Topology-based Plug-and-Play Key-Setup

Amir Herzberg and Yehonatan Kfir

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

Abstract. We study the use of known, well-connected network topology, to improve key setup and management. Specifically, we present the TopKeyS protocol, that uses such known-topology to allow easier, *plug and play* key distribution. Furthermore, ,TopKeyS *improves security* by limiting impact of key exposures: it ensures both *perfect forward secrecy* and *proactive key refresh*, re-establishing security after exposure.
TopKeyS assumes a trusted *the authentication server* device, whose public key is known to all devices. This server is initialized with the network topology.
We analyze the properties of the TopKeyS 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-TopKeyS, an improvement of TopKeyS that is secure even for *Adversarial Routing*.

1 Introduction

We revisit the approach of using network topology to secure communication. This approach was first proposed by Dolev et al., in the seminal paper [6]. They showed how known and sufficiently-redundant network topology, allows perfectly-secure communication - i.e., avoiding complexity assumptions. Essentially, senders secret-share the secret messages and send shares over disjoint routes to the destination.

However, the results of [6] had limited applicability, for several reasons. The first reason is that their focus was on providing information-theoretic security, i.e., avoiding complexity-based cryptography. This is surely a worthwhile goal; however, we believe (and show) that topology can be very useful also to *complement* complexity-based cryptography. First, we show that topology can allow 'plug-and-play' key-setup, allowing secure initialization of keys in devices, using only the public key of a trusted 'authentication server'. Second, we show that topology allows to *improve* security, specifically, recovery from penetrations (*proactive security*).

A second reason limiting the applicability of [6] is that their results assumed *source routing*. Source routing is rarely available in practical networks, which usually use some type of shortest-path routing; furthermore, routing protocols may be vulnerable to attacks. We consider routing as part of our model, and in particular, consider both shortest-path routing as well as adversarial routing.

In total, our result utilize the fact that in many organizational networks, topology is known and quite stable, with considerable redundancy (e.g., for resilience). *Key setup* is a significant challenge in deploying cryptography within such organizational networks, which often contain hundreds or thousands of components. Each component 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 reinitialized if key exposure is suspected. This recovery process is another challenge in the deployment of cryptography.

We take advantage of the known topology and disjoint routes to *authenticate* messages and, in particular, to set up keys. We authenticate the sender by measuring the *distance and route* from the sender to multiple trusted nodes, based on different routing models (assumptions). This also allows us to perform proactive key refresh, similar to Canetti et al. [3], without depending on attack-detection at the device.

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.

One important example for relevant organizational networks, are SCADA and Industrial Control networks. For example, consider the network in Figure 1, which is based on the IEEE 9-bus model [15] - 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 secure (and already upgraded). 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, through edges 1a and 2a, respectively. Only after receiving this evidence, will the server authenticate Device 5 and its cryptographic keys.

To facilitate our design and analysis, we formalized the properties of networks that can use topology-based key setup.
(1) Known topology: The network topology and the routing method are known.
(2) Trusted nodes: a small number of secure, trusted nodes.
(3) Known public key: One of the trusted nodes, the *authentication server*, has a private key, with the corresponding public key known to all (upgraded) nodes.
(4) Safe recovery: An adversary may corrupt some nodes, exposing all of their secrets; however, upon recovery from corruption, nodes return to run the protocol as designed, with the correct public key for trusted node(s).
(5) Disjoint paths: Nodes in the networks have at least two disjoint shortest paths to different trusted nodes.
(6) Use of IP and TTL: Modern wide-area networks generally use the Internet Protocol (IP) with intermediate devices acting as routers. These devices follow

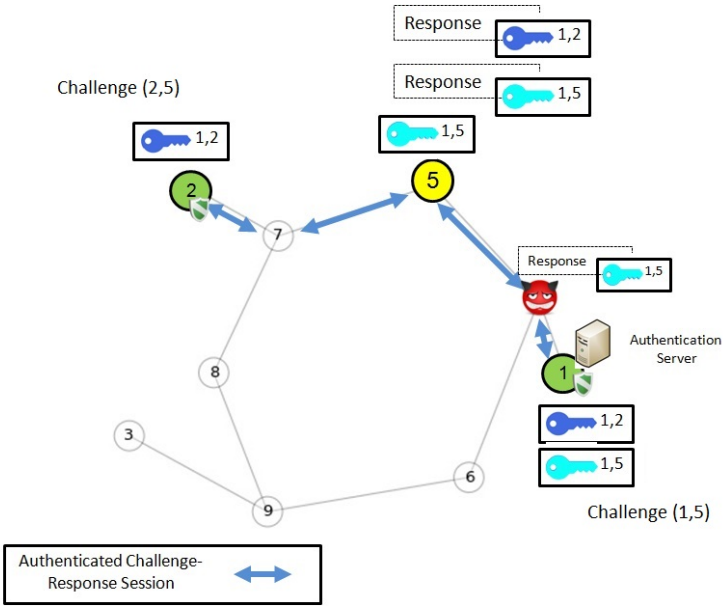

Fig. 1. The IEEE 9-bus model [15] of a small power communication network. The topology-based key setup protocol provides plug-and-play initial key setup and proactive key refresh. The protocol uses multiple authenticated challenge-response sessions, between a client (Device 5) and several trusted devices (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 trusted devices

the TTL hop-count rules [14] and usually use simple shortest-path routing without weights (e.g., provided by the RIP protocol [13]).

Contributions. This work makes the following contributions:

In this work, we present precise model and provably-secure results motivating the use of topology properties to improve deployability and/or security of cryptographic systems. This is in contrast to previous works, using topological properties to avoid complexity-based assumptions, i.e., to use information-theoretic security instead of security based on complexity assumptions.

We further present specific protocols for key setup. Our protocols feature plug-and-play deployment, without requiring installation of per-device secret keys, allowing *incremental deployment* and assuming *different, realistic routing* models. This includes the (practical) case of shortest-path routing models, in contrast to previous works which mostly assumed source routing. Furthermore, we present a necessary extension for the case of malicious routing.

Our protocols further feature security benefits, in particular, *proactive and forward security*, ensuring the the overall 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 [7,8,12,16,18] study the use of network topology to create a shared secret between parties. However, all of these works assumes that the route of the message can be choose, while this is not the case in a realistic network model that contains message routing. Moreover, in realistic networks, the exact routing of messages is not always known, even if the routing method (e.g., shortest-path routing) is known.

Zhu et al. [18] show how to create a shared secret key between two devices in the network, using disjoint channels. However, they did not present a recovery mechanism or a realistic routing model of the communication network. Although our work initially uses the topology to set a key for cryptographic protocols, it does not rely only on the topology for long-term security.

Bellare et al. [2] formulated the definition of key setup protocols. They defined the execution process of such protocols and the definitions of their security. We extend their model [2] to include the network topology and routing method. In this way, we can define the properties of protocols that use the network topology and routing method for key setup.

Canetti et al. [4] present a formulated model for building a secure key-exchange protocols. According to their formulation, the security of a key-exchange protocol can be proved in an ideal communication model, where the channels are authenticated. Then, using generic tools it can be translate to realistic adversary-controls links. The authentication channels in that work assumed a cryptography (and thus, a pre-shared key) for authenticating the message. In our work, we present a method for achieving authenticated channels, using the topology and the routing of the network. In that way, we eliminate the need of sharing a secret key between the parties in the protocol.

Our work is related to the research of Gilad and Herzberg [9], since we share the same motivation for enabling zero-configuration key setup. Their method assumes the existence of an anonymous communication network between the devices that participate in the protocol. Our method differs because it is based on the assumption that the network topology and the network routing method are known to at least one of the devices that participates in the key-setup protocol.

Our work is also related to several works [1,17] , that use non-cryptographic methods to create a shared cryptographic key. In our method, we use the topology location of the device in the network for authentication, and for setting a shared key. We also prove the security of our 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*, 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.

If the network uses the IP protocol, we model an additional property. IP packets have a *TTL field* [14], which defines the number of hops the packet is allowed to pass. The maximal TTL value that can be set for a packet is 255; for simplicity, we assume all non-corrupt devices initialize the TTL to 255 upon sending a packet. We define a *TTL-Network* as one in which devices obey the 'TTL rules', as follows:

Definition 1 ***[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.*

Because availability and security are a primary concern, at least some devices in the network must be well-monitored to ensure (with high probability) that they will not be compromised. We call these devices *trusted devices*, $T \subset V - L$. Since this high level of security requires large operational efforts, often only a small portion of the upgraded devices are also trusted. Some of the non-trusted devices, whether legacy or upgraded, may be compromised.

Examples of trusted devices are Certificate Authority servers. Since these servers are critical for the security of the network, they are highly-secure.

One of the *trusted devices* is the authentication server s, which has a known public key, $s.pu$. It also has a shared secret key with each of the trusted and upgraded devices. Using these keys, the server can send and receive encrypted and authenticated messages.

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. 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*[1]. 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.

[1] 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.

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 model 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. The only exception is that compromised devices are not obliged to forward the message as per the route carried in the message.

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 $\Re_0 : V \times V \to V$, which receives the current device and the destination, and returns the neighbor of the current device, to which the message is forwarded $\Re_0(current, destination)$, such that the sequence of forwarding 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 $\Re_0$, 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, *trusted* and *compromised* devices are not bound by the routing method and can freely select the edge from which to forward each message.

2.2 Protocol Model

Our model is based on that of Bellare et al. [2] 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 *topology-based key setup* protocol π is a message-driven-protocol [2] that has three types of participants in its execution:

Server - A *trusted* 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 coloring function ϕ, 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*; *Success* with a pair (k_s^{OUT},*c*) of key k_s^{OUT} of device *c*; and *Timeout*, when it waits for messages longer than a predefined time threshold.

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 - A *trusted or 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 $P(v, G, \phi, \rho, \Re) = 1$.

The server protocol is activated by a key-request message from a client c, only if $P(c, G, \phi, \rho, \Re) = 1$. Upon activation, the server s uses messages to/from a group of collaborating devices, in order to authenticate the client location. Upon successful authentication, the server registers the client key k_c, where k_c is the key of device c.

If the $P(c, G, \phi, \rho, \Re) = 1$ but the client location cannot be authenticated, the server output will be an Alert. In that case, the server will not register the client key.

For every topology-based key setup protocol π, we define several properties; these will be discussed in Section 3.

3 Problem Formulation

Our problem formulation is based on the execution model by Bellare et al. [2], extended to support known topology and routing models.

3.1 Asynchronous Model

Execution of a topology-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 topology-based key setup protocol π, a topology predicate P, and a security parameter 1^l. We denote this execution by $\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l)$.

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

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 and one *trusted* device as the server.

The output of the execution is the attacker state $\sigma_{\mathcal{A}}$, and 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})* - if the server registers the same key as the client, k_c^{OUT}; and (4) *"Timeout"* - if the server output is "Timeout

We define the following properties of topology-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 2 ***[Secrecy]***

Protocol π ensures secrecy with respect to predicate P, *if* $|Pr\left[IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l)=1\right]-\frac{1}{2}|$ *is a negligible function (in security parameter l), for all PPT attackers $\mathcal{A}$ and $\mathcal{A}_1$, and where $IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l)$ is defined in Algorithm 1.*

Indistinguishability Experiment $IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l)$

1. $\mathcal{A}_1$ chooses $k_0, k_1, S_{ID}^0, S_{ID}^1 \leftarrow \{0,1\}^l$. With them, it creates two activation messages m_0, m_1 *s.t.* $m_i = \{k_i, S_{ID}^i\}$.
2. A random bit is chosen, $b \xleftarrow{\$} \{0,1\}$
3. A successful execution, with activation message equal to m_b, is chosen randomly, $(\text{"}Success\text{"}, k_c^{OUT}, \sigma_{\mathcal{A}}) \xleftarrow{\$} \mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l)$ *where* $m_{init} = m_b$
4. Return 1 if $\mathcal{A}_1(k_c^{OUT}, \sigma_{\mathcal{A}}) = b$, 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 3 ***[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.:*

$Pr(\boldsymbol{EXEC}(\mathcal{A}, \pi, P, 1^l) = (\text{"}Failure\text{"}, \sigma_{\mathcal{A}})) < negl(l)$,

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

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 4 ***[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 5 ***[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 $\mathcal{A}$:*
EXEC$(\mathcal{A},\pi,P,1^l) = \{"Success", k_c^{OUT}, \sigma_{\mathcal{A}}\}$.

3.2 Asynchronous Execution

For each of the defined properties, the attacker1'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 message1'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 4, 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.

3.3 Synchronous Model

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. All the parties in this model have synchronized clocks that proceed at the same rate.

We denote a synchronous execution $\mathbf{EXEC}^{\mathbf{SYN}} = \mathbf{EXEC}^{\mathbf{SYN}}(\mathcal{A},\pi,P,1^l,t_0))$, where t_0 denotes the starting time of the execution. A synchronous execution has all the same outputs as the asynchronous model, with the additional output of Timeout.

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:

```
EXEC(A,π,P,1^l)

// Initialization - see Algo. 3
{G, φ, ρ, ℜ, A, c, s, σ_A, σ_c, σ_s, m_Init, k_c^OUT} = Init_Execution(A,π,P,1^l)

M = ∅
Add_Message(m_Init,M) // see Algo. 4
while True do
    M' = ∅
    m̂ = A(σ_A)
    switch φ(M[m̂].next_hop) do
        case Compromised do
            {M', σ_A} = A(σ_A, M[m̂]) M[m̂] = ∅
        end
        case Trusted OR Upgraded do
            v = M[m̂].next_hop
            {M', σ_v, k_s^OUT, c', isAlert} = π(σ_v, M[m̂])
            if isAlert then
                Return ("Alert",σ_A)
            end
            if v = s AND k_s^OUT ≠ NULL AND c = c' then
                if k_s^OUT = k_c^OUT then
                    Return ("Success",k_c^OUT,σ_A)
                else
                    Return ("Failure",σ_A)
                end
            end
            M[m̂] = ∅
        end
        case Legacy do
            Decrement M[m̂].ttl by 1
            if M[m̂].ttl = 0 OR M[m̂].next_hop = M[m̂].destination then
                M[m̂] = ∅
            else
                // The message should be routed to the next device
                if ρ = source then
                    // In source-routing the messages are route according to the
                       routing list. See section 2.1
                    M[m̂].next_hop = M[m̂].route[next_hop]
                else
                    // adversarial or shortest-path routing
                    M[m̂].next_hop = ℜ(M[m̂].next_hop, M[m̂].destination)
                end
            end
        end
    end
    foreach m ∈ M' do
        Add_Message(m,M)
        σ_A = A(σ_A, m)
    end
end
```

Algorithm 2: Execution Process

Bounded Termination. - A key-setup protocol ensures bounded termination if the protocol1's execution time is bounded, possibly as a function of the network topology $G = (V, E)$.

Definition 6 ***[Bounded Termination]*** *Let t_0 be the time the execution game started and $G = (V, E)$ the network topology.*

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

/* 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.:
$\phi:V\rightarrow\{Legacy,Trusted,Compromised,Upgraded\}$
$\rho\in\{source,shortest\_path,adversarial\}$
$\phi(s)=$ Trusted
$P(c,G,\phi,\rho)=0$ **AND** $\phi(c)=$ Upgraded

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

$k,S_{ID}\xleftarrow{\$}\{0,1\}^l$
$m_{Init}\xleftarrow{\$}\{k,S_{ID}\}$ **OR** receives as an input (for the Secrecy proof only).
$\{\sigma_c,k_c^{OUT}\}\leftarrow\pi.Init\_Client(s.pu,k,\rho,1^l)$

/* 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**
  $\mathcal{A}\rightarrow\Re_0:V\times V\rightarrow V$
  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**
    $\Re_A\leftarrow\mathcal{A},s.t.\Re_A:V\times V\rightarrow V$
    $\sigma_{\mathcal{A}}=\sigma_{\mathcal{A}}\cup\Re_A$
    $\Re=\Re_A$
  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

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

Perfect Forward Secrecy. - We use the definition from [5]. A key-setup protocol ensures forward secrecy if the disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier times.

Proactive Security. - A key-setup protocol ensures proactive security if it has a mechanism for periodically recovering from key compromise, and the recovery process is faster than the time it takes for the attacker to compromise devices. In this way, the protocol helps resist an attacker that tries to increase the number of compromised devices.

We divide the network's coloring function ϕ into time slots. Each time slot is T_P long. At the start of each period, the protocol executes a *refreshment*

```
Procedure Add_Message(m,M)
    Add to M element with:
        source = m.source
        destination = m.destination
        payload = m.payload
        ttl = m.ttl
        if ρ = source then
            route = m.route
            next_hop = m.route[m.source]
        else
            next_hop = ℜ(m.source, m.destination)
        end
```

Algorithm 4: Message Handling

protocol, which changes the keys of all devices in the network. We denote the coloring function at time t as ϕ_t.

During each 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.

Definition 7 ***[Proactive Security]***

Protocol π ensures proactive security, if for every attacker that is able to compromise n_A new devices at T_P time, there exists N_A, such that for all t:

$|\{\ v \in V\ s.t.\ \phi_{t+T_P}(v) = Compromised\ \}| - |\{\ v \in V\ s.t.\ \phi_t(v) = Compromised\ \}| < N_A$

4 Topology-based Key-Setup Protocol (TopKeyS)

In this section, we present an implementation for a topology-based key setup, called TopKeyS.

The goal of the TopKeyS protocol 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 [11]. We denote a TopKeyS that uses these schemes as $TopKeyS^{\xi,\mathcal{M}}$. 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.

k	The random generated key at the client. Will be used for deriving the other keys.
k_c^{OUT}	The key the will be registered for the client, $k_c^{OUT} = PRF_k("Client\_Key")$.
k_c^{AUTH}	The key the will be used during topology-based key setup execution, to authenticate messages between the server and the client, $k_c^{AUTH} = PRF_k("Authentication")$.
$\xi_{s.pu}\{\cdot\}$	Asymmetric encryption with scheme ξ and public key $s.pu$
$\{\ \}_{k_c^{AUTH}}$	Symmetric authenticated-encryption with key k
$s.pr, s.pu$	The authentication server private and public keys
$G = (V, E)$	The ICS network graph, where V are the devices and E are the connections between them.
L	Group of legacy devices, $L \subset V$
T	Group of trusted devices, e.g. devices which share private key with the server, and attacker is not able to control. $T \subset V - L$.
$\Re_0$	The shortest-path routing function.
ϕ	The coloring function, representing the device types. ϕ :V × V → {Legacy,Trusted,Compromised, Upgraded}
ρ	The routing method. $\rho \in$ {source,shortest_path,adversarial}
σ_v	State of device v
$k \xleftarrow{\$} \{0,1\}^l$	Choose k randomly from $\{0,1\}^l$

Fig. 2. Frequently used notations

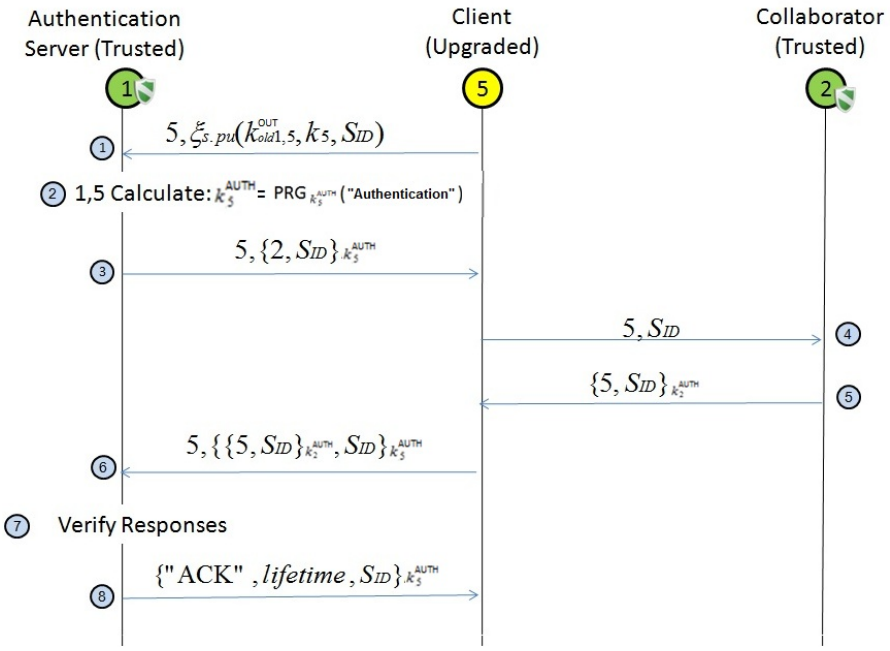

Fig. 3. Example for a successful key-setup session, for the network in Figure 1. Device 5 receives keys, after two challenge-response sessions, with Device 1 and Device 2. Device 1 is the authentication server.

4.1 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 $k \xleftarrow{\$} \{0,1\}^l$.

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. In addition, the client sends its previous key shared with the server k_c^{OUT}. If it is the first time the client is requesting a key, k_c^{OUT} will be set to $Null$. 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 $k_c^{AUTH} = PRF_k(\text{"}Authentication\text{"})$, as well as a shared secret key $k_c^{OUT} = PRF_k(\text{"}Client\_Key\text{"})$. 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 trusted and upgraded devices, $T_c \subset T$, 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 T_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", the session ID S_{ID}, and the key lifetime. After the lifetime period, the key will

no longer be used to authenticate messages between the server and the client. Henceforth, the client will have to initiate new key setup sessions with the server.

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}.

In the Synchronous Model we define additional properties and condition:

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 3.

4.2 Protocol Properties

In this section we present the topology conditions necessary to ensure that the TopKeyS protocol has the properties defined in Section 3. Formal proofs are available in the full version of the paper [10].

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 *trusted* 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 TopKeyS 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 trusted or 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. We include only the proof for correctness in this version of the paper, as this is the main novel property of the protocol. The rest of the proofs are in the full version of the paper, due to length limitations:

Theorem 1. ***[Secrecy]****: For every CPA-secure public-key scheme ξ (and every MAC scheme $\mathcal{M}$), protocol $TopKeyS^{\xi,\mathcal{M}}$ ensures secrecy, as defined in Definition 2.*

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

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

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 $PRF : \{0,1\}^l \rightarrow \{0,1\}^{l'}$, $l < l'$.

Assume that PRF is an ideal random function U, $U : \{0,1\}^l \rightarrow \{0,1\}^{l'}$, $l < l'$. 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 $PubK^{cpa}_{\mathcal{A}_\xi,\xi}(l)$ [11] :

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 $EXEC^{\mathcal{A}_\xi}$ $(\mathcal{A}, \pi, P, 1^l, m_b)$ until the output is "Success". $\mathcal{A}_\xi$ returns $b' = \mathcal{A}_1(k, \sigma_\mathcal{A})$.

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

According to the assumption:

$|\ Pr\ [\ \mathcal{A}_1(k_c^{OUT}, \sigma_\mathcal{A}) = b\ ]\ >\ negl(l) \Rightarrow$

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

and this is contradiction to the assumption that ξ is CCA secure. Thus, if ξ is CPA-secure then $|Pr\left[IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l) = 1\right] - \frac{1}{2}| \ < \ negl(l)$.

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

Theorem 2. ***[Correctness]****: For every CPA-secure public-key scheme ξ, and MAC-secure scheme $\mathcal{M}$, that include predicate P^{DET}, protocol $TopKeyS^{\xi,\mathcal{M}}$ ensures correctness as defined by Definition 3.*

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. $Pr(\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l) = (\text{"}Failure\text{"}, \sigma_{\mathcal{A}})) > negl(l)$.

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. $Pr(\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l) = (\text{"}Failure\text{"}, \sigma_{\mathcal{A}})) < negl(l)$.

Theorem 3. ***[Guaranteed Key-Setup]****: Protocol TopKeyS, with predicates P^{FUL}, ensures the* guaranteed key-setup *property, as defined in Definition 5.*

Proof. First, we will prove that TopKeyS is bounded 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, TopKeyS is a Bounded Protocol.

We will now prove that TopKeyS with predicate P^{FUL} is always finished with key setup.

If $P^{FUL}(c, G, \Phi, \rho, \Re) = 1$, 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.

In the synchronous model we will prove also the Bounded Termination property.

Theorem 4. ***[Bounded Termination]****: Protocol TopKeyS ensures* Bounded Termination *property, as defined at Definition 6.*

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 TopKeyS design there are 6 synchronous transactions (can be seen in Fig. 3). Each transaction, as explained, is bounded by $|V|\,T_{delay}$.

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

Using TopKeyS, the server can set the next time the client will initiate a refresh key session with the server. A new refreshed key will be received only if the client was able to provide enough successful responses. Refreshing the key provides both forward secrecy and 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. ***[Forward Secrecy and Proactive Security]****: Protocol TopKeyS, with respect to predicate P^{DET}, ensures* forward secrecy *and* proactive security *properties, as defined in Section 3.*

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 TopKeyS 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 TopKeyS 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 no greater than n_A.

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

5 Adversarial Routing TopKeyS (AR-TopKeyS)

The TopKeyS protocol presented in Section 4 does not ensure correctness, guaranteed key-setup, forward secrecy, and proactive security 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 4.1. 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-TopKeyS, an enhancement of TopKeyS for networks with adversarial routing.

5.1 Protocol Design

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

AR-TopKeyS assumes a TTL-network as in Definition 1. 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.

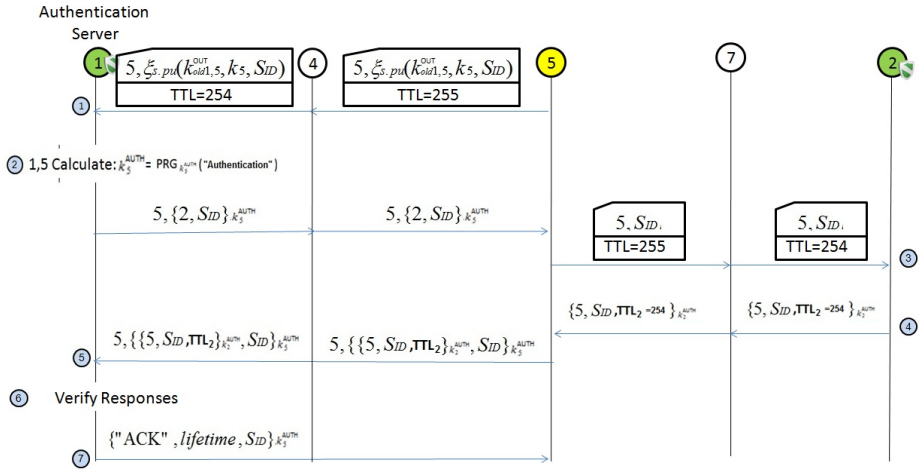

Fig. 4. AR-TopKeyS message sequence with the additional TTL field, based on the network in Figure 1.

The protocol messages and design are based on topology-based key setup, and the protocol activation and sequence are similar to topology-based key setup. 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-TopKeyS can be a *trusted device* or an *upgraded device*. The only requirement is that the collaborator executes the protocol and has a shared-secret key with the server.

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-TopKeyS challenge-response session is defined *successful*, if: (1) it is a successful session in TopKeyS as defined in Subsection 4.1, 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 TopKeyS.

5.2 Protocol Properties

In this section we present the topology conditions necessary to ensure that the AR-TopKeyS protocol has the properties defined in Section 3. Formal proofs are available in the full version of the paper [10].

Let T be the group of *trusted* devices, and let $c_1, ..., c_{N_c}$ be the group of collaborators (which can be *trusted* or *upgraded* devices).

We denote the group of non-trusted 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.*

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,

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

$$\implies N \leq 255 - ttl + 1$$

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

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 trusted and 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 6. ***[Correctness]****: For every CPA-secure public-key scheme ξ and MAC-secure scheme $\mathcal{M}$, with predicate $P^{DET-ROUTE}$, protocol $AR-TopKeyS^{\xi,\mathcal{M}}$ ensures correctness as defined by Definition 3.*

Since AR-TopKeyS is built on TopKeyS, it uses the same key-refresh mechanisms. Since the correctness property of TopKeyS was not ensured in adversarial-routing, the periodic key-setup could not be executed. Because AR-TopKeyS ensures correctness, the periodic key setup will ensure forward secrecy and proactive security simply by renewing the keys of devices after a period of time.

Theorem 7. ***[Forward Secrecy and Proactive Security]****: Protocol AR-TopKeyS, with respect to predicate $P^{DET-ROUTE}$, ensures* forward secrecy *and* proactive security *properties, as defined in Section 3.*

Notice that guaranteed key-setup is not ensured in adversarial routing, even with AR-TopKeyS. 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.

6 Conclusions

Previous work has proposed security solutions based on the user of topological properties of the network, instead of relying on complexity-based assumptions. These works had limited applications, probable since complexity-based cryptography is already widely deployed and not objected-to by most practitioners.

We also use topological properties to improve security, but with a different approach: the topology information allows us to *improve* deployability and/or security of the system.

Specifically, we present TopKeyS, a topology-based key setup protocol. TopKeyS uses known topology, with appropriate (redundancy) properties, to ease the deployment and improve the security of dey distribution and management. Deployment is 'plug and play' - no need to setup individual key for each device; and security is improved by ensuring recovery from periodical break-in into devices, allowing *all* devices to be sometimes-corrupted, as long as 'enough' devices are secure at any given (limited length) interval. This second property is referred to as *proactive security.*

Another area where previous work was lacking was consideration of different - and in particular, realistic - *routing models.* Our model and results support different forms of routing, including the (common in practice) 'shortest-path' routing, and the worst-case 'adversarial routing'. In adversarial routing, the attacker is able to control the route of messages in the network.

Using the formal model, we present and prove the security of TopKeyS, for different routing models. TopKeyS has the following properties: *secrecy, correctness, guaranteed key-setup, forward secrecy*, and *proactive security.*

To support adversarial routing, we also present an enhancement of TopKeyS, called AR-TopKeyS. AR-TopKeyS uses the TTL field of IP packets to preserve security.

References

1. 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.
2. 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.
3. Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.
4. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology—EUROCRYPT 2001*, pages 453–474. Springer, 2001.
5. Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.
6. Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.
7. 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.
8. Jokin Garay, Clint Givens, and Rafail Ostrovsky. Secure message transmission with small public discussion. *Information Theory, IEEE Transactions on*, 60(4):2373–2390, 2014.
9. Yossi Gilad and Amir Herzberg. Plug-and-play IP security. In *Computer Security–ESORICS 2013*, pages 255–272. Springer, 2013.
10. Amir Herzberg and Yehonatan Kfir. Topology-based plug-and-play key-setup ,https://eprint.iacr.org/2016/060.
11. Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC Press, 2014.
12. 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.
13. G Malkin. Rfc 2453: Rip version 2. *Request for Comments*, 2453, 1998.
14. Jon Postel. Rfc 791: Internet protocol, september 1981. *Darpa Internet Protocol Specification*, 1990.
15. 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.
16. K Srinathan, Arvind Narayanan, and C Pandu Rangan. Optimal perfectly secure message transmission. In *Advances in Cryptology–CRYPTO 2004*, pages 545–561. Springer, 2004.
17. 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.

18. 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.