A}_\pi$ provided a pair for device c^*.

We now calculate the probability that the pair msg_A and $sign_A$ are for device $c*$. Since the keys in $EXEC^{SEC}$ were chosen randomly, the distribution for SEC test to fail does not change; it is ϵ. Since the device $c*$ is chosen randomly from all of the devices, the probability of SEC test failing with device $c*$ is $\frac{\epsilon}{|V|}$.

Following the above discussion, SEC test fails because of a pair relevant to device $c*$ with a probability of $\frac{\epsilon}{|V|}$. Therefore,

the CMA-secure game will fails with the probability of $\frac{\epsilon}{|V|}$. And this is a contradiction to the assumption the $\mathcal{S}$ is a CMA-secure signature scheme. Therefore, the claim holds. ■

C. Correctness

In the correctness property, we require that if the server registered a key ($c^s.\nu \neq \perp$), then the client registered the exact same key ($c^s.\nu = c.\nu$).

This property of the protocol depends on the client's location within the network topology. To formulate the topological dependency, we define a *topology predicate* $P^{n_A+1}(\mathcal{H}, c)$ that returns 1 if there are $n_A + 1$ disjoint routes from the group of device $\mathcal{H}$ to c. The formal test algorithm for correctness is described in Algorithm 3).

Theorem 2. ***[Correctness]****: For every CMA-secure signature scheme [13] $\mathcal{S}$ and for all PPT attackers $\mathcal{A}$, $\gamma > 0$, $\exists l_0$ $s.t.$ $\forall l > l_0$:*

$$Pr(\textbf{\textit{EXEC}}^{COR}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)\text{=FAIL}) \; < \; l^{-\gamma},$$

where the probability is taken over the random coins used by $\mathcal{A}$ and $\textbf{\textit{EXEC}}^{COR}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)$.

Proof:

Assume to the contrary that there exists a PPT attacker $\mathcal{A}_\pi$ on CrypTop with probability $\epsilon > 0$ s.t.:

$$Pr(\textbf{EXEC}^{COR}(\mathcal{A}_\pi, \pi_{\mathcal{S}}, 1^l) \text{=FAIL}) \geq \epsilon.$$

COR fails if there is a non-compromised device c, s.t. $c^S.\nu \neq \perp$ and $c.\nu \neq c^S.\nu$.

If $c^S.\nu \neq \perp$, then the server registered a key for c. According to the protocol design, the registration is done when there are at least $n_A + 1$ disjoint routes between the group of helpers and the client c. In such topology, it is clear that an attacker with n_A devices will not be able to receive all the responses. Therefore, the attacker will need to do one of the following: (1) to change the routes sent by the server in the START message; or (2) to create a 'fake' response from an honest helper. Formally, we now show that if the attacker $\mathcal{A}_\pi$ was able to perform one of the above actions, then $\mathcal{A}_\pi$ contradicts the CMA-secure assumption of $\mathcal{S}$.

Let PK be the public key generated by the CMA-secure experiment [13] and let $\mathcal{O}_{\mathcal{S}}$ be a signing oracle, which receives as input a message and returns the signature on it. We will build an attacker $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}(PK)$ for $\mathcal{S}$ in the following way.

For any given public key PK, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ executes $\textbf{EXEC}^{COR}(\mathcal{A}_\pi, \pi_{\mathcal{S}}, 1^l)$ twice. The first time, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ sets the public key of the server to be PK, and whenever s calls for $Sign$, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ will call the oracle to sign the message. In the second execution, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ chooses at random an honest upgraded device $c*$. Whenever $c*$ calls $KeyGen$, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ will return PK as the public key, and $\perp$ as the private. Whenever $c*$ calls $Sign$, $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ will use the oracle to sign the message.

We did not change the execution process and the distribution for FAIL did not change. If $\mathcal{A}_\pi$ changes the START message, the probability of $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ winning the CMA-secure experiment is ϵ. If $\mathcal{A}_\pi$ creates a new signed response of an honest helper, the probability of $\mathcal{A}^{\mathcal{O}_{\mathcal{S}}}$ winning the CMA-secure experiment is $\frac{\epsilon}{|V|}$. In both cases, we receive a non negligible probability for failing the CMA-secure game, and therefore, a contradiction to the assumption that $\mathcal{S}$ is CMA-secure. Therefore, the theorem holds. ■

VI. Experimental Evaluation

In this section we evaluate the applicability of using CrypTop in power networks. For the evaluation, we use power networks architectures from different sizes: IEEE 300, IEEE 118, IEEE 57 and IEEE 30 bus systems [18]. The topology properties of the networks are presented at Table I.

	Devices	Edges	% Devices with number of neighbors			
			> 1	> 2	> 3	> 4
IEEE 300 [18]	300	409	77%	51%	23%	9%
IEEE 118 [18]	118	179	94%	46%	30%	17%
IEEE 56 [18]	57	78	98%	42%	21%	8%
IEEE 30 [18]	30	41	90%	40%	23%	10%

Table I: Evaluated Networks

A. Incremental Deployment

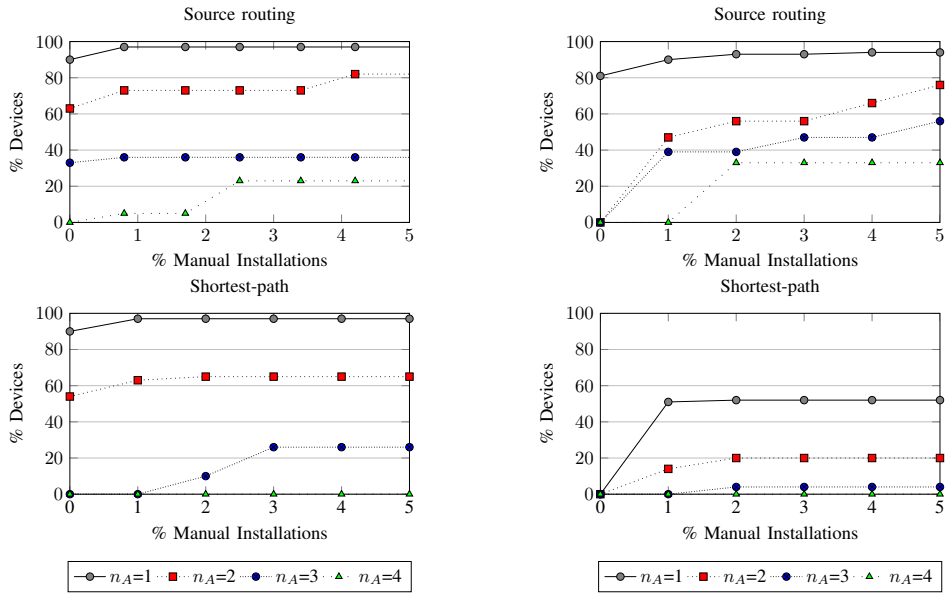
CrypTop key refresh and setup is based on using devices that already have keys, 'helpers'. Those helpers may need to be registered manually with a key. Using a group of initial helpers, the protocol 'bootstraps' to set keys to more devices, and to increase the number of helpers.

We simulated the bootstrapping process, measuring the number of devices that CrypTop can set them keys ('registered devices'), assuming few pre-set devices (manual installations). The simulation executed CrypTop sequentially. At the first execution of the protocol, we choose one device as the authentication server. At the end of the first execution, the simulation provides all the registered devices, that can receive key using the server only. At the next protocol execution, the simulation uses all the registered devices from the previous execution as helpers, and finds new registered devices. In that way, the simulation continues until no more devices can be added. Namely, the simulation reaches the maximal number of devices that can set their keys using CrypTop, in the presence of a single helper.

At this point, the simulation chooses an unregistered-device, adds it to the registered device group (namely, simulating manual key installation on this device), and performing another protocol execution. In that way, we simulate manual installation of devices, at cases where CrypTop can no longer increase the number of registered devices.

For choosing device for manual installation, we used the *highest degree* heuristic method. Namely, we first choose the server to be the device with the highest edge-degree in the network. Later, whenever the simulation chooses another device for manual installation, it would pick the unregistered device with the highest degree. At Appendix C we show that the highest-degree heuristic provides results which are very close to the optimal choice, in a more computational efficient way.

The number of registered devices depends on the topology, and it is limited by the number of devices with degree higher



(a) IEEE 118 (b) IEEE 300

Figure 2: Percentage of devices that securely receive their keys using CrypTop, for different amount of manual installations, for different number of corrupted devices n_A and routing methods. Networks IEEE 56 and IEEE 30 reached the maximal availability (presented at Figure 3) after 2 manual installations, and due to page limitation, we omit their graphs.

than n_A. To allow meaningful comparison between the results for different topologies and different n_A, we present the *percentage* of registered devices, as a fraction of the total number of devices with degree higher than n_A. The results for incremental deployment are shown at Figure 7.

The results show that with few manual installations - e.g., less than 4% of the network - the number of registered devices increases significantly. Those results emphasize the importance of using helpers, and that CrypTop is very effective even with only a small fraction of 'manual' key setups in the network.

B. Key Recovery

In this part of the evaluation we measured the maximal number of devices, that will be able to recover their keys using CrypTop. To evaluate that, we started the simulation at the state where *all* the devices with degree higher the n_A have a registered key (and therefore, helpers). For each device, the simulation assumed that the device the was compromised in the previous execution, and therefore, its key need to be recovered. Using the helpers, the protocol tries to recover the device key.

After the first protocol execution, some of the devices may not be able to recover their key, because they do not fulfill the topology requirements. Those devices will not be part of the helpers group at the following protocol execution. We continue the simulation to more executions, until the number of devices with registered keys does not decrease, i.e., stabilizes. We refer to the percentage of the devices that remain with fresh registered keys, in the stable state, as the *maximal availability* of the protocol to recover devices.

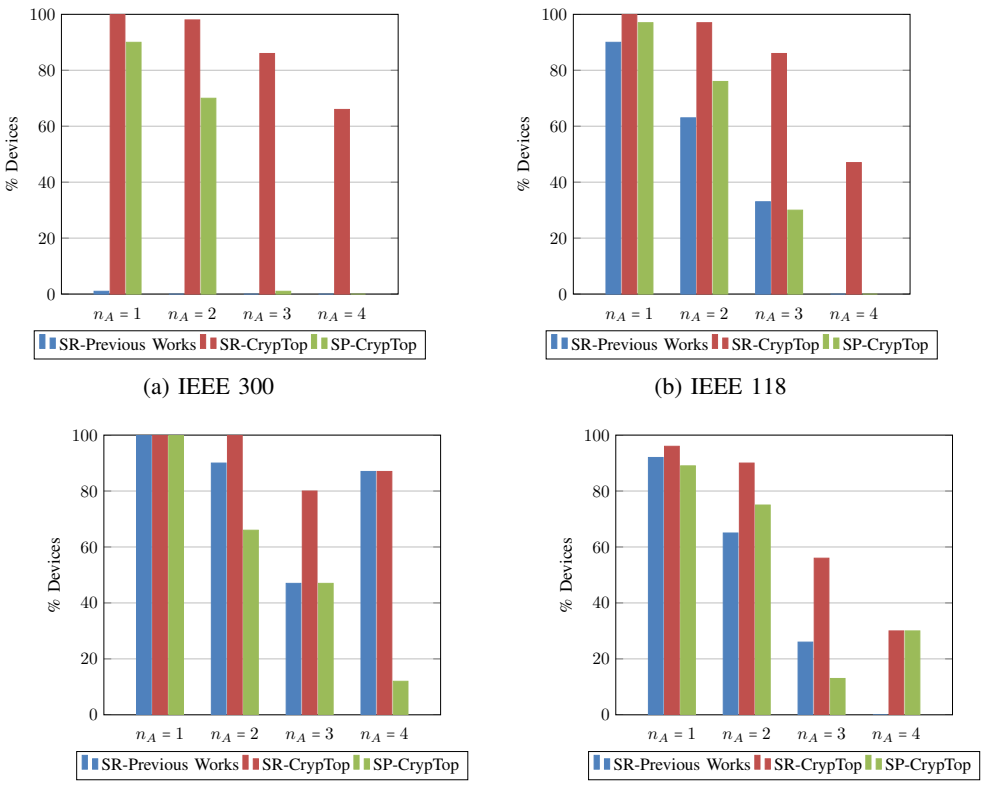
The results are shown in Figure 3. In source-routing, CrypTop is able to recover keys to significantly more devices than previous works, in most of the evaluated network. Those results show the significant impact of the helpers, in increasing the maximal number of registered devices.

Comparing between the routing methods, we find that in shortest-path routing, the number of devices that can recover their keys is, in most cases, significantly less than in source-routing. This is not surprising, as routes in shortest-path networks are more limited. However, the results show that, even with this (much more restrictive) routing mechanism, CrypTop achieves high fraction of registered devices, e.g., over 80% for $n_A = 1$. Notice that previous works does not support shortest-path routing at all.

VII. Conclusions and Future Work

This work introduced CrypTop, a multi-path protocol for key setup and recovery. The protocol uses both cryptography and topology for achieving secure and correct signing on crytpographic keys. For the protocol analysis, the work extends Bellare et al. [4] model with support of the topology and the routing of the network. This extension allows, in the future, to analyze further protocols that uses cryptography and topology.

For a correct and secure key setup and recovery, multi-path protocols require $n_A + 1$ disjoint routes between the communicating parties. In contrast, CrypTop requires $n_A + 1$ disjoint routes from a group of helpers to the client. This requirement allows CrypTop to operate in shortest-path networks, such as power networks. However, for CrypTop to operate the client must have at least n_A+1 neighbors (for having disjoint routes). This limitation can be solved by previously suggested methods, such as opportunistic tunnels [11]. We leave for future work evaluating such methods for CrypTop.



(a) IEEE 300

(b) IEEE 118

(c) IEEE 56

(d) IEEE 30

Figure 3: Comparison of the fraction of devices that can recover from key exposure, for different number of corrupted devices n_A and routing methods: source-routing (SR) and shortest-path (SP). Previous works do not support shortest-path routing at all.

Deploying a new protocol is challenging. The benefit the operator should have must be significant greater than the efforts. We evaluate CrypTop for power networks, and we show that we a small effort of less than 5 manual installations, more than 80% of the device can use CrypTop for key setup and recovery; this, even in shortest-path network, where no previous works support. We leave for future works exploring CrypTop for other networks (e.g. internet) and other routing methods.

References

[1] AGA report no. 12. cryptographic protection of SCADA communications: General recommendations, 2006. V.

[2] Paolo Barsocchi, Stefano Chessa, Ivan Martinovic, and Gabriele Oligeri. A cyber-physical approach to secret key generation in smart environments. *J. Ambient Intelligence and Humanized Computing*, 4(1):1–16, 2013.

[3] Cheryl Beaver, Donald Gallup, William Neumann, and Mark Torgerson. Key management for scada. *Cryptog. Information Sys. Security Dept., Sandia Nat. Labs, Tech. Rep. SAND2001-3252*, 2002.

[4] 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.

[5] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005.

[6] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.

[7] Sergiu Costea, Marios O Choudary, Doru Gucea, Björn Tackmann, and Costin Raiciu. Secure opportunistic multipath key exchange. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2077–2094. ACM, 2018.

[8] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.

[9] 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.

[10] Jokin Garay, Clint Givens, and Rafail Ostrovsky. Secure message transmission with small public discussion. *Information Theory, IEEE Transactions on*, 60(4):2373–2390, 2014.

[11] Yossi Gilad and Amir Herzberg. Lightweight opportunistic tunneling (lot). In *European Symposium on Research in Computer Security*, pages 104–119. Springer, 2009.

[12] Bob Hinden and Dr. Steve E. Deering. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[13] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[14] K Vinoth Kumar and S Bhavani. Trust based multipath authentication protocol for mobile ad-hoc network. *Journal of Computational and Theoretical Nanoscience*, 12(12):5479–5485, 2015.

[15] 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.

[16] Rasel Mahmud, Ranganath Vallakati, Anupam Mukherjee, Prakash Ranganathan, and Arash Nejadpak. A survey on smart grid metering infrastructures: Threats and solutions. In *2015 IEEE International*

Conference on Electro/Information Technology (EIT), pages 386–391. IEEE, 2015.

[17] John Moy. OSPF Version 2. RFC 2328, April 1998.

[18] University of Washington. Power systems test case archive.

[19] Jon Postel. Rfc 791: Internet protocol, september 1981. *Darpa Internet Protocol Specification*, 1990.

[20] K Srinathan, Arvind Narayanan, and C Pandu Rangan. Optimal perfectly secure message transmission. In *Advances in Cryptology–CRYPTO 2004*, pages 545–561. Springer, 2004.

[21] Dan Wendlandt, David G Andersen, and Adrian Perrig. Perspectives: Improving ssh-style host authentication with multi-path probing. In *USENIX Annual Technical Conference*, volume 8, pages 321–334, 2008.

[22] 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.

[23] 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.

APPENDIX A
THE CRYPTOPOLOGY MODEL

A. Network Model

We model the communication network as an undirected hypergraph, $G = (V, E)$, where V represents the 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.

Every device $v \in V$ in the network also acts as a router; it has an address v that uniquely represents it, and it can send and route messages.

B. Legacy Devices and the Routing Model

To support incremental deployment, we consider that devices V fall into two categories: *upgraded devices* that support the CrypTopology protocol and *legacy devices* that do not support it. Legacy devices are assumed to route only messages that pass through them, following the *routing model*.

We focus on two important routing models: *source* routing, where senders can choose arbitrary routes to each destination, and *shortest path* routing, where packets are always sent along the shortest route. Previous works generally considered source routing, while most IP networks use shortest-path routing.

We find it convenient to use common notation for both routing methods. Specifically, routing is formulated as a function $\mathfrak{R} : V \times \{0,1\}^* \to V$, which receives the *current* device and a value called *destination-route* carried in the message; it returns a neighbor of the current device, to which the message is forwarded. The destination-route value depends on the routing method; there can be multiple routing methods. We consider the following $\rho \in \{$ *source,shortest-path* $\}$:

- *Source* routing: the *destination-route* is the sequence of devices along the route, from source till destination. To prevent loops, a device cannot appear twice in the route. The routing function returns the next entry in the destination-route field, after the current device.
- *Shortest-path* routing: the *destination-route* only identifies the destination device. Messages are sent on the shortest-path between the current device and the destination device.

C. Adversary Model

We consider an attacker $\mathcal{A}$ that can control n_A devices. When controlling a device, the attacker receives all messages sent to this device, whether it is the final destination or still needs to be forwarded to its destination. The attacker determines which messages the device will send to its neighbors. Namely, it can delay, block, and manipulate messages that a controlled device received for forwarding. In addition, when controlling a device, the attacker has access to its storage, including its secret keys. The leaked information remains known to the attacker, even after it stops controlling the device.

The devices in the network are divided into three groups, with respect to the level controlled by the adversary:

- *Honest* - devices that are not under the control of the attacker and their internal state (e.g., private keys) is not known to the attacker.
- *Exposed* - upgraded devices that are not under the control of the attacker, but their internal state is known to the attacker, including private keys. We will show that under several conditions, exposed devices become honest (namely, can recover), during a single execution of the protocol.
- *Compromised* - devices that are under the control of the attacker.

Note, the groups *Honest, Exposed*, and *Compromised* are disjoint separations of V, and n_A is the total number of exposed and compromised devices.

The attacker may have an arbitrary, 'byzantine' strategy. The main threats we consider are (1) an attacker that disrupts the key setup process for some (upgraded) devices, and (2) an attacker that learns, or even sets, the key for some upgraded (non-compromised) devices.

D. Time and Synchronization

The goal of the CrypTopology protocol is to establish keys in newly-upgraded devices, and to refresh keys in devices that are already (upgraded and) keyed. The purpose of these keys is to recover, maintain, or improve security. In typical use, the protocol would be invoked repeatedly, periodically, or on demand. However, for simplicity, our definitions and analyses are for a *single execution* of the protocol, from its activation until its termination. Each single execution takes into account the initial state of the devices at the beginning of the single execution. All our definitions and analyses will be based on the relation between the initial and final state of the devices, in that single execution period.

We assume a bound of T_{delay} on the communication delay for each edge in the network, and ignore the processing time. We later demonstrate that this makes sure every execution completes by at most $T_{max} \leq 8|V|T_{delay}$.

For simplicity, we further assume that all devices have perfectly synchronized clocks; it appears straightforward but

is somewhat tricky to extend for bounded bias. Practically, the time between two executions of the protocol will be much larger than the bounded bias, and therefore, there is no significant limitation in the model assumption of perfectly synchronized clocks.

E. Protocol Execution

EXEC$^{TEST,\rho}(\mathcal{A},\pi,1^l)$:

```
// Attacker Choices:
{G = (V, E), φ : V → {Legacy, Upgraded}, s ∈ V, Σ_A} ← A(1^l)
Separation of V to {Honest, Exposed, Compromised} ← A(Σ_A)
// Key Generation Phase:
foreach c s.t. φ[v] = Upgraded do
    if c ∈ Honest then
        // The attacker chooses whether the honest
           device already has a registered key or not.
        b ← A(c, Σ_A)
        if b=1 then
            c.σ_old, c.ν_old ← KeyGen(1^l)
        else
            c.σ_old, c.ν_old ← ⊥
        end
        c^S.ν_old ← c.ν_old
    end
    if c ∈ Exposed then
        c.σ_old, c.ν_old ← KeyGen(1^l)
        Σ_A ← c.σ_old
        c^S.ν_old ← c.ν_old
    end
    if c ∈ Compromised then
        c.σ, c.ν, c.σ_old, c.ν_old, c^S.ν_old ← A(c)
    end
    Σ_A ← c.ν_old
end
// Initializing the message pool
M = ∅
repeat forever
    Activated devices insert messages in the message pool M.
    Attacker chooses the next message to handle m ∈ M.
    Message m forwarding, according to the touring method ρ (see
      Subsection A-E)
    RESULT = TEST_P(G, φ, {c.ν}_{c∈V}, {c^S.ν}_{c∈V})
    if RESULT ≠ CONTINUE then
        Return RESULT
    end
end
```

Algorithm 4: Execution Overview

a) Initialization and helpers.: A *CrypTopology* protocol π is a message-driven-protocol [4], involving a server s and other devices. The *server* s is initialized with security parameter 1^l and a pair of public-private keys for signatures $s.\nu, s.\sigma$; the public key and 1^l are given to all upgraded devices. The server is also initialized with the network topology $G = (V, E)$ and the mapping $\phi : V \to \{$*Legacy,Upgraded*$\}$ of legacy vs. upgraded devices.

The server is further initialized with preset public verification keys denoted $c^S.\nu_{old}$, for a set $\mathcal{H} \subset V$ of upgraded devices; we refer to these as *helper* devices.

Formally, we define the group of helpers $\mathcal{H}$ as upgraded devices whose public keys are already validated by the server (or known otherwise, e.g., manually):

$$\mathcal{H} = \{h \in V \ s.t. \ \phi[h] = Upgraded \textbf{ AND } h^S.\nu_{old} \neq \perp\}.$$

The use of *helpers* is the main difference between CrypTopology and previous topology-based key setup works [2], [9], [10], [15], [20], [22], [23], which require disjoint routes between the client and the server. In contrast, the CrypTopology model also uses routes between the client and the helpers. This increases resiliency, allowing the use of more compromised devices for source routing and facilitating shortest-path routing.

Each *client* $c \in V$ is an *upgraded* device. During the protocol execution, the client c creates a pair of public (verification) and private (signature) keys, $c.\nu, c.\sigma$; it then attempts to register the public key $c.\nu$ with the server. The client terminates an execution period with an output of its registered key $c.\nu$. We denote $c.\nu = \perp$ if the client terminates without registering a key.

The server terminates an execution period, with value $c^S.\nu$ for every device $c \in V$, where $c^S.\nu = \perp$ if no key was set for device c. When all goes well, the key $c^S.\nu$ output by the server for client c is the same as $c.\nu$ output by c.

b) Signature scheme: The CrypTopology protocol π uses a secure Chosen-Message-Attack (CMA-Secure) signature scheme $\mathcal{S}$ = (*KeyGen,Sign,Verify*) [5]. It uses this scheme to sign and validate messages. In the general case, and specifically in the protocols we present, the signing and verification functions provided by π are not necessarily identical to the ones $\mathcal{S}$ provides. We denote the signing π functions by π.*Sign* and by π.*Verify*. π uses the same keys to provide secure signing for arbitrary client messages.

The signing functions receive as input a message and a signing key $c.\sigma$, and return the signature of that message. We also consider an *oracle* $\mathcal{O}_{\pi.Sign}$, which receives as input a message, and returns the signature on it. The verification functions receive as input a public-key for verification $c.\nu$, a message msg, and a candidate signature sig. They return *Accepted* if the signature of msg is sig, and return *Rejected* otherwise.

We focus on registering signature-verification keys, since these can be used to establish keys for encryption and other goals, using known key-exchange protocols (e.g., [4]).

c) Topology predicates.: The properties that the protocol ensures for a client c, depend on the client's location within the network topology, status (corrupted or not, upgraded or legacy), and initial state. To formulate the topological dependencies of properties, we define for each property a *topology predicate* $P_{s,G,\phi}(c)$ that returns 1 if the necessary conditions are met for client $c \in V$. The predicate may depend on the network graph G, on the position of the server $s \in V$, and on the device mapping function ϕ.

For example, for the k-connectivity predicate, $P^{k-conn}_{s,G,\phi}(c) = 1$ when c has at least k vertex-disjoint paths to the server s in the graph G. For simplicity of notation, we use $P^k_G(c)$ as the result of $P^{k-conn}_{s,G,\phi}(c)$, where s, G, ϕ are constants and known. Using these notations, previous works [2], [9], [10], [15], [20], [22], [23] ensure key establishment in the presence of n_A compromised devices, for devices c s.t. $P^{2n_A+1}_G(c) = 1$.

d) Protocol execution.: To facilitate provable security, we adopted a precise execution model; (see also Algorithm 4). The execution is a process **EXEC**$^{TEST}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l)$that receives as input: the protocol π, a security parameter 1^l, a probabilistic

polynomial time (PPT) algorithm as the adversary $\mathcal{A}$, and a routing method ρ. The execution initializes every device. Compromised devices are initialized by the adversary and exposed, and honest devices are initialized by the protocol. Exposed devices, however, provide their private keys to the attacker.

The execution activates π on each upgraded and honest device. In addition, the execution activates the attacker $\mathcal{A}$ on corrupted devices. Here, the attacker determines the order of relayed messages.

A final input to the execution is an efficient algorithm denoted TEST. The execution activates TEST after handling every message, to test whether the specified requirements were satisfied. We define different TEST algorithms for different protocol requirements.

A TEST algorithm receives as input the network properties G, ϕ and for all of the devices, the registered verification keys at the client and at the server $\{c.\nu\}_{c\in V}, \{c^S.\nu\}_{c\in V}$, and the preset keys $\{c.\nu_{old}\}_{c\in V}, \{c^S.\nu_{old}\}_{c\in V}$. In addition, the TEST receives a topology predicate P, and verifies that the tested property holds for every device c s.t. $P(c) = 1$. We often abuse the notation and write $TEST_P$ to refer to $TEST_P(G, \phi, \{c.\nu\}_{c\in V}, \{c^S.\nu\}_{c\in V})$.

The TEST checks the output and state of different devices, and has three possible outcomes: *Continue, Success,* and *Fail*. When the TEST outputs Continue, the execution continues; it is not yet a success or a failure.

The execution process is *adversarial* in the sense that the attacker $\mathcal{A}$ can choose most parameters, including the topology $G = (V, E)$ $s.t.$ $|V| = n$, the weight function w in shortest-path routing, the server device $s \in V$, and, for each device $c \in V - \{s\}$, the legacy/upgrade state $\phi(c)$. The adversary assigns each device $c \in V$ to be one of the following: *{Honest, Exposed, Compromised}*. For *Compromised* devices, the attacker also chooses the internal state (e.g., the private keys).

For each of the upgraded devices u, the attacker $\mathcal{A}$ chooses whether the device already has a shared key with the server or not. If the device has a shared key, the execution will choose $u.\nu$ and $u.\sigma$, and provide $u.\nu$ to the server. In addition, the attacker chooses s to be an *honest upgraded* device.

Every message that is sent or relayed by the protocol includes the sender address as the message *source*, the recipient address as the message *destination* , the message *payload*, and the *destination-route* field. The sender sets the *destination-route* according to the routing method, as described in Section A-B.

Whenever a device sends or relays a message, the execution inserts the message to a shared global *message pool*, and attaches the *sending time* to the message. In addition, the execution attaches the address of the *next hop* device that is on the route of the message, according to the routing method and to the sending/relaying device.

During the execution process, the attacker is able to see (only) the source, destination, and insertion time fields of all the messages in the message pool; it can also choose which message the execution process will handle next. We limit the attacker to delaying messages no more than T_{delay} time, and to delivering at most one message at each time.

APPENDIX B
ADDITIONAL PROPERTIES

BOUND $(T_{max}, \{c.\nu, c^s.\nu \mid c \in V\}, Compromised \subset V)$

```
// The test shows Success if after T_max time all the
   devices present their output. Without loss of
   generality, we assume that the current_time at the
   beginning of the execution is 0.
if current_time() ≥ T_max then
    foreach c ∈ V − {s} s.t φ[c] = Upgraded do
        if c did not terminate then
            Return FAIL
        end
    end
    Return SUCCESS
else
    Return CONTINUE
end
```

Algorithm 5: Termination within T_{max} test

GKS $(\{c.\nu, c^s.\nu \mid c \in V\}, Compromised \subset V)$

```
// The test Fails if after all non-compromised devices
   terminated, there is a device that fulfills the
   topology predicate and did not register its key
Return CONTINUE until all the upgraded devices terminated.
// Check that all the keys were set correctly.
foreach c ∈ V s.t. P(c) = 1 AND φ[c] = Upgraded AND c ∈
  Honest ∪ Exposed AND c^s.ν_old = c.ν_old do
    if c.ν = ⊥ OR c.ν ≠ c^s.ν then
        Return FAIL
    end
end
// If all the devices have a key, and the key is
   correct, the Test succeeded
Return SUCCESS
```

Algorithm 6: Guarantee key setup test

COR-M $(\{c.\nu, c^s.\nu \mid c \in V\}, Compromised \subset V)$

```
// The test Fails if the attacker was able to set
   different keys in the server and in the client, for
   an honest device that already had a signed key
   c.ν_old ≠ ⊥ .
foreach c ∈ V − {s} φ[c] = Upgraded AND c ∈ Honest do
    if c.ν_old ≠ ⊥ then
        if c.ν ≠ c^s.ν then
            Return FAIL
        end
    end
end
Return CONTINUE
```

Algorithm 7: Correctness resiliency to MitM test

In this section we prove some more advanced properties of the protocols. The properties include Bounded Termination, Guaranteed key setup and Correctness resiliency to Man-in-the-Middle.

We say that Protocol π *ensures* requirement TEST, with respect to predicate P and routing method ρ, if for all PPT attackers $\mathcal{A}$, $\gamma > 0$, $\exists l_0$ $s.t.$ $\forall l > l_0$:

$$Pr(\textbf{EXEC}^{TEST}(\mathcal{A}, \pi_{\mathcal{S}}, 1^l) = \text{FAIL}) \; < \; l^{-\gamma},$$

where the probability is taken over the random coins used by $\mathcal{A}$ and $\textbf{EXEC}^{TEST,\rho}(\mathcal{A}, \pi, 1^l)$.

a) Secure signing (Algorithm 2).: Protocol π ensures *secure signing*, if for every attacker $\mathcal{A}$, the attacker cannot generate a pair of message msg_A and signature $sign_A$ for a non-compromised device c, without knowing the private key of c, $c.\sigma$. The TEST is based on the forgery test of the chosen-message-attack secure signature scheme [13]; the key generation is done internally in the client and not by the experiment.

The signing requirement is related to the secrecy of $c.s$, the key generated by the *client*; if this key is exposed, surely *signing* cannot hold. The remaining requirements deal with the public key c_v^S that the server (should) learn for client c.

We next present the complementary *correctness* requirement: the server should learn the correct public key $c.v$ chosen by the client. Note, correctness and secure signing together achieve a non-trivial requirement that the registered key at the server is secret and allows secure signing *and* verification.

b) Correctness (Algorithm 3).: Protocol π ensures correctness, with respect to predicate P, if for every non-compromised device c s.t. $P(c) = 1$, whenever the server outputs a key $c^S.\nu \neq \perp$ for specific client c, then the client c outputs the same key $c.\nu$.

c) Guaranteed key setup (Algorithm 6).: Protocol π *guarantees key setup*, with respect to predicate P, if **every** non-compromised upgraded device c, s.t. $P(c) = 1$ and $c^S.\nu_{old} = c.\nu_{old}$, the server registers a key $c.\nu \neq \perp$, which is the same key as the server registered $c^S.\nu = c.\nu$.

d) Correctness resiliency to Man-in-the-Middle (Algorithm 7).: Protocol π is *MitM-resilient*, if for every **honest** device s.t. $c^S.\nu_{old} = c.\nu_{old} \neq \perp$, the new keys that will be registered at the server and at the client will be identical $c^S.\nu = c.\nu$. This property is based on the use of cryptography and secret keys, and not on the topology. Therefore, it holds even for a MitM attacker that controls all the routes between the client and the server.

A. Analysis

Termination within time $\mathbf{T_{max}} = 8|\mathbf{V}|\mathbf{T_{delay}}$. We show that $CrypTop$ execution is bounded. Specifically, we show that after $T_{max} = 8|V|T_{delay}$, all the devices and the server terminate and present their output.

Theorem 3. ***[Termination within time $\mathbf{T_{max}}$]****: Protocol $CrypTop$* ***ensures termination within time $\mathbf{T_{max}}$****, for every $T_{max} > 8|V|T_{delay}$.*

Proof: Every message of the protocol can be sent through a maximal route that includes all the devices in the network, n. Therefore, the maximal delay of a single message is $|V|T_{delay}$, where T_{delay} is the maximal delay on an edge.

The protocol has eight sequential messages, which therefore bounds the maximal execution time of the protocol to $8|V|T_{delay}$. ∎

Theorem 4. ***[Guaranteed Key-Setup]****: Protocol $CrypTop^{\mathcal{S}}$* ***ensures guaranteed key-setup*** *for every CMA-secure signature scheme $\mathcal{S}$, with respect to predicate $P_{AUG}^{2n_A+1} \wedge P_G^{n_A+1}$.*

Proof: For every non-compromised device c s.t. $P_{AUG}^{2n_A+1}(c) = 1$, the correctness property holds, namely if $c^S.\nu \neq \perp$ then $c^S.\nu = c.\nu$. It is left to prove that $c^S.\nu \neq \perp$.

First, we will show that c can receive $n_A + 1$ signed responses. From the topology conditions of $P_{AUG}^{2n_A+1}(c) = 1$, c has at least $2n_A + 1$ disjoint routes to helpers. In the presence of n_A compromised devices, the attacker will be able to block n_A signed responses, which allows $n_A + 1$ responses to reach c. Therefore, c can provide $n_A + 1$ signed responses.

Second, we will show that c and s can communicate with CrypTop, which is necessary for the key registration. The topology condition $P_G^{n_A+1}(c) = 1$ ensures that there will be at least one route between s and c without a compromised device. Therefore, the attacker will not be able to block CrypTop communication between c and s.

∎

Correctness resiliency to Man-in-the-Middle. We show that for any honest device c that has a preset key $c.\nu_{old} \neq \perp$, no attacker will be able to cause the server to sign an incorrect key for c, namely, $c^S.\nu \neq c.\nu$.

Theorem 5. ***[Correctness Resiliency to Man-in-the-Middle]****: Protocol $CrypTop^{\mathcal{S}}$* ***ensures correctness resiliency to Man-in-the-Middle****, for every CMA-secure signature scheme $\mathcal{S}$.*

Proof:

We need to prove that for any CMA-secure signature scheme $\mathcal{S}$ = (*KeyGen,Sign,Verify*), and for all PPT attackers $\mathcal{A}$, $\gamma > 0$, $\exists l_0$ *s.t.* $\forall l > l_0$:

$$Pr(\mathbf{EXEC}^{COR-M,\rho}(\mathcal{A}, CrypTop, 1^l)\text{=FAIL}) \; < \; l^{-\gamma},$$

where the probability is taken over the random coins used by $\mathcal{A}$ and $\mathbf{EXEC}^{COR-M,\rho}(\mathcal{A}, CrypTop, 1^l)$.

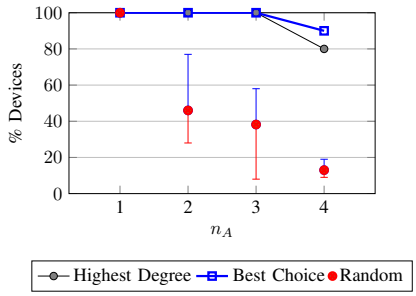
Let $\mathcal{S}$ be a CMA-secure signature scheme [5].

For every honest device c (including the server), the secrecy property holds. Therefore, the attacker will not be able to provide a signed message with a probability greater than ϵ. It is left to show that since the secrecy property holds, for every honest device c s.t. $c^S.\nu_{old} = c.\nu_{old}$ then $c^S.\nu = c.\nu$. This is derived from the CrypTop design since without valid signatures of the server and the client, the protocol stops and does not register a new key. Therefore, in case of not-verified signatures, $c^S.\nu = c.\nu = \perp$ and the preset keys will not change, $c^S.\nu_{old} = c.\nu_{old}$.

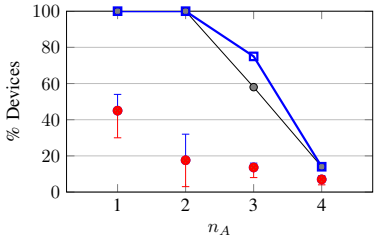
∎

APPENDIX C
CHOOSING THE SERVER

The results of CrypTopology protocols depend on the specific choices of the relevant devices: the server and the helpers. In the previous section, we measured the maximal number of available devices, while choosing the device with the *highest degree* as the server. In this section, we compare this highest-degree method to two other methods: *random* and *best-choice*. In the random strategy, the simulation chooses the server randomly. In the best choice method, the simulation

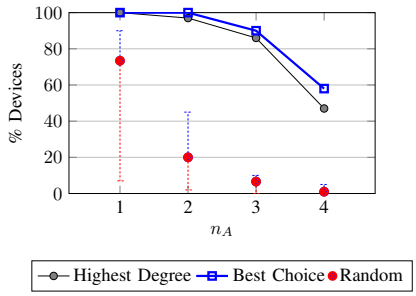


(a) Source routing

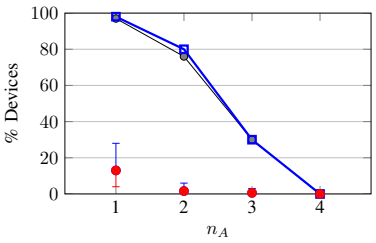


(b) Shortest-path

Figure 4: Recovery ratios, using different methods for choosing the server. Results are shown as a function of the number of corrupted devices n_A, and for ISP 4755 network. For the random method we also show the (wide) range of results.

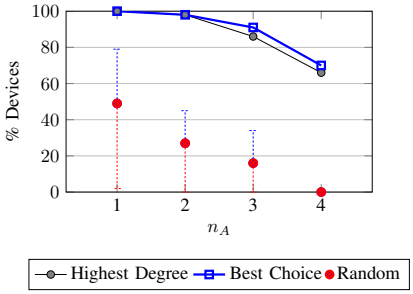


(a) Source routing

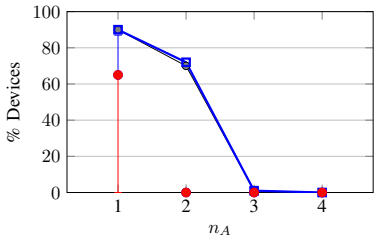


(b) Shortest-path

Figure 5: Recovery ratios, using different methods for choosing the server. Results are shown as a function of the number of corrupted devices n_A, and for the IEEE 118 network. For the random method we also show the (wide) range of results.



(a) Source routing



(b) Shortest-path

Figure 6: Recovery ratios, using different methods for choosing the server. Results are shown as a function of the number of corrupted devices n_A, and for the IEEE 300 network. For the random method we also show the (wide) range of results.

iterates over all the devices, and chooses the server to be the device that results in the maximal number of available devices.

We evaluated the impact of the selection method on the recovery ratio. This is impacted only by the choice of the server, since the recovery ratio measurement assumes that all the devices already have a key in the first round of the simulation.

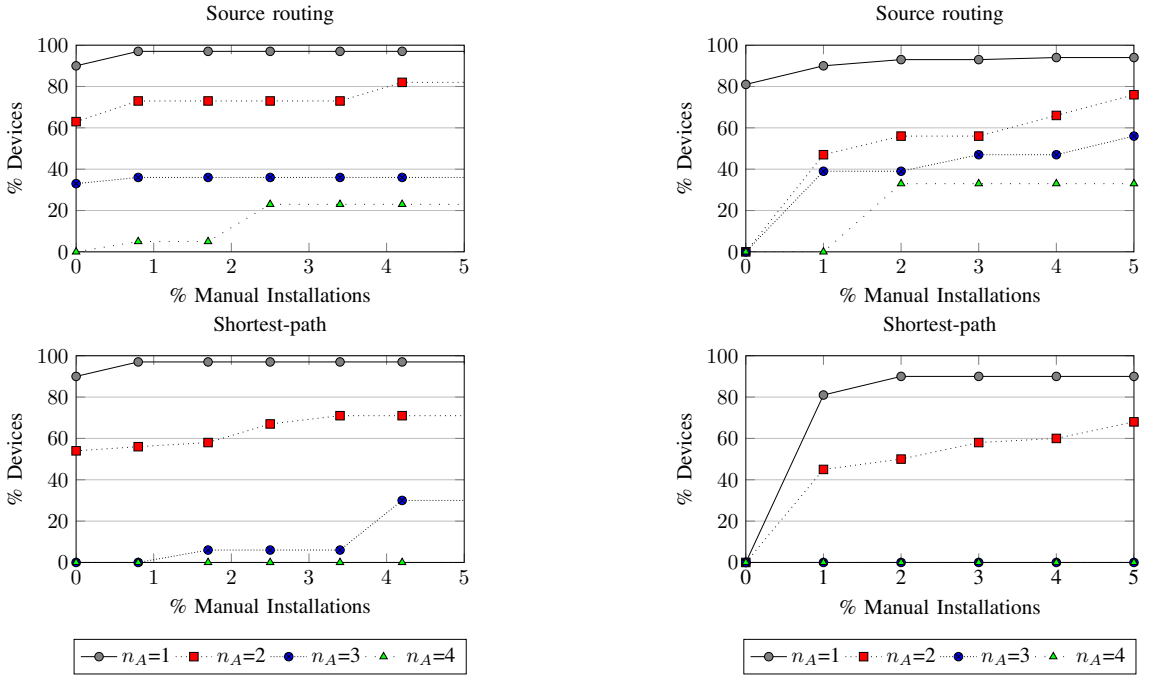
The results comparing the different methods are shown in Figures 5,4 for IEEE 118 and ISP 4755, respectively. On each graph for the random method, the circle points represent the average maximal availability. The blue and the red line represent the maximal and minimal results over five simulations of randomly choosing the server. In addition, each graph presents the results received using the highest degree and best choice strategies.

The best-choice method chooses the optimal server based on the maximal availability. If we compare the best-choice strategy to the highest degree method, we see that in most cases the highest degree gives very close, even identical, results. In contrast, the random strategy is significantly and consistently inferior, especially for $n_A > 1$.

What can we learn from these results? Our main conclusion is the impact of the choice of the server, which has a substantial influence on the results. We compared the highest degree and the best choice strategies to the random strategy. Observing the results, it is clear that the first two strategies present a much better recovery ratio. Hence, having a strategy for choosing the server is important for a higher ratio.

We also compared the highest degree strategy to the

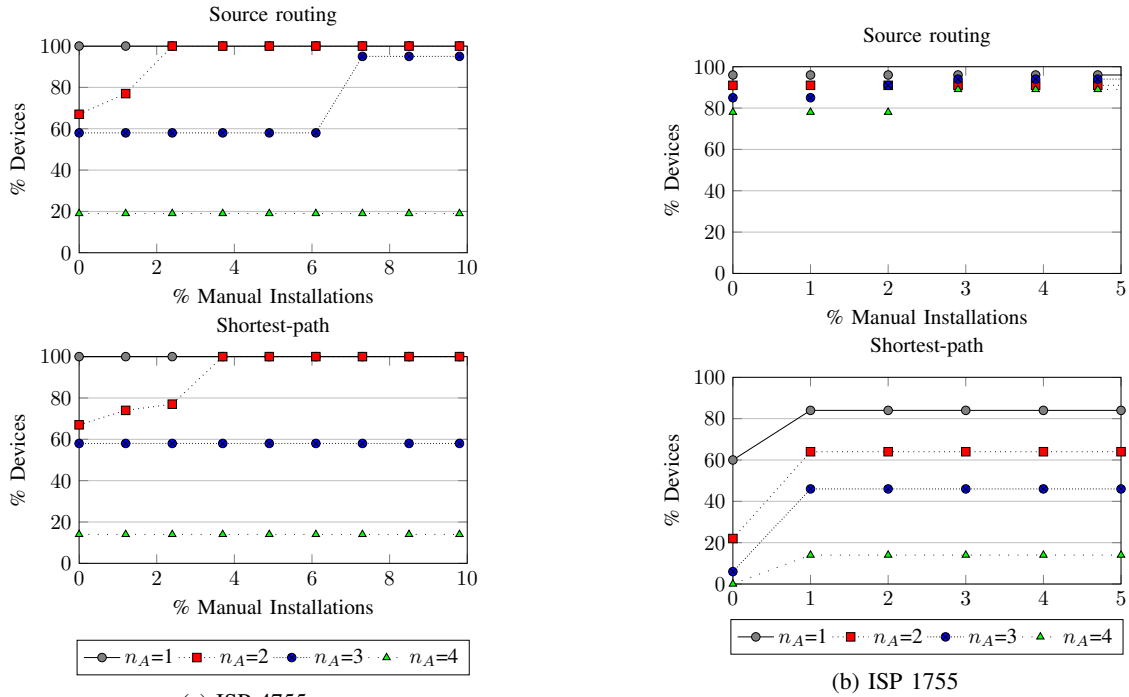
optimal, inefficient, best-choice strategy. We found that the highest degree strategy, even though it may not be optimal, presents results that are very close to the optimal strategy. A further study is required to determine an efficient, yet optimal, method.



(a) IEEE 118

(b) IEEE 300

Figure 7: Power Networks: Incremental deployment ratios for different amounts of manual installations, using CrypTop and SP-CrypTop.



(a) ISP 4755

(b) ISP 1755

Figure 8: Internet Service Providers (ISP): Incremental deployment ratios for different amount of manual installations, using CrypTop and SP-CrypTop.