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 present a *topology-based key setup protocol (ToBKeS)* to facilitate the *plug and play* deployment of cryptography, in networks with known topology. This protocol uses the topology to authenticate messages of devices.
ToBKeS assumes that there is at least one device that is initialized with the known network topology, *the authentication server*, that it has a known public key and that it shares secret keys with some of the other devices in the network.
ToBKeS eases the adoption of security by eliminating the need to manually set every device with its own private key. Furthermore, ToBKeS limits the impact of key exposures by ensuring both *perfect forward secrecy* and *proactive key refresh*, re-establishing security after exposure.
We analyze the properties of the ToBKeS protocol and show sufficient topology conditions for its applicability. In addition, we prove its security against power-full attacker, that is able to control the route of the network, as well as an attacker that is able control some of the devices in the network.

1 Introduction

The use of the topology for sending secret message have already been discuss in the past. At [4] the researchers uses different routes in the network for transmitting secret messages. Assuming such routes, they define and prove the security properties, against an attacker that controls multiple device on those routes.

However, there are some gaps in the model of [4]. First, in real networks, the exact message routing is not always known to both parties. In addition, those works did not related to the way for finding the different routes, nor did not related to the routing method the network may have. In addition, they neglected attackers that are able to change the route of messages. Hence, an extended topology model is required.

A significant challenge in cryptographic deployment is *key setup* in networks that construct from 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.

In addition, once deployed, if key exposure is suspected, the keys need to be manually re-initialized. This recovery process is another challenge in deployment of cryptography.

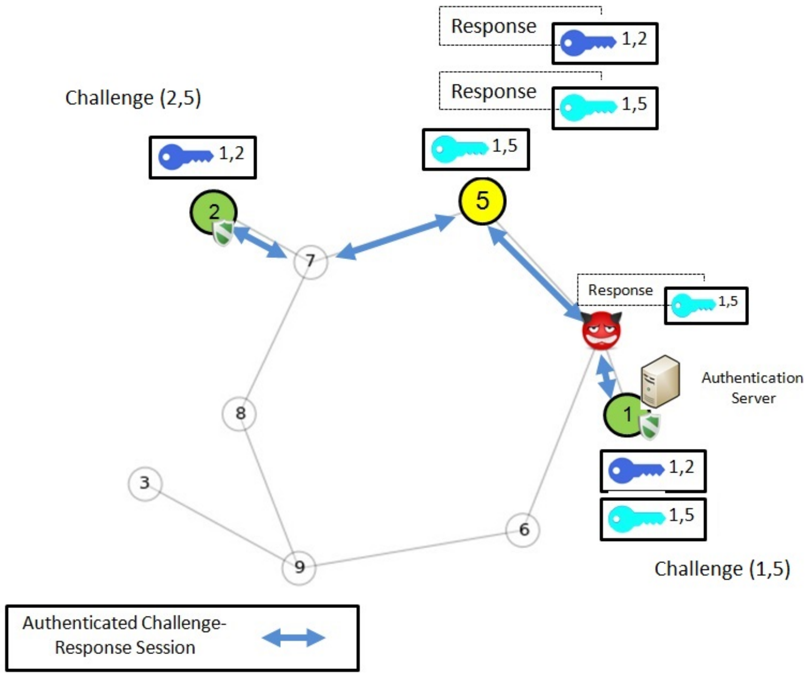


Fig. 1. The IEEE 9-bus model [10] of a small power network. The topology-based key setup protocol provides plug-and-play initial key setup and proactive key refresh. The protocol uses multiple authenticated challenge-response sessions, between a client (device 5) and several trusted devices (devices 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

We address these challenges by presenting a *topology-based key setup* scheme to facilitate the *plug and play* deployment of cryptography, with *proactive key refresh*.

Our topology-based scheme leverages the fact that the topology of several networks is usually known and quite stable, with considerable redundancy (e.g. for resilience). This facilitates *plug and play* deployment, which does not require manual setup of each device. Our scheme supports *proactive key refresh*, allowing a completely automated recovery of devices from private key exposures.

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 standard assumptions about internet routing. This also allows us to perform proactive key refresh, similar to [2], but without depending on attack-detection at the device.

At last, we prove the security of the protocol using the extended topology model we created.

For example, consider the network in Figure 1, which is based on the IEEE 9-bus model [10] - a known topology of power network. This system is built from 9 communication devices, each one of them represent a communication device of 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. As device 5 is upgraded, the server will ask it for evidence that it is connected with 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 formalizing the properties of networks that applicable to use topology-based key setup. In future works, we will evaluate the applicability of those assumptions in real world networks, such as ICS networks:
(1) Known topology: The network topology and the routing method are known.
(2) Trusted nodes: Several highly-secure and trusted nodes.
(3) Known public key: One or more of the trusted nodes 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 mostly use the Internet Protocol (IP), with intermediate devices acting as routers, in particular following the TTL (hop-count) rules [9], and usually with simple shortest-path routing without weights, e.g., provided by the RIP protocol [8].

Contributions. We make the following contributions:

(1) A topology-based plug-and-play initial key setup scheme for networks with known topologies.

(2) A topology-based proactive key refresh scheme, that does not require (manual) attack-detection capabilities.

(3) Proving the security of the scheme.

1.1 Related Work

Maybe to add more related works regarding the use of topology

The work at [11] presented how that method can be used to create a shared secret key between two devices in the network. However, they did not present a recovery mechanism nor a realistic routing model of the communication network. In addition, our work uses the topology for a short time, just for seting a key for cryptographic protocols. It does not rely only on the topology for long-term security.

Formulation of key setup protocols was presented by [1]. In that work, the researchers formulated the definition of protocols, and specifically key setup protocols. In addition, they define the execution process of such protocols, and the definitions of their security. We extend the model in [1] to include the network topology and routing method. Using our model, we will be able to define the properties of protocols that are using the network topology and routing method for key setup.

Our work is related to previous research of [5], as we share the same motivation for enabling zero-configuration key setup. The method in [5] assumed that there is an anonymous communication network between the devices that participates in the protocol. Our method is different as 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.

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 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 in the network. An example for 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 that there is a group of devices that do not support cryptographic modules. We call this group of devices, *legacy devices*, and we denote it by $L \subset V$.

Another group in the network, are the *upgraded devices* which support cryptographic modules and need to be initialized with keys. These devices also need to have a recovery plan to receive a new key in case of (suspected) key leakage.

In case the network uses the IP protocol, we model additional property. IP packets have a *TTL field* [9], which defines the number of hops the packet is allowed to pass. The maximal TTL value that can be set to a packet is 255; for simplicity, we assume all (non-corrupt) devices initialize the TTL, upon sending a packet, to 255. We define a *TTL-Network* as one where devices follow 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 may be *compromised*; both legacy and upgraded (but not trusted) devices 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 on 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 $\phi : V \times V \to \{Legacy, Upgraded, Trusted, Compromised\}$, 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 *Compromised*.

We consider an attacker $\mathcal{A}$, who controls all *Compromised devices*[1]. The attacker tries to disrupt key setup by an upgraded device, to register its own key for some device, or to learn the key setup in the server for some (upgraded) device.

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

The attacker is able to initiate, delay, block, or manipulate messages that pass through its devices. In addition, the attacker is able to eavesdrop on messages in the entire network, even 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, by setting a *route* in the message. 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.

[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.

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 $\Re_{\mathcal{A}}$. The adversarial routing is not necessarily the shortest path because the network routing is executed according to the adversarial routing function, $\Re_{\mathcal{A}}$.

Legacy and *Upgraded* devices always send messages according to the routing method: the routing list in source-routing networks; the $\Re$ function in shortest-path network; and $\Re_{\mathcal{A}}$ in adversarial routing network. 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 [1] for message-driven-protocol. For simplicity of our discussion, we gave a more specific definition for key-setup protocol, and for that, we extended their model to support several modes of the protocol on different parties.

A *topology-based key setup* protocol π is a message-driven-protocol [1] 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 $s.pr$. In addition, the server is 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 the 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 devices that have $P(v, G, \phi, \rho, \Re) = 1$ from all the devices in the network.

The server protocol is activated by a key-request message from a client c and 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, that will be discussed in Section 3.

3 Problem Formulation

Our problem formulation is based on the execution model by Bellare et al. [1], and 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.*

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).

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

1. $\mathcal{A}_1$ is choosing $k_0, k_1, S^0_{ID}, S^1_{ID} \leftarrow \{0,1\}^l$. With them, it creates two
   activation messages $m_0, m_1$ $s.t.$ $m_i = \{k_i, S^i_{ID}\}$.
2. A random bit is chosen, $b \overset{\$}{\leftarrow} \{0,1\}$
3. A successful execution, with activation message equal to $m_b$, is chosen
   randomly, $(''Success'', k_c^{OUT}, \sigma_{\mathcal{A}}) \overset{\$}{\leftarrow} \mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l)$ $where\ m_{init} = m_b$
4. Return 1 if $\mathcal{A}_1(k_c^{OUT}, \sigma_{\mathcal{A}}) = b$, and 0 otherwise.
```

Algorithm 1: Indistinguishability experiment

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

Definition 3 ***[Correctness]*** *Protocol π ensures Correctness, with respect to predicate P, if for all PPT attacker $\mathcal{A}$, there exists a negligible function negl s.t.:*

$Pr(\boldsymbol{EXEC}(\mathcal{A}, \pi, P, 1^l) = (''Failure'', \sigma_{\mathcal{A}})) < negl(l)$,
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 is scheduling 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 those 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 parameter 1^l and Eventually Delivering attacker $\mathcal{A}$:*
$\boldsymbol{EXEC}(\mathcal{A},\pi,P,1^l) = \{''Success'', k_c^{OUT}, \sigma_{\mathcal{A}}\}$.

3.2 Asynchronous Execution

For each of the defined properties, the attacker goal is to create an execution process that its output contradict one of the protocol requirements. In order 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 the attacker choose.

In addition to them, 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 messages routing $\Re = \Re_0$.

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

At 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, in addition to the routing method and the security parameter 1^l.

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

The activations are controlled and scheduled by the attacker. It also decides which incoming messages or external requests the activated party is to 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 source-route network, each message contains also the route of the message.

In order to send a random string *payload* to device d, a party s adds the message m to a set of pending messages M. The message next hop device will be 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, than a new message m' will be added to M (Algorithm 4). The next hop field of m' will be the neighbour 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 it 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.

In addition to activating parties, the adversary $\mathcal{A}$ can corrupt parties. Upon corruption $\mathcal{A}$ learns the entire current state of the corrupted party. In addition, 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.

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

Algorithm 2: Execution Process

```
**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

3.3 Synchronous Model

In the synchronous model, every sender is measuring the time passed from sending the message. As that time reached a timeout value, the sender will no longer wait for response for that message. All the parties in this model have synchronized clocks, which proceed in the same rate.

We denote a synchronous execution $\mathbf{EXEC^{SYN}} = \mathbf{EXEC^{SYN}}$ ($\mathcal{A}$,π,P,1^l,t_0)), were t_0 denote the starting time of the execution. A synchronous execution has all the outputs as the asynchronous model, with 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 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,

```
Procedure Add_Message($m$,$M$)
    Add to $M$ element with:
        $source = m.source$
        $destination = m.destination$
        $payload = m.payload$
        $ttl = m.ttl$
        if $\rho = source$ then
            $route = m.route$
            $next\_hop = m.route[m.source]$
        else
            $next\_hop = \Re(m.source, m.destination)$
        end
```
Algorithm 4: Message Handling

at each device, is zero. Thus, the maximal time that will be passed from sending a message till receiving response is $T_{max} = 2\,|V|\,T_{delay}$. 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.

Using the synchronous model we define the Bounded Termination property:

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.*

Protocol π ensures Bounded Termination, if there exists $T_{EXEC}(G)$ s.t. for every attacker $\mathcal{A}$, and the execution is finished after the 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 [3]. A key-setup protocol ensures Forward Secrecy if 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 compromises, and the recovery process is faster than the time it takes to the attacker to compromise devices. In that way, the protocol helps in resisting to an attacker that tries to increase the number of compromised devices.

In more formal words, 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 execute a *refreshment protocol* that change 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 (maximum of) n_A devices. While receiving control on a device, the attacker is able to retrieve this device key. This key information is known to the attacker, even after he release control from the device.

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

Protocol π ensures Proactive Security, if for every $n_A -$ node attacker 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 (TobKeS)

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

The goal of the ToBKeS 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 upgrading a device or to recover from (possible) compromise of the secret keys of the device.

The protocol uses a public key encryption scheme ξ and a MAC scheme $\mathcal{M}$, as defined at [7] . We denote a ToBKeS that uses these schemes as $ToBKeS^{\xi,\mathcal{M}}$. In cases we would like to emphasize that $\mathcal{M}$ is not relevant for the discussion, we will omit it.

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. By that, under several conditions that will be discussed, the key holder is authenticated.

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, $k_c^{OUT} = PRF_k(\text{"}Client\_Key\text{"})$.
k_c^{AUTH}	The key the will be used during topology-based key setup execution, to authenticate messages between the server and the client, $k_c^{AUTH} = PRF_k(\text{"}Authentication\text{"})$.
$\xi_{s.pu}\{\cdot\}$	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 $\times$ V $\rightarrow$ {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

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. In order to initiate the protocol, the client generates a random key $k \xleftarrow{\$} \{0,1\}^l$.

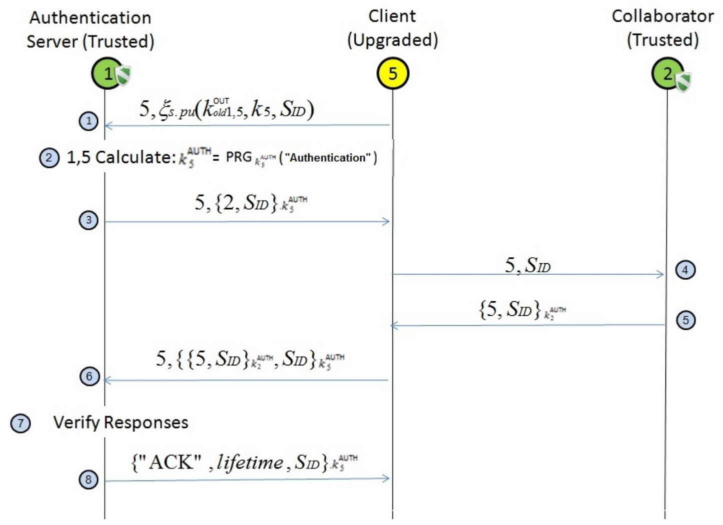

Authentication Server (Trusted)
Client (Upgraded)
Collaborator (Trusted)
1
5
2
1 $5, \xi_{s.pu}(k^{OUT}_{old1,5}, k_5, S_{ID})$
2 1,5 Calculate: k_5^{AUTH} = PRG $_{k_5^{AUTH}}$ ("Authentication")
3 $5, \{2, S_{ID}\}_{k_5^{AUTH}}$
4 $5, S_{ID}$
5 $\{5, S_{ID}\}_{k_2^{AUTH}}$
6 $5, \{\{5, S_{ID}\}_{k_2^{AUTH}},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

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.

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 $s.pu$. In addition, the client 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 claimed device location.

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

If there are no $2n_A+1$ such collaborators, then the server will simply selects 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 *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 number 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: Upon receiving one ACK message 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 Analysis

In this section we will prove that ToBKeS fulfill the requirement from section 2.

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. We prove in this section 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) $\rho \in \{source, shortest-path, adversarial\}$, (2) and v has $2n_A + 1$ disjoint routes to the security server. We prove in this section, that if P^{FULL} holds for device v, then v will always set up keys successfully.

Theorem 1. ***[Secrecy]:***
For every CPA-secure public-key scheme ξ protocol $ToBKeS^{\xi}$ ensures Secrecy, as defined at Definition 2.

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

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

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

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

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 $PubK^{cpa}_{\mathcal{A}_{\xi},\xi}(l)$ [7] :

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 $b \overset{\$}{\leftarrow} \{0,1\}^{l}$ 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 $|Pr\,[IND_{\mathcal{A},\mathcal{A}_1,\pi,P}(l) = 1] - \frac{1}{2}| \;<\; negl(l)$.

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

Theorem 2. ***[Correctness]:*** *For every CPA-secure public-key scheme ξ , and MAC-secure scheme $\mathcal{M}$, and with predicate P^{DET}, protocol $ToBKeS^{\xi,\mathcal{M}}$ 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 client c and the server.

According to protocol design at **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 to the client, and the client registers a key k_c^{OUT}. If not, than 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 at more than one route. This is contradiction to the disjoint routes assumption.

According to the protocol design at **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 exist a PPT attacker $\mathcal{A}$ s.t. $Pr(\mathbf{EXEC}(\mathcal{A}, \pi, P, 1^l) = ("Failure", \sigma_{\mathcal{A}})) > negl(l)$.

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

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

Assume (3): Then, the attacker was able to retrieve the key k_c^{OUT} from the activation message, and this is contradiction to the Secrecy property.

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

Theorem 3. ***[Guaranteed Key-Setup]****: Protocol ToBKeS , with predicates* P^{FUL}*, ensures* Guaranteed Key-Setup *property, as defined at Definition 5.*

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

We will now prove that ToBKeS 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 ToBKeS 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 ToBKeS 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: $6\,|V|\,T_{delay}$

Theorem 5. ***[Forward Secrecy and Proactive Security]****: Protocol ToBKeS, with respect to predicate P^{DET}, ensures* Forward Secrecy *and* Proactive Security *properties, as defined in Section 3.*

5 Adversarial Routing ToBKeS (AR-ToBKeS)

The ToBKeS that was presented in Section 4, does not ensure **Correctness** , **Forward Secrecy** and **Proactive Security** in networks with adversarial routing. The reason for that is that the authentication server does not know the adversarial routing, and hence, can not ensure disjoint routes.

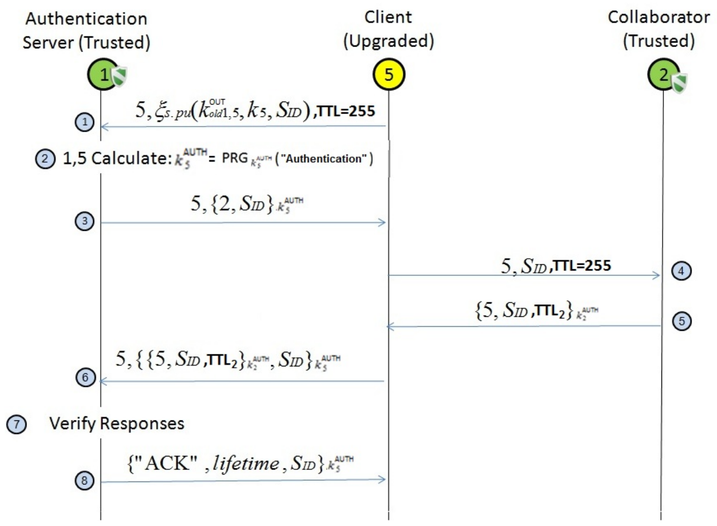
For example, consider an adversarial routing in the network from Figure 4.1. An attacker that controls device 9, will be able to authenticate as device 5, simply by routing the responses from devices 1 and 2, to pass through device 9. In that way, device 9 will be able to provide the needed authenticated responses, and to authenticate as device 5.

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

5.1 Protocol Design

The goal of AR-ToBKeS is to provide cryptographic keys to upgraded devices, without requiring manual installation. As we describe AR-ToBKeS ensures **Correctness** and **Proactive Security** in networks with adversarial routing.

AR-ToBKeS assumes a TTL-network as in Definition 1. The topology authentication is based on an authentication of the TTL field of the challenges that were received by the collaborators, and comparing them to their expected values according to 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.



Authentication Server (Trusted)
Client (Upgraded)
Collaborator (Trusted)
1
5
2
1 $5, \xi_{s.pu}(k^{OUT}_{old1,5}, k_5, S_{ID})$,TTL=255
2 1,5 Calculate: $k_5^{AUTH} = PRG_{k_5^{AUTH}}$("Authentication")
3 $5, \{2, S_{ID}\}_{k_5^{AUTH}}$
4 $5, S_{ID}$,TTL=255
5 $\{5, S_{ID}, TTL_2\}_{k_2^{AUTH}}$
6 $5, \{\{5, S_{ID}, TTL_2\}_{k_2^{AUTH}}, S_{ID}\}_{k_5^{AUTH}}$
7 Verify Responses
8 $\{$"ACK"$, lifetime, S_{ID}\}_{k_5^{AUTH}}$

Fig. 4. AR-ToBKeS message sequence, with the additional TTL field, for the network in Figure 1.

The protocol messages and design are based on topology-based key setup. The protocol activation and sequence is similar to topology-based key setup. In this section we describe only the differences.

First difference is that all the messages that are sent from the client, are sent with the maximal value of TTL, ttl_{MAX}. Those messages route through the network, to the collaborators or to the authentication server. During the routing, the TTL fields are decreased, according to the TTL-network assumption. We denote the TTL value of the message that reach to collaborator i by ttl_i.

A collaborator in AR-ToBKeS can be *trusted device* or even *upgraded device*. The only requirement is that the collaborator will execute the protocol, and have a shared-secret key with the server.

As a collaborator i receives a challenge, it check the value of the TTL field, ttl_i. The collaborator responses with an authenticated messages that contains the TTL value of the received message, in addition to the client identification and the session ID S_{ID}). The response is authenticated with the collaborator shared key with the authentication server.

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

A AR-ToBKeS challenge-response session is defined *successful*, if (1) it is a successfull session in ToBKeS as defined at Subsection 4.1; and (2) the ttl-fields in all the authenticated responses 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 control of the attacker n_A. The behavior is similar to ToBKeS.

5.2 Protocol Properties

In this section we present the topology conditions necessary to ensure that the AR-ToBKeS protocol has the properties defined in Section 3. Formal proofs are available in the full version of the paper [6].

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 located at distance smaller 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 TTL of ttl_i, indicates that the sender of the challenge (the client device or a compromised device) is not far than $255 - ttl_i + 1$ hops from c_i.*

Proof. Every device in the network that processes the packet must decrease the TTL that was sent by the sender by amount of 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 is the distance of the sender in hop count.

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 or less than the distance between the client and the collaborator.

Hence,for a 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 $n_A - node$ attacker, we define the following topology-predicates:

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 authenticate 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 $\mathcal{M}$, and with predicate $P^{DET-ROUTE}$, protocol $AR - ToBKeS^{\xi,\mathcal{M}}$ ensures Correctness as defined by Definition 3.*

Since AR-ToBKeS is build on ToBKeS, it uses the same key-refresh mechanisms. However, the Correctness property of ToBKeS was not ensured in adversarial-routing. Hence, the periodic key-setup could not be executed. Since AR-ToBKeS 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-ToBKeS, with respect to predicate $P^{DET-ROUTE}$, ensures* Forward Secrecy *and* Proactive Security *properties, as defined in Section 3.*

work that processes the packet must decrease the TTL that was sent by the sender by amount of 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 is the distance of the sender in hop count.

6 Conclusions

We present a model for topology-based key setup. The model extend previous works that use topology for sending secret messages in several ways: (1) adding a formal model for shortest-path routing networks, as well as source-route networks; (2) defining security properties against an attacker that is able to control the route of messages in the network.

Using the formal model, we present and prove the security of ToBKeS; a novel method for plug-and-play key setup in networks with known topology. The method is based on authenticating the sender location, using authenticated challenge-response sessions. We discussed that ToBKeS has the following properties: Secrecy, Correctness, Guaranteed Key-Setup, Bounded Termination, Forward Secrecy, and Proactive Security.

At last, we present an extension for ToBKeS that uses the ttl-field of IP packets, for preserving the security of ToBKeS in the presence of attacker that is able to control the route of messages in the network.

References

1. 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.
2. Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.
3. Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.
4. Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM (JACM)*, 40(1):17–47, 1993.
5. Yossi Gilad and Amir Herzberg. Plug-and-play IP security. In *Computer Security–ESORICS 2013*, pages 255–272. Springer, 2013.
6. Amir Herzberg and Yehonatan Kfir. Technical report 15-02, bar ilan university - department of computer science,http://u.cs.biu.ac.il/ herzbea/security/15-02-tbks.
7. Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC Press, 2014.
8. G Malkin. Rfc 2453: Rip version 2. *Request for Comments*, 2453, 1998.
9. Jon Postel. Rfc 791: Internet protocol, september 1981. *Darpa Internet Protocol Specification*, 1990.
10. 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.
11. 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.