Secure Communication Topologies in the Internet of Things

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master Thesis

Secure Communication Topologies in the Internet of Things

Danny Schweizer

Supervisor: Dr. Saša Radomirović
Professor: Prof. Dr. David Basin
Issue date: September 19, 2014
Submission date: March 19, 2015

Abstract

Secure communication between smart devices is a key issue in the Internet of Things. In this thesis, we extend the HISP communication topology model with novel channel properties and introduce two types of channel restrictions in order to better model the interaction between humans and devices in the Internet of Things. We then analyze settings involving a human H, a smart device D and a smart phone P and see for which topologies there is a protocol that establishes a shared key between D and P and for which there is none. We then analyze the security of several real-world products from the realm of the Internet of Things.

Furthermore, we present an algorithm that works on general topologies consisting of an arbitrary number of devices and constructs protocols that provide authentic, confidential and secure channels between two designated nodes A and B in the network, if such protocols exist.

Acknowledgments

First and foremost, I would like to express my gratitude to Prof. Dr. David Basin for giving me the opportunity to write this thesis and to work on such an interesting topic.

Moreover, I would like to thank my supervisor Dr. Saša Radomirović for his continuous support and valuable feedback while writing this thesis.

Finally, special thanks to my family and friends for supporting me during my entire studies.

Contents

Chapter 1

Introduction

The emergence of the Internet of Things gives rise to many novel security issues. For example, if smart devices in a connected home are not secured well enough, an attacker might get access to the building or cause a blackout by remotely turning off smart light bulbs. To secure communication between smart devices is a very critical issue and should not be overlooked.

In a typical setting in a connected home, a human wants to interact with smart things such as light bulbs or thermostats through their physical interfaces. We model the human as well as the devices as communicating agents. However, there are restrictions in the communication between a human and a device in comparison to the communication between two devices. First, the human is restricted in the sense that he cannot perform any cryptographic operations on his own and second, the communication channels between the human and the device are restricted by the type of interface the device offers. A small device might only have a button or an LED instead of a keypad or a display. The amount of information which can feasibly be transmitted between the human and the device therefore depends on what the device looks like.

In this thesis, we extend the communication topology model proposed in [1], which already allows for the analysis of communication protocols including humans. The verification of security properties of protocols in this model is supported by the Tamarin prover [11]. We extend the model in order to support new kinds of channels as well as restrictions, more closely modeling the real-world channels between the human and a device, such as buttons or LEDs.

We then consider all realistic topologies involving a human who wants to secure the communication between a smart thing and his smart phone and analyze the security of these topologies. Afterwards, we give some real-world examples and analyze the security of their protocols.

In the penultimate chapter of the thesis, we address possibility and impossibility of secure communication in a general communication topology. More precisely, given a topology consisting of an arbitrary number of nodes, connected by channels with different security properties, we present an algorithm that computes the strongest channel (among insecure, authentic, confidential and secure

channels) that a protocol could provide between two designated nodes A and B as well as a protocol actually providing it.

The thesis is structured as follows. We introduce the basic security protocol model in Chapter 2 and present our extensions in Chapter 3. We introduce the communication topology model in Chapter 4 and analyze Internet of Things topologies by providing protocols or proving non-existence of them in Chapter 5. In Chapter 6, we analyze some examples of real-world products from the realm of the Internet of Things. We present impossibility and possibility results in general topologies in Chapter 7. A final conclusion is then drawn in Chapter 8.

Chapter 2

Security Protocol Model

The model in which we describe our protocols is based on the security protocol model discussed in [1] which is itself based on the model of Tamarin [11], a protocol verification tool. We describe the basic elements of the model before introducing some extensions.

The term algebra of the Tamarin model is denoted by $\mathcal{T}$ and its underlying signature by Σ. The signature contains functions $\langle \_, \_ \rangle$ for pairing terms, $\pi_1(\_)$ and $\pi_2(\_)$ for the first and second element of a pair, $senc(\_, \_)$ and $sdec(\_, \_)$ for symmetric encryption and decryption, $aenc(\_, \_)$ and $adec(\_, \_)$ for asymmetric encryption and decryption, $sign(\_, \_)$ for signing a message, $verify(\_, \_, \_)$ for signature verification, $h(\_)$ for hashing and $pk(\_)$ for the public key corresponding to a certain secret key. Furthermore, Σ also contains a countably infinite amount of fresh and public constants, denoted as two disjoint sets $\mathcal{C}_{\mathsf{fresh}}$ and $\mathcal{C}_{\mathsf{pub}}$.

Instead of $senc(m, k)$ or $aenc(m, k)$, we will also write $\{m\}_k$ if the type of encryption is clear or irrelevant.

The following equations define the equational theory of these functions. For pairing and projection, we have $\pi_1(\langle a, b \rangle) = a$ and $\pi_2(\langle a, b \rangle) = b$. For symmetric encryption, we have $sdec(senc(a, b), b) = a$ and for asymmetric encryption $adec(aenc(a, pk(b)), b) = a$. For signing, we have $verify(sign(a, b), a, pk(b)) = true$, where true is a constant function. The verification function therefore takes a signature, a message and a verification key and outputs a binary value, indicating whether the verification succeeded or not.

2.1 Rewriting System

The core of the Tamarin model is the labeled multiset term rewriting system. Such a system consists of a set of rules which change the state of the system. A system state is a finite multiset of facts, where facts are functions over $\mathcal{T}$ with symbols in a signature Σ_{Fact}. There are linear and persistent fact symbols. Linear facts are resources that can be consumed only once, while persistent facts stay in the system state. To be able to differentiate between the two, persistent facts are prefixed with a '!'. The initial state is the empty multiset.

The rules which are responsible for state transitions are denoted by $[l] \xrightarrow{a} [r]$, where l, a and r are sequences of facts. The facts in l, a and r are called *premises*, *actions* and *conclusions* respectively. The facts in l need to be in the state such that the rule can be applied. The rule deletes all *linear* facts in l from the state and replaces them with the facts in r. The facts in a label the rule.

An important concept in such a rewriting system is the concept of a trace. A trace describes which ground instances of rules were applied in which order during the protocol execution. It is defined as a sequence of sets of ground facts (facts containing no variables). The ground facts correspond to the actions in a which label the rules. Formally, a trace tr is an element of $\mathcal{P}(\mathcal{G})^*$, where $\mathcal{G}$ is the set of all ground facts.

A protocol $\mathcal{R}$ is a set of rules, which is split into two kinds of rules: First, there are model rules which include rules modeling the capabilities of the adversary as well as the channel rules described below. Second, we have protocol specification rules which specify a certain protocol. The model rules stay the same for every protocol we analyze. We denote the set of all traces of $\mathcal{R}$ by $TR(\mathcal{R})$.

2.2 Channel Rules

One of the extensions of the Tamarin model described in [1] are channel rules, which model the communication between agents over insecure, authentic, confidential and secure channels. There are two facts in Tamarin which concern the adversary's knowledge. Out facts are used to model transmitted messages learned by the adversary and In facts are used to model the injection of messages from the adversary's knowledge.

Note that we assume that all agents are honest. This is in contrast to [1], where agents might be dishonest and leak information to the adversary by additional 'dishonest agent rules'.

The channel rules for insecure channels are as follows:

$$[\mathsf{Snd_I}(A, B, m)] \xrightarrow{\mathsf{Snd_I}(A,B,m)} [\mathsf{Out}(\langle A, B, m\rangle)] \tag{2.1}$$

$$[\mathsf{In}(\langle A, B, m\rangle)] \xrightarrow{\mathsf{Rcv_I}(A,B,m)} [\mathsf{Rcv_I}(A, B, m)] \tag{2.2}$$

The first rule represents the sending of a message m from A to B over an insecure channel. The $\mathsf{Snd_I}$ fact is inserted into the system state by some protocol specification rule. When performing the transition, an Out fact is produced, representing the adversary learning the message. The second rule models the receiving by B. The In fact models that the adversary can freely inject any message and change the sender and receiver arbitrarily.

The channel rules for authentic channels are:

$$[\mathsf{Snd_A}(A,B,m)] \xrightarrow{\mathsf{Snd_A}(A,B,m)} [!\mathsf{Auth}(A,m), \mathsf{Out}(\langle A,B,m\rangle)] \tag{2.3}$$

$$[!\mathsf{Auth}(A,m), \mathsf{In}(B)] \xrightarrow{\mathsf{Rcv_A}(A,B,m)} [\mathsf{Rcv_A}(A,B,m)] \tag{2.4}$$

The difference compared to the insecure channel is that here, a persistent !Auth fact is created in the send rule which is then used in the receive rule. This fact makes sure that the adversary can change neither the sender nor the message. Only the receiver can be freely chosen. Furthermore, the fact is persistent in order to model the ability to replay the message. Like in the insecure setting, the message is given to the adversary, which reflects the fact that the channel is not confidential.

The next rules model confidential channels:

$$[\mathsf{Snd_C}(A,B,m)] \xrightarrow{\mathsf{Snd_C}(A,B,m)} [!\mathsf{Conf}(B,m)] \tag{2.5}$$

$$[!\mathsf{Conf}(B,m), \mathsf{In}(A)] \xrightarrow{\mathsf{Rcv_C}(A,B,m)} [\mathsf{Rcv_C}(A,B,m)] \tag{2.6}$$

$$[\mathsf{In}(\langle A,B,m\rangle)] \to [\mathsf{Rcv_C}(A,B,m)] \tag{2.7}$$

The first two rules are analogous to the rules for authentic channels. The difference is that here, the adversary can only change the sender, but not the receiver or the message. The !Conf fact is persistent, which again means that the adversary can replay messages. Also, the adversary does not get the message anymore (which reflects the confidentiality of the channel), however, there is a third rule which allows him to send anything from his knowledge over this channel, which is possible, since the channel is not authentic.

For secure channels, the rules are as follows:

$$[\mathsf{Snd_S}(A,B,m)] \xrightarrow{\mathsf{Snd_S}(A,B,m)} [!\mathsf{Sec}(A,B,m)] \tag{2.8}$$

$$[!\mathsf{Sec}(A,B,m)] \xrightarrow{\mathsf{Rcv_S}(A,B,m)} [\mathsf{Rcv_S}(A,B,m)] \tag{2.9}$$

The secure channel combines the security guarantees of the authentic and confidential channels: The adversary cannot change anything, he can just replay the message an arbitrary number of times.

2.3 Channels as Goals

A goal of this thesis is to find protocols in communication topologies which provide channels with security properties between two nodes, such as confidentiality of some secret. In this chapter, we define what 'providing' means more formally. The definitions are taken from [1] and [13], where more details can be found. We introduce two channel properties, namely non-injective agreement and confidentiality. In general, channel properties are defined as a pair of predicates (p, q), where p is an existential requirement, making sure that it is actually possible to transmit some data we want to be confidential or agreed upon by

two agents, and q is a universal requirement such as the non-injective agreement or confidentiality itself, which needs to be satisfied for all traces, agents and messages.

Both p and q have domain $\mathcal{P}(\mathcal{G})^* \times \mathcal{C}_{\mathsf{pub}} \times \mathcal{C}_{\mathsf{pub}} \times \mathcal{M}$. A protocol $\mathcal{R}$ *provides* a channel with the property (p, q) if there exists a trace, two (honest) agents and a message such that p is satisfied and if for all traces, agents and messages, q is satisfied. More formally, $\mathcal{R}$ provides a channel with property (p, q) if

$$\exists tr \in TR(\mathcal{R}), S, R \in Honest(tr), m \in \mathcal{M} : p(tr, S, R, m) \wedge$$
$$\forall tr \in TR(\mathcal{R}), S, R \in Honest(tr), m \in \mathcal{M} : q(tr, S, R, m).$$

Note that $Honest(tr)$ denotes the set of all honest agents in trace tr. In our case, these are *all* agents, since we assume that no agent is dishonest.

A property introduced in [7] is non-injective agreement (NIA), which is a kind of authenticity property. It requires that if an agent A in role S has run a protocol, apparently with agent B in role R, then B has also run the protocol (in role R), apparently with A and both A and B agree on some data m. In [13], a definition of the NIA property is introduced as follows.

We use two new facts, namely $\mathsf{Commit}(S, R, m)$, which denotes that S believes to share m with R and $\mathsf{Running}(S, R, m)$, which expresses that R believes that he has provided m to S. We want to make sure that for every Commit action, there previously was a corresponding $\mathsf{Running}$ action on the same m and on the same agents. Formally, this is defined as $(p_{\mathrm{NIA}}, q_{\mathrm{NIA}})$, where

$$p_{\mathrm{NIA}}(tr, S, R, m) \coloneqq \mathsf{Commit}(S, R, m) \in tr$$
$$q_{\mathrm{NIA}}(tr, S, R, m) \coloneqq \mathsf{Commit}(S, R, m) \in tr \rightarrow \exists tr', tr'' \in \mathcal{P}(\mathcal{G})^* : tr = tr' \cdot tr'' \wedge$$
$$\mathsf{Running}(S, R, m) \in tr' \wedge \mathsf{Commit}(S, R, m) \in tr''.$$

The confidentiality property states that the adversary should not learn a certain message. If we want to state that a message m needs to stay confidential, we add the action $\mathsf{Secret}(S, R, m)$ to a protocol rule. The confidentiality property is defined as $(p_{\mathrm{conf}}, q_{\mathrm{conf}})$, where

$$p_{\mathrm{conf}}(tr, S, R, m) \coloneqq \mathsf{Secret}(S, R, m) \in tr \wedge \exists tr', tr'' \in \mathcal{P}(\mathcal{G})^* : tr = tr' \cdot tr'' \wedge$$
$$\mathsf{Running}(S, R, m) \in tr' \wedge \mathsf{Commit}(S, R, m) \in tr''$$
$$q_{\mathrm{conf}}(tr, S, R, m) \coloneqq \mathsf{Secret}(S, R, m) \in tr \rightarrow !\mathsf{K}(m) \notin tr.$$

The $!\mathsf{K}$ fact models the knowledge of the adversary, where $!\mathsf{K}(m)$ means that the adversary knows the message m. The conditions in the existential requirement make sure that there is a trace where a secret message m is communicated from S to R, which means that confidentiality is not trivially satisfied by never sending anything.

2.4 Protocol Specification

As described above, there are two kinds of rules: Model rules, which are the same rules for all protocols and protocol specification rules, which specify a certain protocol. We already discussed model rules, so let us have a look at protocol specification rules. Such rules are either setup rules or rules describing the behavior of roles in the system. The setup rules initialize the roles and create their initial $\mathsf{AgentState}(A, c, n)$ facts. Such a fact contains the name A of the agent, a role step counter c and the agent's knowledge n. In the rules describing the behavior of roles, these facts are then used to model the order in which a certain agent executes the role steps.

The rules defining the behavior of a role have to contain a single AgentState fact and may contain Fresh and receive facts in its premise. In its conclusion, it may contain send and AgentState facts (all such rules in all protocols described in this paper contain a single AgentState fact in the conclusion). In all occurrences of AgentState in a single rule, the same agent name has to appear. As an example, we might have the following rule:

$$[\mathsf{Fresh}(A, m), \mathsf{AgentState}(A, \text{`}A_0\text{'}, \langle B, k\rangle)]$$
$$\rightarrow [\mathsf{Snd}_\mathsf{I}(A, B, senc(m, k)), \mathsf{AgentState}(A, \text{`}A_1\text{'}, \langle B, k, m\rangle)]$$

In words, this rule describes a step where the agent A is in step 'A_0', knows a key k and an agent B, generates a fresh value m and sends it encrypted with k to B. After this step, A is now in step 'A_1' and has additionally gained knowledge of m.

More details about the security protocol model can be found in [1].

Chapter 3

Model Extensions

We are now extending the security protocol model from [1] with new types of channels as well as different restrictions.

3.1 Replay-Free and Invariant Channels

The first extension are two new types of channels, namely replay-free channels and invariant channels.

3.1.1 Replay-Free Channels

If we are considering the case of a physical button or a keypad, the adversary cannot replay any button or key presses. The channel between the human and the device is therefore replay-free. The replay-free channel rules are introduced in [13] and basically work as follows. We can transform authentic and confidential channels into replay-free channels by declaring the !Auth and !Conf facts to be linear instead of persistent. We are mostly interested in secure replay-free channels though.

A secure replay-free channel can be modeled by the following rule:

$$[\mathsf{Snd}_{\mathsf{SR}}(A, B, m)] \xrightarrow{\mathsf{Snd}_{\mathsf{SR}}(A,B,m)} [\mathsf{Rcv}_{\mathsf{SR}}(A, B, m)] \tag{3.1}$$

Note that the two send and receive rules are merged into one rule here. We can achieve this because the only way the linear Sec fact on the right side of the send rule can be used is in the corresponding receive rule. Furthermore, it can only be used at most once. The adversary could only block the transmission, which he can also do in the merged rule. Therefore, one rule is enough.

3.1.2 Invariant Channels

In order to better model cases such as buttons or keypads, we are going to introduce a new type of channel. When a human types something into a keypad or presses a button, it does not make sense to talk about changing the sender or receiver of e.g. a button press. The adversary is not able to intercept button presses and relay them to a different device, because we assume that all devices

are honest and therefore not corrupted by the adversary. Under this assumption, a button or a keypad should not be modeled as an insecure, authentic or confidential channel, because in each of them, an adversary can possibly change sender or receiver. However, using a secure channel would mean that the adversary does not see the button press or keypad presses at all. This assumption might be too strong if the adversary can physically observe what the human is typing. The same principle also applies to LEDs on a device which could send information to the human. There, it is also not possible to alter the sender or the receiver, but the adversary might still see the LED. Therefore, we need a channel where the sender, the receiver and the message cannot be changed, but the message itself is still given to the adversary. Furthermore, the channel is obviously replay-free.

Because nothing can be changed by the adversary (essentially, the adversary can only act passively and not actively), we call the channels *invariant channels*. Note that this notion of invariance is different from the notion of sender and receiver invariance used to denote a channel where one agent knows that the other agent is the same one for each message sent or received, but does not know the true identity of the other agent.

The rule for invariant channels looks like this:

$$[\mathsf{Snd}_{\mathsf{Inv}}(A,B,m)] \xrightarrow{\mathsf{Snd}_{\mathsf{Inv}}(A,B,m)} [\mathsf{Rcv}_{\mathsf{Inv}}(A,B,m), \mathsf{Out}(\langle A,B,m\rangle)] \tag{3.2}$$

The channel therefore behaves like a replay-free secure channel except that it outputs the messages on the channel to the adversary. It is easy to see that an invariant channel is stronger than an authentic channel but weaker than a secure channel. It is incomparable to a confidential channel, since the transmission of the message m is authentic using an invariant channel (which is not the case for a confidential channel), but the message is not confidential using an invariant channel.

3.2 Restrictions

We now want to extend the set of assumed channels by restricted channels, which are channels that are restricted in the type of messages they can transmit. In this section, we will go over different types of restrictions.

The reason why we want restricted channels is that in the topologies we want to analyze later, we have a human interacting with physical devices such as smart things. Note that we do not consider protocols where the human can type an arbitrary message into a smart thing, because it would be too inconvenient for the human to actually type a large value such as a cryptographic key into a device and also, a manufacturer might want the smart thing to have only one or two buttons instead of a keypad in order to keep it simpler and cheaper. So, overall, the channels involving the human are all somehow restricted in the types of messages that can be sent through them.

Now, obviously, many questions come up. How many different levels of restrictions do we need? Is the restriction of having one button the same as the restriction of having two buttons? What if the device has a keypad, where the human could type in some number with a few digits? There is no definite answer, but we want to analyze the situation and justify the abstraction we are going to use.

Our first observation is that it does not really matter whether we have one or two buttons: If we have only one button, we can just simulate the second button by considering for example a long button press to be a button press on the second button. In the other direction (from the device to the human), the same thing applies to whether the device has one or two LEDs. One level of restriction is therefore going to be 'a small number of buttons or LEDs'.

Now, what if the device has a display or a keypad where the human could read or type a number of several digits, e.g. a short password or a PIN? Strictly speaking, we could even transmit a password with only one button, namely by transforming it into a bitstring and sending the bits one by one (for example, a short button press could encode a 0, while a long button press could encode a 1). But obviously, in a realistic setting, this would be infeasible, so we assume that a button or an LED can only send a few different messages, but not something from a bigger space such as a PIN or a password.

Thus, we can naturally distinguish between two kinds of restricted channels. The first kind are buttons and LEDs which can only transmit messages from a very small message space and the second kind are keypads and displays which can transmit messages of a bigger space, e.g. the space of all 10-digit numbers. Note that this distinction is a bit fuzzy and there is no definite answer to what the highest possible amount of information a button can feasibly transmit is. However, in the protocols we are later going to describe, a button press or an LED lighting up is always used as just a confirmation with a message space of only one possible message.

Therefore, we consider only two kinds of channel restrictions in this thesis: Channels with small message spaces (which we call 'small channels') and channels where the human can send or read numbers and passwords (which we call 'password channels').

3.2.1 Small Channels

Let us first look at small channels. This type of restriction can be used in combination with all types of channels seen so far. Because there might be several small channels (which may be restricted to different messages) between two nodes, we need to include an identifier of the channel itself into the facts.

The channel rules for small channels are adapted versions of the channel rules for unrestricted channels. The justification for the use of the Out, In, !Auth, !Conf and !Sec facts is the same.

The channel rules for insecure small channels are:

$$[\mathsf{Snd_{IS}}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Snd_{IS}}(A,B,c,m)} [\mathsf{Out}(\langle A,B,c,m\rangle)] \quad (3.3)$$

$$[\mathsf{In}(\langle A,B,c,m\rangle), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Rcv_{IS}}(A,B,c,m)} [\mathsf{Rcv_{IS}}(A,B,c,m)] \quad (3.4)$$

The variable c denotes the unique public name of the small channel. The $!\mathsf{Possible}(A,B,c,m)$ fact tells us that it is possible to send message m from role A to role B via channel c. These facts are created in the setup rules of the protocol and make sure that we can only use these rules if the channel actually allows the transmission of the message m.

The rules for authentic small channels are:

$$\begin{aligned}&[\mathsf{Snd_{AS}}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)]\\ &\xrightarrow{\mathsf{Snd_{AS}}(A,B,c,m)} [!\mathsf{Auth}(A,m), \mathsf{Out}(\langle A,B,c,m\rangle)] \quad (3.5)\\ &[!\mathsf{Auth}(A,m), \mathsf{In}(\langle B,c\rangle), !\mathsf{Possible}(A,B,c,m)]\\ &\xrightarrow{\mathsf{Rcv_{AS}}(A,B,c,m)} [\mathsf{Rcv_{AS}}(A,B,c,m)] \quad (3.6)\end{aligned}$$

The rules for confidential small channels are:

$$[\mathsf{Snd_{CS}}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Snd_{CS}}(A,B,c,m)} [!\mathsf{Conf}(B,m)] \quad (3.7)$$

$$[!\mathsf{Conf}(B,m), \mathsf{In}(\langle A,c\rangle), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Rcv_{CS}}(A,B,c,m)} [\mathsf{Rcv_{CS}}(A,B,c,m)] \quad (3.8)$$

$$[\mathsf{In}(\langle A,B,c,m\rangle), !\mathsf{Possible}(A,B,c,m)] \to [\mathsf{Rcv_{CS}}(A,B,c,m)] \quad (3.9)$$

There is still an issue regarding Rule 3.8. Imagine a setting where there are three channels c_1, c_2 and c_3 between nodes A and B. The channel c_1 allows the sending of messages x and y, the channel c_2 allows x and z and the channel c_3 allows y and z. If A sent a message confidentially to B, the adversary should not be able to see which of the messages was sent. However, since he can choose the channel in Rule 3.8 freely, he can try to replay the confidential message over all three channels. Because for each of the three messages x, y and z there is a unique channel where the sending of the message fails, the adversary can find out which of the three messages was sent. One way to avoid this issue is by not allowing different restricted channels to have overlapping message spaces, except for the case where the message spaces are exactly equal. So, for every pair of two small channels, either they have distinct or the same message spaces.

The rules for secure small channels are:

$$[\mathsf{Snd_{SS}}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Snd_{SS}}(A,B,c,m)} [!\mathsf{Sec_S}(A,B,c,m)] \quad (3.10)$$

$$[!\mathsf{Sec_S}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Rcv_{SS}}(A,B,c,m)} [\mathsf{Rcv_{SS}}(A,B,c,m)] \quad (3.11)$$

We use the name $!\mathsf{Sec}_\mathsf{S}$ instead of $!\mathsf{Sec}$, because $!\mathsf{Sec}$ is already defined to have three parameters instead of four.

We can also define replay-free small channels. For example, a secure replay-free small channel looks as follows:

$$[\mathsf{Snd}_\mathsf{SRS}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)] \xrightarrow{\mathsf{Snd}_\mathsf{SRS}(A,B,c,m)} [\mathsf{Rcv}_\mathsf{SRS}(A,B,c,m)] \tag{3.12}$$

Finally, the rule for invariant small channels is:

$$\begin{aligned}&[\mathsf{Snd}_\mathsf{InvS}(A,B,c,m), !\mathsf{Possible}(A,B,c,m)]\\ &\xrightarrow{\mathsf{Snd}_\mathsf{InvS}(A,B,c,m)} [\mathsf{Rcv}_\mathsf{InvS}(A,B,c,m), \mathsf{Out}(\langle A,B,c,m\rangle)]\end{aligned} \tag{3.13}$$

All these rules are analogous to their unrestricted counterparts. Basically, the only difference is that they can only be used if m can actually be transmitted over channel c. Also, the channel name c is included in the send and receive facts.

Justification

To justify the rules, we are going to look at an example. Imagine a smart device with two buttons *ok* and *cancel* which can be pressed by a human. Each of these two buttons can be seen as a small channel from the human to the device. The setup rules might then insert $!\mathsf{Possible}(A,B,c_1,\text{`}ok\text{'})$ and $!\mathsf{Possible}(A,B,c_2,\text{`}cancel\text{'})$ into the system state, which reflects the fact that we can send the message *'ok'* over the first button (whose channel we are calling c_1) and the message *'cancel'* over the second button (whose channel we are calling c_2). The channels can then only be used for transmitting *'ok'* and *'cancel'* respectively. Alternatively, we could also consider the buttons to be a single channel c and insert $!\mathsf{Possible}(A,B,c,\text{`}ok\text{'})$ and $!\mathsf{Possible}(A,B,c,\text{`}cancel\text{'})$ into the system state. Now, we have a single channel c which can be used to send the two messages *'ok'* and *'cancel'*.

3.2.2 Password Channels

Password channels are channels where some message of small entropy such as a password or a number consisting of a few digits can be sent through. However, it is not feasible to send a cryptographic key over such a channel.

Since our model is only symbolic, it cannot distinguish between terms of low and high entropy. Therefore, we need to model terms with low entropy with a special function pw wrapped around them. To model a password, a term such as $pw(m)$ is therefore used. Password channels are channels that only transmit terms of this form. For example, the rule for invariant password channels is as follows:

$$\begin{aligned}&[\mathsf{Snd}_\mathsf{InvPw}(A,B,pw(m))]\\ &\xrightarrow{\mathsf{Snd}_\mathsf{InvPw}(A,B,pw(m))} [\mathsf{Rcv}_\mathsf{InvPw}(A,B,pw(m)), \mathsf{Out}(\langle A,B,pw(m)\rangle)]\end{aligned} \tag{3.14}$$

For channels with no confidentiality guarantee (such as insecure, authentic and invariant channels), such a single rule is enough. The password is leaked anyway and therefore, it cannot be used as a cryptographic key.

There is still a question remaining: How do we make sure that such a password cannot be used as a cryptographic key if it is sent over a confidential or secure channel? What we want to model is the fact that after enough time, the adversary will have checked all possible passwords with brute-force and could decrypt what was encrypted with the password. More precisely, we want the adversary to be able to guess the password after all runs associated to this password have stopped.

We will now show how we define the rules in our model. We use secure replay-free password channels, because they will be used in our topologies in the end, but it also works with confidential or regular secure channels.

First of all, to each $\mathsf{AgentState}$ fact, we add another argument, namely a list of identifiers (IDs) associated with the agent. The number of different IDs in the list corresponds to the number of times an agent receives something confidentially over a password channel.

Every time an agent A with ID id_1 uses a secure replay-free password channel to send $pw(m)$ to an agent B, it generates a fresh id_2 meant for B and adds the facts $\mathsf{Tic}(id_1, pw(m))$ and $\mathsf{Tic}(id_2, pw(m))$ into the action of the protocol rule. This means that the password $pw(m)$ is now associated with these two IDs. The password is then sent using the following rule.

$$[\mathsf{Snd}_{\mathsf{SRPw}}(id, A, B, pw(m))] \xrightarrow{\mathsf{Snd}_{\mathsf{SRPw}}(id,A,B,pw(m))} [\mathsf{Rcv}_{\mathsf{SRPw}}(id, A, B, pw(m)), \mathsf{PwSent}(pw(m))] \tag{3.15}$$

The id in this rule corresponds to id_2 above, which is assigned to the receiver. The $\mathsf{PwSent}(pw(m))$ fact says that $pw(m)$ was sent over the channel and can be correctly guessed by the adversary. Guessing is modeled with the following rule:

$$[\mathsf{PwSent}(pw(m))] \xrightarrow{\mathsf{CorrectGuess}(pw(m))} [\mathsf{Out}(pw(m))] \tag{3.16}$$

Furthermore, an agent can be stopped at any point in the protocol by an additional stopping rule defined as follows:

$$[\mathsf{AgentState}(A, c, n, \langle id_1, id_2 \rangle)] \xrightarrow{Stop(id_1), Stop(id_2)} [] \tag{3.17}$$

The rule deletes the agent state and indicates that all its IDs are stopped. Note that this rule needs to be specified differently depending on how many IDs the list has (in the case above, there are two).

Now, the core idea, namely that guessing a password can only happen after all associated runs were stopped, can be expressed by an axiom in Tamarin, which makes sure that all considered traces satisfy it. We want to express

that every time a password is associated with a certain ID and this password is guessed (after the association) by the adversary, then, the run with this ID needs to have been stopped before the password is guessed. In Tamarin, the axiom looks as follows:

```
   axiom guess_after_stop:
"All id p #i #j.  Tic(id,p) @i & CorrectGuess(p) @j & i<j
==> Ex #k.  Stop(id) @k & k<j"
```

3.3 Run Identifiers

One of the problems encountered with buttons is that in case the human needs to verify something on the screen of a device with a button press, it is possible in our model that the human presses the button because in the current run with the device, the human sees the correct message, but the button press itself is then received by the device in a different run where it sent something different. In reality, this might happen in a situation where the screen of the device changes right before the human presses the button, but the human does not immediately realize this and so he verifies a wrong message. If we want to exclude scenarios like this, we can include a run identifier into the send and receive facts. In the protocol, the device generates a run identifier freshly and then sends a message to the human, where the facts include the run identifier. The same run identifier is then also included in the facts of the button press the human sends to the device. The device then only accepts the message if the run identifiers are the same.

In fact, in all protocols we analyze in Chapter 5, we need to include run identifiers whenever we use a button or an LED to verify some property about a message previously sent the other way round.

We could change all rules seen so far to include run identifiers by including them in all send and receive facts as well as the auxiliary facts such as !Auth. The channels where we actually include run identifiers later in the protocols are the invariant small and invariant password channels. The rule for the invariant small channel with run identifiers looks as follows:

$$[\mathsf{Snd}_{\mathsf{InvSId}}(rid, A, B, c, m), !\mathsf{Possible}(A, B, c, m)] \xrightarrow{\mathsf{Snd}_{\mathsf{InvSId}}(rid,A,B,c,m)} [\mathsf{Rcv}_{\mathsf{InvSId}}(rid, A, B, c, m), \mathsf{Out}(\langle A, B, c, m\rangle)] \tag{3.18}$$

The rule for the invariant password channel with run identifiers looks as follows:

$$[\mathsf{Snd}_{\mathsf{InvPwId}}(rid, A, B, pw(m))] \xrightarrow{\mathsf{Snd}_{\mathsf{InvPwId}}(rid,A,B,pw(m))} [\mathsf{Rcv}_{\mathsf{InvPwId}}(rid, A, B, pw(m)), \mathsf{Out}(\langle A, B, pw(m)\rangle)] \tag{3.19}$$

We call the identifier rid in order to avoid confusion with the identifier id used in password channels.

Chapter 4

Communication Topology Model

In this section, we introduce a communication topology model, which is basically the same one used in [1] except for the fact that we allow multiple edges between two nodes as well as our new types of channels.

A communication topology τ is an edge- and vertex-labeled, directed multigraph $\tau = (V, E, \eta, \mu)$, where V is the set of vertices, $E \subseteq V \times V \times \mathbb{N}$ the set of edges and η and μ are labeling functions for vertices and edges. We define E as a set where two different edges between two vertices are distinguished by the third component of the triples in E. We use this definition rather than defining E to be a multiset of ordered pairs of vertices because of the labeling function μ which needs a set as a domain in order to distinguish two different edges between two vertices.

A vertex in this topology corresponds to a role in the protocol and an edge to a communication channel between the two roles. The vertex labeling function $\eta : V \to \mathcal{P}(\Sigma)$ assigns capability to the vertices. The edge labeling function $\mu : E \to C_{net} \cup C_{human}$ assigns channel types to edges, where $C_{net} = \{\rightarrow, \bullet\!\!\rightarrow, \rightarrow\!\!\bullet, \bullet\!\!\rightarrow\!\!\bullet\}$ are the standard unrestricted channels used between devices over a network, representing insecure, authentic, confidential and secure channels respectively and $C_{human} = \{\overset{s}{\bullet\!\!\rightarrow}, \overset{p}{\bullet\!\!\rightarrow}, \overset{s}{\bullet\!\!\rightarrow\!\!\bullet}, \overset{p}{\bullet\!\!\rightarrow\!\!\bullet}\}$ represent invariant small, invariant password, secure small and secure password channels respectively. These channels are used to model the communication between the human and the devices. Note that in C_{human}, the secure channels are replay-free. The channels between a human and a device are always at least invariant, since the adversary can never change the sender, receiver or message in the setting of the human typing or seeing something on a screen. However, in the invariant case, the adversary can see the message (e.g. he can observe what the human types), while in the secure case, he does not see anything. Furthermore, recall that all agents executing the roles are assumed to be honest. Also, we assume that the nodes have no private shared initial knowledge.

Chapter 5

Internet of Things Topologies

In this chapter, we are going to analyze different topologies in a setting where there is a human who owns a smart device (or smart thing) such as a light bulb or a thermostat he wants to control with a Wi-Fi-enabled device such as a smart phone, tablet or desktop computer. In this analysis, we are always talking about a smart phone, although that is just one example of a device that could be used. We denote the human as H, the smart device as D and the smart phone as P. Recall that there is no private shared initial knowledge. The capabilities of the human are restricted: He cannot encrypt, decrypt, sign or hash any terms, verify signatures or calculate a public key given a secret key. Between D and P, only channels from C_{net} exist, while between H and D as well as H and P, only channels from C_{human} exist. Furthermore, we always have insecure channels between D and P in both directions (corresponding to a wireless connection).

We are going to draw such topologies as graphs in the following way: H is drawn as a dashed circle to represent its limited capabilities, while P and D are drawn as simple circles. The edges are drawn according to what their edge label looks like. An example of such a graph is depicted in Figure 5.1. In this example, the channels between P and D are insecure. Furthermore, H has an invariant small channel to D, D has a secure small channel to H, H has a secure password channel to P and P has an invariant password channel to H.

Our objective is to establish a shared key between D and P in order for the smart phone to be able to send messages to the device securely (and vice versa), such that an adversary is not able to see or alter the messages. Expressed with our channel properties from Chapter 2.3, we want the key to be confidential and satisfy non-injective agreement. All protocols we are introducing are verified with Tamarin. The Tamarin theory files can be found on [12].

We are going to show the protocols in the extended Alice & Bob notation, also taken from [1]. In this notation, we write $A \to B : m$ to denote the transmission of m from A to B over the channel written between the two roles (in this case, it is an insecure channel, but it could be any channel from $C_{net} \cup C_{human}$). The way in which the sender and the receiver parse the message might be different though: For example, the sender might encrypt a message and then send it,

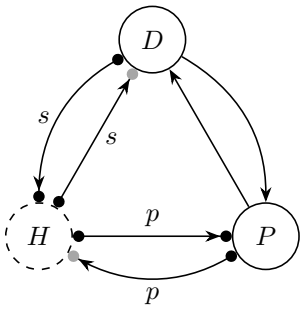


Figure 5.1: An example of an Internet of Things topology

but the receiver cannot decrypt it and just gets some value it cannot parse. In the same way, the sender might send a value it cannot parse (because it got it from another agent), but the receiver expects the message to be of some specific encryption. To deal with such cases, we write $A \to B : m \,/\, m'$ if m is the way A parses the message and m' the way B parses the message. If A generates a fresh value and then sends it to B, we write $A \to B : \text{fresh}(m).m$. Obviously, the message sent could also be more complicated, it just needs to contain m as a subterm. Initial knowledge of roles is denoted as $A : \text{knows}(m)$ at the beginning of the protocol.

We will first analyze different topologies and find protocols (in our symbolic model) in Sections 5.1 and 5.2, before we then leave the symbolic model in Section 5.3 and ask ourselves how much entropy the values used in the protocols need to have in order to compare the protocols in terms of convenience for the human.

5.1 Possibility in General Topologies

In this section, we want to see for which topologies consisting of H, D and P we can find a protocol establishing a shared key between D and P and for which ones no such protocol exists. Let us first consider the case where there are only insecure channels between D and P. These channels model the insecure wireless connection between the two nodes. We assume that no other direct channels exist between D and P.

We are first using impossibility results from [1] to exclude some topologies where no protocol exists. Because these results work regardless of restrictions, we do not consider the different types of restrictions (small and password channels) for a moment. There are three possibilities for channels from H to D: Either, there is no channel at all, there is an invariant channel or there is a secure channel. From D to H, from H to P and from P to H, the same three possibilities exist. Therefore, we get $3^4 = 81$ different cases.

The impossibility results are captured in two lemmas. The first one essentially states that for two distinct nodes S and R, where at least one of them has no initial knowledge, there is no protocol providing a confidential channel

from S to R if there is no confidential (or secure) channel outgoing from S or incoming to R and if there is no authentic (or stronger, namely invariant or secure) channel outgoing from R or incoming to S. As stated in a remark in [1], the lemmas also work if neither S nor R has an empty initial knowledge, but none of them have any private knowledge initially shared with a different agent. Since this is true in our topology model anyway, we do not have to state that explicitly in the lemmas.

Formally, the first lemma is stated as follows.

Lemma 1. *Let $\tau = (V, E, \eta, \mu)$ be a communication topology where $S, R \in V$ are distinct roles. If the following two conditions are satisfied, then there exists no protocol for τ that provides a confidential channel from S to R.*

1. $\forall (a, b) \in E : (a = S \vee b = R) \rightarrow \mu(a, b) \in \{\rightarrow, \bullet\!\!\rightarrow, \overset{s}{\bullet\!\!\rightarrow}, \overset{p}{\bullet\!\!\rightarrow}\}$
2. $\forall (a, b) \in E : (a = R \vee b = S) \rightarrow \mu(a, b) \in \{\rightarrow, \rightarrow\!\!\bullet\}$

The second lemma is analogous to the first one and says that if we have two distinct nodes S and R, there is no protocol providing an authentic channel from S to R if there is no authentic (or stronger, namely invariant and secure) channel outgoing from S or incoming to R and if there is no confidential (or secure) channel outgoing from R or incoming to S. Formally, it is stated as follows.

Lemma 2. *Let $\tau = (V, E, \eta, \mu)$ be a communication topology where $S, R \in V$ are distinct roles. If the following two conditions are satisfied, then there exists no protocol for τ that provides an authentic channel from S to R.*

1. $\forall (a, b) \in E : (a = S \vee b = R) \rightarrow \mu(a, b) \in \{\rightarrow, \rightarrow\!\!\bullet\}$
2. $\forall (a, b) \in E : (a = R \vee b = S) \rightarrow \mu(a, b) \in \{\rightarrow, \bullet\!\!\rightarrow, \overset{s}{\bullet\!\!\rightarrow}, \overset{p}{\bullet\!\!\rightarrow}\}$

Note that the authenticity property introduced in [1] which is referenced in Lemma 2 is a weaker form of authenticity than non-injective agreement. The lemma therefore also works if we replace the basic authenticity property by non-injective agreement.

Lemmas 1 and 2 are taken from [1]. The difference is that we do not explicitly make the assumption that the nodes are honest and that they have no private shared initial knowledge, because that is true in our definition of topologies anyway. The proof of the lemmas are also found in [1].

Note that in [1], there was no mention of invariant channels. However, it should be easy to extend the proof to topologies with invariant channels.

We are going to prove the following impossibility lemma using Lemma 1. Note that just for the lemma, we are going to relax our constraints and also allow authentic and confidential (not only invariant and secure) channels between H and the devices. This is fixed right after the lemma.

Lemma 3. *If there is no confidential channel from D to H, no authentic channel from H to D and only insecure channels between D and P, then there is no protocol providing a confidential channel from D to P.*

Proof. Let us merge the nodes H and P into one node HP by considering all channels between D and H or between D and P to be channels between D and HP.

If there is no confidential channel from D to H in the original topology, there is also no confidential channel from D to HP, since there are only insecure channels between D and P. Analogously, if there is no authentic channel from H to D, there is no authentic channel from HP to D. Since D and HP are the only nodes in the new topology, that means that there is no confidential channel outgoing from D or incoming to HP and no authentic channel outgoing from HP or incoming to D. By applying Lemma 1, there is no protocol providing a confidential channel from D to HP. Because of the way we defined HP, there is no protocol providing a confidential channel from D to P in the original topology either. □

Therefore, we certainly need a confidential channel from D to H or an authentic channel from H to D if we want secure communication between D and P. Since in reality, we only have invariant and secure channels, we need at least a secure channel from D to H (an invariant channel is not confidential) or any (invariant or secure) channel from H to D (invariant and secure channels are both authentic).

Analogously, we get the following lemma:

Lemma 4. *If there is no confidential channel from H to P, no authentic channel from P to H and only insecure channels between D and P, then there is no protocol providing a confidential channel from D to P.*

Proof. Let us merge the nodes H and D into HD in the same manner as in the previous lemma.

If there is no confidential channel from H to P in the original topology, there is also no confidential channel from HD to P, since there are only insecure channels between D and P. Analogously, if there is no authentic channel from P to H, there is no authentic channel from P to HD. Since HD and P are the only nodes in the new topology, that means that there is no confidential channel outgoing from HD or incoming to P and no authentic channel outgoing from P or incoming to HD. By applying Lemma 1, there is no protocol providing a confidential channel from HD to P. Because of the way we defined HD, there is no protocol providing a confidential channel from D to P in the original topology either. □

Therefore, we certainly need a confidential channel from H to P or an authentic channel from P to H if we want secure communication between D and P. In our case, this means that we need a secure channel from H to P or any (invariant or secure) channel from P to H.

Therefore, Lemma 1 leads to two necessary conditions: There needs to be a secure channel from D to H or any channel from H to D (this follows from Lemma 3) and there needs to be a secure channel from H to P or any channel from P to H (this follows from Lemma 4).

Using Lemma 2, we could again come up with two secondary lemmas analogous to Lemmas 3 and 4 to see that there needs to be at least a secure channel from H to D or any channel from D to H and at least a secure channel from P to H or any channel from H to P.

So, if we want to find a protocol providing secure channels between D and P, we have four necessary conditions to be satisfied. Because establishing a key between D and P is one way to provide secure channels, the conditions need to be satisfied in order to be able to establish a key. They are as follows:

1. A secure channel from D to H or any channel from H to D
2. A secure channel from H to D or any channel from D to H
3. A secure channel from P to H or any channel from H to P
4. A secure channel from H to P or any channel from P to H

Looking at nodes D and H, we need to have at least one secure channel in any direction or some channels in both directions. So, if there is no channel at all or just one invariant channel in any direction, there is no protocol. We have six combinations of channels such that a protocol is still possible:

1. A secure channel from D to H
2. A secure channel from H to D
3. An invariant channel from D to H and an invariant channel from H to D
4. An invariant channel from D to H and a secure channel from H to D
5. A secure channel from D to H and an invariant channel from H to D
6. A secure channel from D to H and a secure channel from H to D

For nodes P and H, we analogously also have six possibilities, so we get 36 cases in total where there might be a protocol. In all the other $81 - 36 = 45$ cases, it is impossible (by the impossibility lemmas) to find a protocol. Since the last three of the six cases are all stronger than the third, we are going to postpone the treatment of them and for now consider the channels between H and a device to be invariant if there exist channels in both directions.

Now, let us take into account our restrictions. Each channel can either be a small channel (button, LED) or a password channel (keypad, display). Instead of only three, we get the following eight cases between H and D:

1. A secure small channel from H to D
2. A secure password channel from H to D

3. A secure small channel from D to H
4. A secure password channel from D to H
5. An invariant small channel from H to D and an invariant small channel from D to H
6. An invariant password channel from H to D and an invariant small channel from D to H
7. An invariant small channel from H to D and an invariant password channel from D to H
8. An invariant password channel from H to D and an invariant password channel from D to H

Between H and P, we assume that there is a keypad or a display, because P is a smart phone. We are therefore not going to distinguish between the restrictions of the channels between H and P and just consider the channels to be password channels. In the graphs representing the topologies, we therefore omit the p label of the channels between H and P. There are still three possibilities for channels between H and P: A secure channel from H to P, a secure channel from P to H and invariant channels in both directions. In total, we get $8 \cdot 3 = 24$ cases.

Again, note that in theory, a small channel could act as a password channel, for example by sending each bit of a password separately. So, theoretically, if we found a protocol using a password channel, it would also work for a small channel. However, due to inconvenience, we want to avoid such things in a real-life scenario, so, we want to find different protocols here.

We are now trying to find protocols covering as many of those cases as possible. Note that trivially, a password channel is stronger than a small channel and a secure channel is stronger than an invariant channel. A protocol using a small channel can therefore also be used if it is replaced by a password channel and a protocol using an invariant channel can also be used if it is replaced by a secure channel.

Note that in most of our protocols, at some point, a device calculates a hash of some key and sends it to the human who will then send it on or compare it to some other value. In order to make the protocol more convenient for the human, but still ensure that the hash function is a one-way function, we define it as first applying a cryptographic hash function and then taking the first few bits (or transforming it into decimal and taking the first few digits) of the result and use this as a hash. The question of how many digits should be taken is dealt with in Section 5.3.

5.1.1 SPEKE Protocols

Simple Password Exponential Key Exchange (SPEKE) is a method for password-authenticated key exchange (PAKE), proposed in [5]. The protocol runs as follows: A and B share a password π and there is a fixed safe prime p (a prime of

the form $2q+1$, q being a prime as well) and a cryptographic hash function $h()$. They now both construct $g = h(\pi)^2 \bmod p$ and then perform a Diffie-Hellman key exchange using this g as the generator. They both get a key k which is the same for both parties if and only if they agree on π.

In the regular (and unauthenticated) Diffie-Hellman protocol, the adversary can perform a man-in-the-middle-attack by intercepting the messages and constructing two different keys with A and B, masquerading as A to B and vice versa. Here, this is only possible if the adversary guesses the password correctly and therefore gets the correct g. Because the attack is on-line, the adversary can only make one password guess per run of this protocol, which makes the protocol secure enough for a password with low entropy.

Let us have a look at some protocols relying on the SPEKE protocol. Assume that we have a secure channel from H to P and a secure password channel from H to D.

The protocol starts with the human typing a certain password securely into both devices. The devices then perform the SPEKE protocol in order to get a shared key k. Since the password is the same for both devices, they end up with the same k. The whole protocol is called SPEKE1.

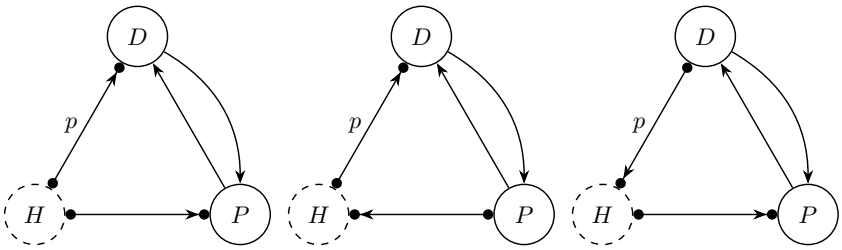
In Alice & Bob notation, the SPEKE1 protocol looks as follows:

$$
\begin{aligned}
H &: \text{knows}(\langle P, D, pw \rangle) \\
H &\overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P : pw \\
H &\overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D : pw \\
P &\rightarrow D : \text{fresh}(a).h(pw)^{2a} \;/\; A \\
D &\rightarrow P : \text{fresh}(b).h(pw)^{2b} \;/\; B \\
P &\rightarrow D : senc(\langle P, D \rangle, B^a) \;/\; senc(\langle P, D \rangle, A^b) \\
D &\rightarrow P : senc(\langle D, P \rangle, A^b) \;/\; senc(\langle D, P \rangle, B^a)
\end{aligned}
$$

The key k mentioned above is $B^a = A^b$ here. The last two protocol steps are necessary in order to let P and D know that the other device also knows the key.

There is an alternative version of this protocol, which we call SPEKE2: Instead of having a secure channel from H to P, we have a secure channel from P to H. The protocol then begins with P choosing the password, showing it securely to H who then types it into D. The rest stays the same.

In Alice & Bob notation, the SPEKE2 protocol looks as follows:



(a) SPEKE1 (b) SPEKE2 (c) SPEKE2

Figure 5.2: The different topologies in which SPEKE protocols can be executed

$$
\begin{aligned}
H &: \text{knows}(\langle P, D\rangle)\\
P &: \text{knows}(pw)\\
P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H &: \langle \text{`1'}, pw\rangle\\
H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D &: \langle \text{`2'}, pw\rangle\\
P \rightarrow D &: \text{fresh}(a).h(pw)^{2a} \;/\; A\\
D \rightarrow P &: \text{fresh}(b).h(pw)^{2b} \;/\; B\\
P \rightarrow D &: senc(\langle P, D\rangle, B^a) \;/\; senc(\langle P, D\rangle, A^b)\\
D \rightarrow P &: senc(\langle D, P\rangle, A^b) \;/\; senc(\langle D, P\rangle, B^a)
\end{aligned}
$$

The terms '1' and '2' serve as tags in order to avoid a confusion between the two parts of sending the password from P to D via H. This is needed to analyze the protocol in Tamarin properly.

Obviously, the SPEKE2 protocol also works with P and D interchanged, so, in a setting where D sends a password to P via H. Therefore, the SPEKE protocols can be used in the three cases depicted in Figure 5.2.

5.1.2 Relay Protocol

Let us consider a protocol where the device D has a keypad (modeled as an invariant password channel from H to D) and an LED (modeled as an invariant small channel from D to H). The smart phone P has a button and the ability to display information on the screen for the human (modeled as two invariant channels in both directions, we do not differentiate between the restrictions here).

The protocol now works as follows: First, D generates a key pair consisting of a secret key sk and the corresponding public key pk. Then, it sends the public key to P over the insecure channel. P then generates a fresh key k, encrypts it with the public key pk and sends the encrypted key $\{k\}_{pk}$ over the insecure channel to D, which can decrypt the message and get k. Now, obviously, P and D are not sure whether they actually have the same k and pk. The adversary might intercept the encrypted key $\{k\}_{pk}$ and replace it with any other

key $\{k'\}_{pk}$, because he knows the public key pk. In this case, P has key k, but D actually has key k'. The adversary might also intercept the public key sent from D to P, replace it with its own public key pk' which makes P send the message $\{k\}_{pk'}$, which the adversary can decrypt, therefore getting k. He then encrypts k with D's public key and sends it to D. In this case, D and P own the same key k, but the adversary also has knowledge of the key.

However, note that if the adversary does neither alter the public key pk nor the encrypted key $\{k\}_{pk}$, the key k is actually a shared secret between D and P, since the adversary is not able to decrypt it. Therefore, if D and P have the same values for k and pk, the protocol works. In order to verify this, they both compute a hash of the pair $\langle k, pk\rangle$ (recall that this hash corresponds to a prefix of a cryptographic hash as explained earlier). P then shows the hash to H on the screen. The human types it into D which compares it to the hash it calculated itself. If the hashes are the same, it lights up the LED. The human then presses the button on P to make it aware that the hashes were actually equal. Therefore, the devices can now use k as a shared key. Because H acts as a relay in this protocol, we call it the Relay protocol.

All channels involving the human are modeled as invariant channels: The adversary might observe whatever the human types or sees, but is not able to change the sender, receiver or message.

The Relay protocol in Alice & Bob notation looks as follows:

$$
\begin{aligned}
D &: \text{knows}(skD)\\
H &: \text{knows}(\langle D, P\rangle)\\
D \to P &: pk(skD) \;/\; pkD\\
P \to D &: \text{fresh}(k).aenc(\langle k, P, D\rangle, pkD) \;/\; aenc(\langle k, P, D\rangle, pk(skD))\\
P \overset{p}{\bullet\!\!\to} H &: \langle\text{`1'}, h(\langle k, pkD\rangle)\rangle \;/\; \langle\text{`1'}, hash\rangle\\
H \overset{p}{\bullet\!\!\to} D &: \langle\text{`2'}, hash\rangle \;/\; \langle\text{`2'}, h(\langle k, pk(skD)\rangle)\rangle\\
D \overset{s}{\bullet\!\!\to} H &: \text{`}ok\text{'}\\
H \overset{s}{\bullet\!\!\to} P &: \text{`}ok\text{'}
\end{aligned}
$$

Obviously, the protocol works the same way if D and P are interchanged.

Therefore, the Relay protocol needs invariant channels in both directions between H and P as well as between H and D. If P sends the hash and H needs to relay it to D, the channel from H to D needs to be a password channel (the one from D to H can be small). If D and P are interchanged in the protocol, it is the other way round: The channel from D to H needs to be a password channel, while the channel from H to D can be small. The protocol therefore covers three cases, also seen in Figure 5.3.

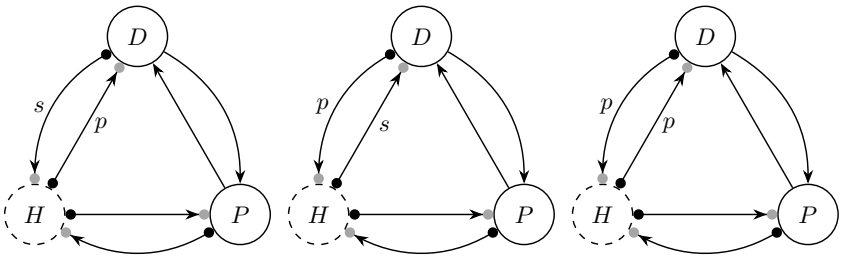


Figure 5.3: The different topologies in which the Relay protocol can be executed

5.1.3 Confirmation Number Protocols

There are two different protocols using confirmation numbers. In the first one (which we call Conf1), we assume there is a secure channel from P to H and that D has a keypad (modeled as an invariant password channel from H to D) and an LED (modeled as an invariant small channel from D to H).

First, as in the Relay protocol, D sends its public key pk insecurely to P and P then sends $\{k\}_{pk}$ to D. P then sends the hash of $\langle k, pk\rangle$ securely to H, together with a freshly generated confirmation number (similar to a password). H then sends the hash to D which lights up the LED if the hash it gets from H is equal to the hash it calculated from its values of k and pk. If that happens, H types the confirmation number into D which sends it to P. If P now gets the same number as it originally sent, it knows that the insecure key exchange must have been successful and therefore, k can be used as a shared key.

Note that if D had a display, it could also send the hash directly to H, who then compares the hashes and sends the confirmation number if they are equal. While this protocol would be one step shorter, it would need stronger channels (namely invariant password channels between H and D in both directions).

The Conf1 protocol looks as follows in Alice & Bob notation:

$$
\begin{aligned}
D &: \text{knows}(skD)\\
H &: \text{knows}(\langle D, P\rangle)\\
D \to P &: pk(skD) \;/\; pkD\\
P \to D &: \text{fresh}(k).aenc(\langle k, P, D\rangle, pkD) \;/\; aenc(\langle k, P, D\rangle, pk(skD))\\
P \overset{p}{\bullet\!\!\to\!\!\bullet} H &: \text{fresh}(conf).\langle h(\langle k, pkD\rangle), conf\rangle \;/\; \langle hash, conf\rangle\\
H \overset{p}{\bullet\!\!\to\!\!\bullet} D &: hash \;/\; h(\langle k, pk(skD)\rangle)\\
D \overset{s}{\bullet\!\!\to\!\!\bullet} H &: \text{`}ok\text{'}\\
H \overset{p}{\bullet\!\!\to\!\!\bullet} D &: conf\\
D \to P &: conf
\end{aligned}
$$

The intuition behind this protocol is as follows. As in the Relay protocol, D and P want to make sure they both have the same value for $h(\langle k, pk\rangle)$. P's hash

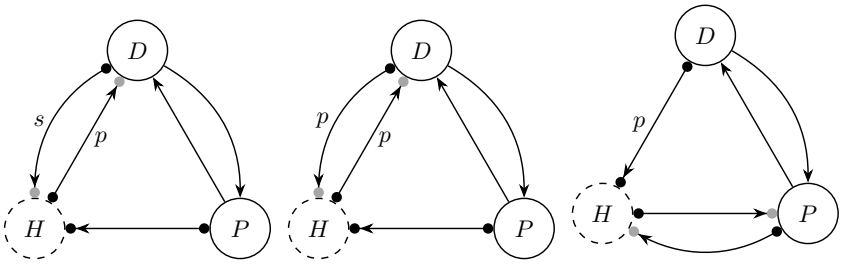


Figure 5.4: The different topologies in which Conf1 can be executed

is sent to D via H, meaning that D definitely knows whether the two hashes are equal. P additionally sends a confirmation number secretly to H and waits for D to send the same number back. Now, obviously, the adversary might try to insert the confirmation number in order to make P believe that the key exchange worked as expected, while in reality, he tampered with it. However, the confirmation number is only made public (by sending it over the invariant channel from H to D) if the hashes are actually equal, so, if the adversary did not tamper with the key exchange. If he did attack the protocol, then, with very high probability, the hashes are not equal and he can only guess the confirmation number in order to succeed in the attack.

The Conf1 protocol certainly needs a secure channel from P to H, an invariant password channel from H to D and an invariant small channel from D to H. The symmetric version (with D and P interchanged) needs a secure password channel from D to H and invariant channels in both directions between H and P. We therefore cover the three cases seen in Figure 5.4.

In the second protocol (which we call Conf2), we assume that there is a secure channel from H to P and D has a display (modeled as an invariant password channel from D to H) and a button (modeled as an invariant small channel from H to D).

As usual, D sends its public key pk insecurely to P and P then sends $\{k\}_{pk}$ to D. D then sends the hash of $\langle k, pk \rangle$ to H. H then sends the hash together with a freshly generated confirmation number to P. P can then check if the hash of its values of k and pk equals the hash it received from H. If this is the case, it sends the confirmation number to D which then shows it to H (together with the same hash as before, in order to keep the context of the message). If H gets the right value, he presses the button. Therefore, D also knows that the insecure key exchange must have been successful and therefore, k can be used as a shared key. The basic idea of this protocol is again the same: The confirmation number only becomes public if the hashes were equal.

The Conf2 protocol looks as follows in Alice & Bob notation:

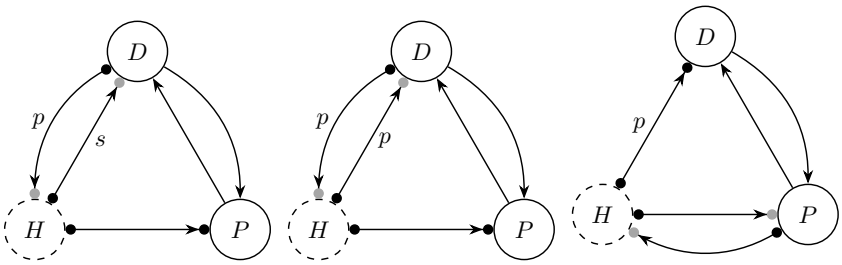


Figure 5.5: The different topologies in which Conf2 can be executed

$$
\begin{aligned}
D &: \text{knows}(skD) \\
H &: \text{knows}(\langle D, P\rangle) \\
D \to P &: pk(skD) \ / \ pkD \\
P \to D &: \text{fresh}(k).aenc(\langle k, P, D\rangle, pkD) \ / \ aenc(\langle k, P, D\rangle, pk(skD)) \\
D \overset{p}{\bullet\!\!\to\!\!\bullet} H &: h(\langle k, pk(skD)\rangle) \ / \ hash \\
H \overset{p}{\bullet\!\!\to\!\!\bullet} P &: \text{fresh}(conf).\langle hash, conf\rangle \ / \ \langle h(\langle k, pkD\rangle), conf\rangle \\
P \to D &: conf \\
D \overset{p}{\bullet\!\!\to\!\!\bullet} H &: \langle conf, h(\langle k, pk(skD)\rangle)\rangle \ / \ \langle conf, hash\rangle \\
H \overset{s}{\bullet\!\!\to\!\!\bullet} D &: \text{`}ok\text{'}
\end{aligned}
$$

For the Conf2 protocol, we need a secure channel from H to P, a small channel from H to D and a password channel from D to H. The symmetric version needs a secure password channel from H to D and invariant channels in both directions between H and P. We therefore cover the three cases seen in Figure 5.5.

Together, the protocols cover 12 out of 24 cases. We can see which topologies are covered in Table 5.1.

5.1.4 Impossible Topologies

For two cases, we already know that no protocols can exist: If there are only channels from P to H and from D to H (small or password) but none in the other directions, there is no information flowing from H to the devices and therefore, we cannot find any protocol that establishes a key between D and P. The entries with a dash in the table represent these two cases.

The cases which are now still blank are cases where there is probably no protocol (apart from the ones using a small channel as a password channel): For the cases where there are only small channels between H and D, we cannot get more than one bit of information from the state of D or give D more than one bit of information of a password or the state of P. Therefore, there is probably no protocol there.

	$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P$ $P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$			
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$	SPEKE1	SPEKE2	Conf2
$D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$		-	
$D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	SPEKE2	-	Conf1
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$			
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$		Conf1	Relay
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	Conf2		Relay
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	Conf2	Conf1	Relay

Table 5.1: Topologies for which the indicated protocols provide a shared key between D and P. Entries with a dash indicate that no protocol exists.

Now, consider the case where there is a secure channel from H to P, an invariant password channel from H to D and an invariant small channel from D to H. A protocol along the lines of Relay, Conf1 or Conf2, where the two devices exchange some keys insecurely and want to verify that they are the same is not possible, since H never gets any information from the state of P directly, and, since D only has a small channel to H, nothing can be relayed from D to P via H.

Another possibility for H would be to send some freshly generated values to the devices. However, since H cannot perform cryptographic operations, for every secret value x sent from H to P, he can either also send it to D or not send anything containing x to D. In the former case, the value is seen by the adversary and is therefore not secret anymore and in the latter case, it is irrelevant if H really sends x to P or if P itself generates x. Every protocol in which x is sent from H to P also works the same way if P generates x. This means that it is not possible to send any relevant secret value from H to P. So, it seems highly likely that there is no protocol.

Similarly, if there is a secure channel from P to H, a small channel from H to D and a password channel from D to H, the only information that H can send is a bit to D, but none to P. So, H cannot confirm any equality between some hashes like it did in Relay, Conf1 and Conf2. Furthermore, P cannot relay anything to D via H since there is only a small channel from H to D. So, also in this topology, there is probably no protocol.

5.1.5 Upgrading Invariant Channels

Recall that up to now, we considered the channels between H and a device to be invariant if they exist in both directions. We now want to upgrade the channels

and also look at cases where one or both of them are secure. Analyzing these topologies is not too difficult: For all cases where there are channels in both directions between H and D and we already found a protocol, we can just use the same protocol even if some of the channels are now secure. For the case where we have a secure channel from P to H, a small channel from H to D and a password channel from D to H (and we have found no protocol), strengthening the channels does not change the arguments we made for not finding a protocol. The same goes for the cases where both channels between H and D are small.

For the case where there is a secure channel from H to P, a password channel from H to D and a small channel from D to H, making the small channel secure does not change anything, but making the password channel secure actually makes the topology a supergraph of the topology where SPEKE1 can be used, so we obviously can use this protocol here as well.

If we consider upgrading the channels between H and P, making invariant channels secure does not change anything since in all cases, we either already found a protocol for invariant channels (in the rightmost column of Table 5.1) or we are in a case where there are only small channels between H and D and we therefore have no protocol anyway.

5.2 Possibility With Authentic Public Keys

Let us consider the same setting as above except for one additional channel: The device D is able to send its public key in some way authentically to P. For example, there could be a QR code containing the public key on the device which can be read by the camera of the smart phone.

Analogous to the case without this authentic transmission of the public key, we can use Lemma 1 to conclude that we need at least a secure channel from D to H or any channel from H to D. Also, we need at least a secure channel from H to P or any channel from P to H. Lemma 2 cannot be used to exclude anything because we now have a way to authentically transmit a message from D to P.

We therefore get a new case of possible channels between H and D, namely an invariant channel from H to D only. Considering restrictions, this becomes two new cases, namely an invariant small channel from H to D and an invariant password channel from H to D. Also, we have a new possibility between H and P, namely an invariant channel from P to H only. Together with the eight cases we had before, we get ten cases for channels between H and D. Between H and P, we only get one new case, since we do not differentiate between the restrictions here. In total, we get $10 \cdot 4 = 40$ cases. For the cases where we already found a protocol without the QR code, we can of course still use this protocol. Table 5.2 shows the new possibilities. We want to find new protocols filling in some of the blanks.

	$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} P$ $P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$	SPEKE1	SPEKE2	Conf2	
$D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$		-		
$D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	SPEKE2	-	Conf1	
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$		Conf1	Relay	
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	Conf2		Relay	
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	Conf2	Conf1	Relay	
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$				

Table 5.2: Topologies for which the indicated protocols provide a shared key between D and P. Entries with a dash indicate that no protocol exists.

5.2.1 The QR_Relay Protocol

Let us look at the case where we have an invariant channel from P to H and an invariant password channel from H to D. We find a protocol which looks as follows: D sends its public key pk via QR code to P, which then sends $\{k\}_{pk}$ to D and the hash of k to H. H sends this hash on to D, which then checks if the hash of its value of k corresponds to the hash it received from H. If yes, it knows that k is secure. Because the human basically only relays a hash from P to D, we call this protocol QR_Relay. Note that since pk is authentic, the two participants only need to make sure that their hashes of k are equal.

In Alice & Bob notation, the protocol looks as follows:

$$
\begin{aligned}
D &: \text{knows}(skD) \\
H &: \text{knows}(\langle D, P\rangle) \\
D \bullet\!\!\rightarrow P &: pk(skD) \ / \ pkD \\
P \rightarrow D &: \text{fresh}(k).aenc(\langle k, P\rangle, pkD) \ / \ aenc(\langle k, P\rangle, pk(skD)) \\
P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H &: \langle \text{`1'}, h(k)\rangle \ / \ \langle \text{`1'}, hash\rangle \\
H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D &: \langle \text{`2'}, hash\rangle \ / \ \langle \text{`2'}, h(k)\rangle
\end{aligned}
$$

Tamarin again needs tags to analyze the protocol correctly.

Obviously, the protocol also works in some other cases which are all stronger than this one. All cases which we have not yet covered before (without the QR

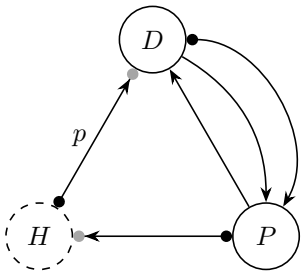


Figure 5.6: Minimal topology needed for the QR_Relay protocol

code) and where this protocol can be used are the following ones. The minimal topology (the first in this list) is seen in Figure 5.6.

1. An invariant channel from P to H and an invariant password channel from H to D
2. An invariant channel from P to H and a secure password channel from H to D
3. An invariant channel from P to H, an invariant password channel from H to D and an invariant small channel from D to H
4. An invariant channel from P to H, an invariant password channel from H to D and an invariant password channel from D to H
5. A secure channel from P to H and an invariant password channel from H to D
6. Two invariant channels between P and H (one in each direction) and an invariant password channel from H to D

5.2.2 The QR_Compare Protocol

Now, consider the case where there is an invariant channel from P to H, an invariant small channel from H to D and an invariant password channel from D to H. We can find a protocol here which looks as follows: First, D sends its public key by QR code to P. P then sends a key k encrypted with this public key back to D. They now both send the hash of k to H, who then presses a button on D if the hashes are equal. This way, D knows that the key is secure. P does not get any confirmation, but since it encrypts the key with an authentic public key, the key is secret from the point of view of P anyway. Since H is essentially just comparing two numbers, we call this protocol QR_Compare. Note that this protocol also works for the case where the channel from P to H is secure. This is a case we already had before and where we could not find a protocol without a QR code. These topologies are seen in Figure 5.7.

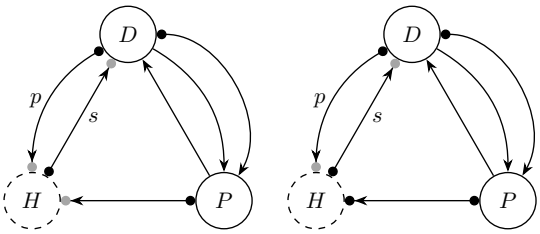


Figure 5.7: The different topologies in which the QR_Compare protocol can be executed

In Alice & Bob notation, the protocol QR_Compare looks as follows:

$$
\begin{aligned}
D &: \text{knows}(skD) \\
H &: \text{knows}(\langle D, P\rangle) \\
D \bullet\!\!\rightarrow P &: pk(skD) \ / \ pkD \\
P \rightarrow D &: \text{fresh}(k).aenc(\langle k, P\rangle, pkD) \ / \ aenc(\rangle k, P\rangle, pk(skD)) \\
D \overset{p}{\bullet\!\!\rightarrow} H &: h(k) \ / \ hash \\
P \overset{p}{\bullet\!\!\rightarrow} H &: h(k) \ / \ hash \\
H \overset{s}{\bullet\!\!\rightarrow} D &: \text{`}ok\text{'}
\end{aligned}
$$

5.2.3 The QR_Helper Protocol

Let us look at the case where there is a secure channel from H to P, an invariant password channel from H to D and an invariant small channel from D to H. We can find a protocol as follows: D sends its public key by QR code to P. H securely sends some helper value (with the same entropy as a password) called x to P. P then sends $\langle k, x\rangle$ encrypted with the public key to D. As soon as D gets the key, it lights up an LED to let the human know that it got the message. H then sends x to D which can then compare it with the x it already got from P. If the two values are the same, the key has to be secure. This works because the adversary is not able to change the key without (with a very large probability) also changing x. D would then get a different x from H which indicates that the key was changed by the adversary. To prevent the disclosure of x during the key exchange, H only sends x after he got the confirmation that the key exchange is finished. Because this protocol uses a helper value x, we call it QR_Helper. The corresponding topology is seen in Figure 5.8.

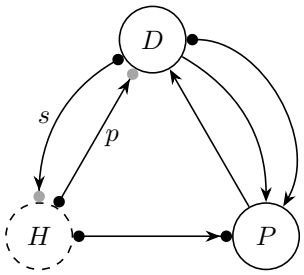


Figure 5.8: Minimal topology needed for the QR_Helper protocol

In Alice & Bob notation, the protocol QR_Helper looks as follows:

$$
\begin{aligned}
D &: \text{knows}(skD) \\
H &: \text{knows}(\langle D, P\rangle) \\
H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P &: \text{fresh}(x).x \\
D \bullet\!\!\rightarrow P &: pk(skD) \;/\; pkD \\
P \rightarrow D &: \text{fresh}(k).aenc(\langle k, x, P\rangle, pkD) \;/\; aenc(\langle k, x, P\rangle, pk(skD)) \\
D \overset{s}{\bullet\!\!\rightarrow} H &: \text{`}ok\text{'} \\
H \overset{p}{\bullet\!\!\rightarrow} D &: x
\end{aligned}
$$

Table 5.3 shows which topologies the new protocols cover.

5.2.4 Impossible Topologies

For the case where there is an invariant channel (small or password) from H to D and a secure channel from H to P, we cannot find any protocol, because it is not possible for H to send anything relevant securely from H to P, by arguments already discussed in Chapter 5.1.4 (relying on the fact that H cannot perform any cryptographic operations). As a side note: If he could perform all operations, the following protocol would be possible: H sends a password securely to P. P sends this password to D, encrypted with the public key P got from the QR code. H then sends a hash of the password to D, which can then check if the hash of its password and the hash received from H match. If yes, D knows that it has the correct password and D and P can execute the SPEKE protocol to get a shared key. The problem of this protocol is that H is not able to calculate a hash on his own without a computing device doing it for him.

For the cases where we only have an invariant (or secure) channel from P to H and a secure channel (small or password) from D to H (and none in the other directions), we again cannot find any protocol for the usual reason, there is no information flow from H to any of the devices.

For all the other cases, we observe that between H and D, we only have small channels and no password channels at all. So, H can only get at most

	$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} P$ $P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$	SPEKE1	SPEKE2	Conf2	QR_Relay
$D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$		-		-
$D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	SPEKE2	-	Conf1	-
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$	QR_Helper	Conf1	Relay	QR_Relay
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	Conf2	QR_Compare	Relay	QR_Compare
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	Conf2	Conf1	Relay	QR_Relay
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$	-			
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$	-	QR_Relay	QR_Relay	QR_Relay

Table 5.3: Topologies for which the indicated protocols provide a shared key between D and P. Entries with a dash indicate that no protocol exists.

one bit of information from D and vice versa. That means that apart from the protocols which use a small channel as a password channel, we probably cannot find any protocols.

Also, upgrading invariant channels to secure channels in the case of channels in both directions between H and D does not change anything, since in the case that both channels are small, upgrading the security does not help and in all other cases, we already found a protocol using only invariant channels. Upgrading them between H and P does not change anything either, because either we already found a protocol or we are in a case where between H and D, only small channels exist, so upgrading channels between H and P does not change anything there.

It is clear that if the roles of P and D are switched (so, P has a QR code that D can read), everything works symmetrically. In the case where they both can send their public keys authentically to one another, they can directly apply for example the Needham-Schroeder-Lowe protocol without even needing H. However, the manufacturer of the device D wants to keep it as simple as possible, so there will certainly not be a camera just for the purpose of scanning a QR code. So, in reality, only D can send a public key authentically to P.

5.3 Security Parameters

In this section, we want to find out how long the passwords, hashes and confirmation numbers need to be in order to keep the success probability of attacks of the adversary small while still not requiring the human to enter very long

values into the devices.

Note that the following analysis is done in a computational model, where we can talk about probabilities that two values are equal. This is in contrast to the symbolic model we have analyzed the protocols in, where two different symbols always denote two different values.

Let us consider some possible attacks an adversary can perform on our protocols. Recall the protocols beginning with an insecure key exchange (Relay, Conf1, Conf2). These protocols all follow the same principle: First, D sends its public key pk to P. Then, P generates a fresh key k, sends $\{k\}_{pk}$ to D who can decrypt it and therefore get k. Since everything is insecure, the adversary obviously might have intercepted and changed the keys. However, if he did not tamper with the channel, the key k is secure. That's why in all these protocols, P and D want to be sure that both of them have the same values of k and pk. Because there is a human involved, they transform the keys into a hash $h(\langle k, pk\rangle)$ and, by doing some protocol steps involving the human, they make sure that they both have the same value $h(\langle k, pk\rangle)$.

If the adversary injects keys such that there is a collision and D and P actually have the same hashes, but not the same values of k and pk, the adversary has successfully attacked the protocol. P and D believe they have a common secret, while in reality, this is not the case. Concretely, the adversary might do the following. First, D sends pk to P. The adversary intercepts this message and injects its own public key pk'. P, thinking that pk' is D's public key, sends back $\{k\}_{pk'}$, which the adversary can intercept and decrypt, since it is encrypted with his public key. The adversary now generates a key k', sends $\{k'\}_{pk}$ to D, which then decrypts it. So, D now has values k' and pk while P has values k and pk'. To successfully perform the attack, the adversary needs to generate a key k' such that $h(\langle k', pk\rangle) = h(\langle k, pk'\rangle)$. He tries to do this with brute-force: He just generates random keys k' until he finds one where the equation is satisfied. He then injects this key k' and the attack is successful. Recall that the hash function we use is defined as taking a prefix of the result of a cryptographic hash function. We need to make this assumption in order to prevent the adversary from making precalculations. With our hash function, he is only able to attack the protocol during the execution.

Let us try to calculate the approximate success probability he has. The number of keys he can check of course depends on his computing power and how long D waits for the key from P until there is a timeout. Let us just assume that he can check 10^9 (a billion) keys before there is a timeout. In our hash function, all possible hashes are equally likely to occur. With every key the adversary checks, the probability that the equation is satisfied is around $\frac{1}{n}$, where n is the number of possible hashes. The probability that the adversary will succeed in any of the trials is therefore $p = 1 - \left(\frac{n-1}{n}\right)^{10^9}$. If $n = 10^9$, so, if we used 9-digit numbers as hashes, $p \approx 0.63$. If $n = 10^{12}$, $p \approx 0.001$. If $n = 10^{15}$, $p \approx 10^{-6}$, so, one in a million. So, in order to keep the success probability of the adversary at around one in a million, we would need 15-digit numbers as hashes. In our case, we are calculating a cryptographic hash function, transforming it into a

decimal number and taking the first 15 digits.

Now, let us compare this to the situation where the protocols begin with a key exchange using an authenticated (e.g. by QR code) public key. Recall the protocols QR_Relay and QR_Compare. In both of them, D sends its public key pk authentically to P. P then generates a fresh key k, sends $\{k\}_{pk}$ insecurely to D who can decrypt it and therefore gets k. In these protocols, since pk is authentic, D and P must ensure that they both have the same key k and they do this by making sure that they have the same value $h(k)$. Here, the adversary might again try to inject a key k' such that $h(k) = h(k')$. However, since pk is authentic, he cannot inject his own public key and hence cannot get the key k. So, he cannot directly check whether the equation is satisfied. However, in QR_Relay and QR_Compare, as soon as P has sent $\{k\}_{pk}$, it immediately sends $h(k)$ to H, which the adversary can observe and therefore the brute-force attack would become possible. This problem can be solved by having D send a public value encrypted with k to P, e.g. $\{\text{'0'}\}_k$, as soon as D got the key k. From the point of view of P, as soon as it receives this value, it knows that D must have received the key (k is secret, only D is able to encrypt something with it) and so, he can send $h(k)$ to H without the adversary being able to use this value for injecting a key, since the key exchange has already finished. That means, the best thing he could do is to just pick a random key k' and send $\{k'\}_{pk}$ to D. If our usual hash function is used, the success probability of the adversary is $\frac{1}{n}$, where n is the number of possible hashes. So, in order to keep the success probability of the adversary at around one in a million, we only need 6-digit numbers instead of 15-digit numbers here! So, these protocols are more convenient for the human (he only needs to type in a short number) while still providing the same amount of security.

The Tamarin theory files of the extended versions of the protocols QR_Relay and QR_Compare with the added step mentioned above are denoted with the suffix '_nobf' on [12].

The SPEKE protocols rely on the security of the password used. Recall that the adversary can only make one password guess per protocol run. In SPEKE2, one of the devices generates the password and it is sent via the human to the second device. Assuming that the password is chosen uniformly at random, we only need a password space of 6-digit numbers in order to keep the success probability of the adversary at one in a million. In SPEKE1, where the human chooses the password, it is obviously not uniformly distributed and the security depends on the chosen password. We generally want a bigger password space when the human chooses the password.

Conf1 and Conf2 can additionally be attacked by first injecting different keys in the insecure key exchange and later guessing the confirmation number in order to make the agent checking the confirmation number believe that everything went alright and that the key is secure. If the agent checking the confirmation number only considers the first one it receives, brute-force attacks, which just send many confirmation numbers, are not possible. Therefore, the probability of an attack being successful is $\frac{1}{n}$, where n is the number of possible confirmation numbers. So, again, to keep the success probability of the adversary at one in

	$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} P$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} P$ $P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	$P \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$
$H \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} D$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} D$	SPEKE1	SPEKE2	QR_Relay	QR_Relay
$D \overset{s}{\bullet\!\!\rightarrow\!\!\bullet} H$		-		-
$D \overset{p}{\bullet\!\!\rightarrow\!\!\bullet} H$	SPEKE2	-	QR_Conf1	-
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$				
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{s}{\bullet\!\!\rightarrow\!\!\circ} H$	QR_Helper	QR_Relay	QR_Relay	QR_Relay
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	QR_Conf2	QR_Compare	QR_Compare	QR_Compare
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$ $D \overset{p}{\bullet\!\!\rightarrow\!\!\circ} H$	QR_Helper	QR_Relay	QR_Relay	QR_Relay
$H \overset{s}{\bullet\!\!\rightarrow\!\!\circ} D$	-			
$H \overset{p}{\bullet\!\!\rightarrow\!\!\circ} D$	-	QR_Relay	QR_Relay	QR_Relay

Table 5.4: Topologies for which the indicated protocols provide a shared key between D and P. Entries with a dash indicate that no protocol exists.

a million, we can use 6-digit numbers as confirmation numbers.

For the QR_Helper protocol, the value x also needs to be a 6-digit number in order to keep the success probability of the adversary at one in a million for the same reasons: The adversary can guess x once in the protocol (he injects a key k and some x and tries to guess the correct x such that D then gets the same x from H and deduces that the key is secure).

To sum up, we could say that SPEKE and the QR-protocols are more convenient than the others in the sense that they need less complex values to give the same amount of security. Looking back at the table of the protocols in case we have a QR code, we can replace several protocols by QR-protocols just by noticing that if a protocol works on certain topology, it also works on a stronger topology. For example, if we can use the protocol QR_Relay with an invariant channel from P to H, we can also use it with a secure channel from P to H. By doing this, we see that we can replace nearly all 'inconvenient' protocols except in two cases. However, note that in these protocols, if we replace the insecure transmission of the public key by an authentic transmission of the public key and the hashes $h(\langle k, pk\rangle)$ by $h(k)$, we get protocols that work for these cases as well and have the same advantage of working with smaller hashes. The two cases where this is needed are covered by Conf1 and Conf2 respectively. By doing this replacement, we get two protocols QR_Conf1 and QR_Conf2 which we can use here. The complete table with all 'inconvenient' protocols eliminated can be seen in Table 5.4.

There is some slight change in QR_Conf1 that we should mention: In the

case where we need this protocol, we originally (without QR code) had the protocol Conf1, but with P and D interchanged. So, P sent its public key and D then sent a key encrypted with the public key. Obviously, here, we need D to be the one to send the public key (because the QR code belongs to D) and P to be the one to send the encrypted key. Still, the rest of the protocol needs to remain interchanged (because the topology between H and the devices remains the same), so, for example, D is the one that sends to H the confirmation number securely and not P. Therefore, compared to the original confirmation number protocol Conf1, we keep the agents of the first part (the insecure key exchange) the same, but interchange them in the rest of the protocol. In the case where we need QR_Conf2, this problem does not occur, we can directly use Conf2 with the changes mentioned above the table.

5.3.1 Getting Rid of Hashes

Because hashes might bring security risks along (e.g. the brute-force attack described above), it might be better to try to completely get rid of hashes. For protocols with the QR code, this is actually possible. In Table 5.4, we have three protocols which do not contain any hashes: SPEKE1, SPEKE2 and QR_Helper. The other four protocols (QR_Relay, QR_Compare, QR_Conf1 and QR_Conf2) can be changed (all in a similar way) in order to get rid of the hashes.

The first two steps of each protocol stay the same: D sends its public key pk authentically to P, which then generates a fresh key k and sends back $\{k\}_{pk}$. Since pk is sent authentically, from the point of view of P, k is secure, since it is encrypted with the authentic key pk, which only D can decrypt. From the point of view of D, however, the key k is not yet secure, since the adversary could just inject any key (anybody can encrypt something with pk). To solve this problem, as a third step, D generates a fresh value x and sends $\{x\}_k$ to P. P only accepts this value if it is able to decrypt it, so if it really is encrypted with the key k that P generated before. D therefore knows that if the adversary injected some different key, P does not get the value and therefore, the protocol stops. However, if the adversary did not change anything, P should get the value x and now needs to convince D that it received it.

This is done differently in each protocol, namely in the same way as in the corresponding original protocol, just with x replacing $h(k)$ everywhere. In more detail, in QR_Relay, P sends x to H which forwards it to D. In QR_Compare, both D and P send x to H and if the two values are the same, H presses a button on D. In QR_Conf1, D sends x and a fresh value $conf$ securely to H. H then sends x to P. If the two values of x are the same, P sends an 'ok', after which H sends $conf$ to P, which forwards it to D, which can then check if it got the same $conf$ it generated. In QR_Conf2, D sends x to H, which forwards it securely to P, along with a fresh value $conf$. P compares the x it got from H with the one it got from D earlier and, if they are the same, sends $conf$ to D. D then forwards $conf$ (together with x to keep the context) to H. If H got the right value $conf$, he presses a button on D.

Another advantage of these versions of the protocols is that x is not a value

that needs to stay secret, so we do not have to worry about how much entropy the value has. In fact, it seems to be enough if x is a counter variable, so if every time D executes the protocol, it just increments the value of x by 1.

The Tamarin theory files of these new protocols without hashes are denoted with the suffix '_nohash' on [12].

Chapter 6

Examples

In this section, discuss additional protocols and real-world examples concerning the Internet of Things to further illustrate our topology model.

6.1 Compare Protocol

Consider the Bluetooth standard for data exchange. There, pairing two devices can be done by several versions of the so-called Secure Simple Pairing [3]. One of the versions looks as follows: Both devices have displays and buttons. The devices do some key exchange and afterwards both show a number to the human who then compares the two numbers and, if they are equal, presses the buttons on both devices. We introduce a protocol on an Internet of Things topology that works along the lines of this number comparison approach. The protocol is verified by Tamarin.

We are therefore given the nodes H, D and P as well as insecure channels between D and P. Furthermore, both P and D have a display and a button.

The protocol now works as follows: As usual, D sends its public key pk to P, which then generates a fresh key k, encrypts it with the public key and sends the encrypted key $\{k\}_{pk}$ over the insecure channel to D, which can decrypt the message and get k. D and P now want to verify that they have the same values for k and pk. They do this by both computing the hash of the pair $\langle k, pk \rangle$, and showing it to the human. The human can then verify if both hashes he sees are the same. If this is the case, he presses the buttons on both devices. The devices are now aware that they have the same values and can therefore use k as a shared key. Because H is only comparing two hashes and confirming their equality, the protocol is called Compare protocol. Note that the previously described QR_Compare protocol is similar, the difference is that in that protocol, the confirmation is only sent to D.

All channels involving the human are modeled as invariant channels: The adversary might observe the LED, display or button press of the human, but is not able to change the sender, receiver or message.

The Compare protocol in Alice & Bob notation looks as follows:

$$
\begin{aligned}
D &: \text{knows}(skD) \\
H &: \text{knows}(\langle D, P\rangle) \\
D \to P &: pk(skD) \ / \ pkD \\
P \to D &: \text{fresh}(k).aenc(\langle k, P, D\rangle, pkD) \ / \ aenc(\langle k, P, D\rangle, pk(skD)) \\
D \overset{P}{\bullet\!\!\to} H &: h(\langle k, pk(skD)\rangle) \ / \ hash \\
P \overset{P}{\bullet\!\!\to} H &: h(\langle k, pkD\rangle) \ / \ hash \\
H \overset{S}{\bullet\!\!\to} D &: \text{`}ok\text{'} \\
H \overset{S}{\bullet\!\!\to} P &: \text{`}ok\text{'}
\end{aligned}
$$

6.2 Key Derivation Function Protocols (WPA2)

Let us analyze protocols similar to the SPEKE protocols, using the same topologies. However, these protocols are less secure, as they do not satisfy confidentiality of the shared key, just non-injective agreement.

The basic protocol works as follows: The human types a password and a cryptographic salt securely into devices D and P. Both D and P use a publicly known key derivation function in order to get a key k with which they can encrypt what they are sending. An instantiation of such a protocol is the IEEE 802.11i-2004 standard [4], also known as WPA2. There, a key derivation function called PBKDF2 [6] is used. This function calculates a key k depending on five arguments: $k = \text{PBKDF2}(PRF, pw, salt, c, kLen)$. PRF is a pseudo-random function, pw is a password, $salt$ is a cryptographic salt, c is the number of iterations and $kLen$ is the desired length of the key in bits. In WPA2, the pseudo-random function used is HMAC-SHA1, the number of iterations is 4096 and the key length is 256 bits. So, if we consider these values to be fixed constants of this function, the key only depends on a password and a salt. The SSID is used for salting. The human H enters the SSID and a password into both D and P, which can then calculate a key k. This can be seen as connecting both devices D and P to a secure network. This protocol is called WPA2_1.

In Alice & Bob notation, it looks as follows:

$$
\begin{aligned}
H &: \text{knows}(\langle P, D, pw, ssid\rangle) \\
H \overset{P}{\bullet\!\!\to} P &: \langle pw, ssid\rangle \\
H \overset{P}{\bullet\!\!\to} D &: \langle pw, ssid\rangle \\
P \to D &: senc(\langle P, D\rangle, kdf(pw, ssid)) \\
D \to P &: senc(\langle D, P\rangle, kdf(pw, ssid))
\end{aligned}
$$

The function $kdf(pw, ssid)$ stands for the key derivation function PBKDF2, where PRF, c and $kLen$ are fixed. In particular, it could stand for WPA2.

Analogous to SPEKE2, there is an alternative version of this protocol, which we call WPA2_2: P shows an SSID and a password securely to H, who forwards them to D after which the protocol proceeds as before. It obviously works analogously with P and D interchanged. In Alice & Bob notation, the protocol looks as follows:

$$
\begin{aligned}
H &: \text{knows}(\langle P, D\rangle)\\
D &: \text{knows}(\langle pw, ssid\rangle)\\
P \bullet\!\!\xrightarrow{P}\!\!\bullet\ H &: \langle \text{`1'}, pw, ssid\rangle\\
H \bullet\!\!\xrightarrow{P}\!\!\bullet\ D &: \langle \text{`2'}, pw, ssid\rangle\\
P \to D &: senc(\langle P, D\rangle, kdf(pw, ssid))\\
D \to P &: senc(\langle D, P\rangle, kdf(pw, ssid))
\end{aligned}
$$

The downside of this protocol compared to SPEKE is that the adversary can perform an offline brute-force attack by calculating keys from many different passwords and trying to use them to decrypt a message sent over the channel. In the SPEKE protocols, the adversary can only make one password guess per protocol run. That is why in our model, where the adversary is eventually able to guess the password correctly, the protocol does not satisfy confidentiality of the key. It does satisfy non-injective agreement though.

Note that in the two presented protocols pw and $ssid$ are sent at the same time. On [12], the Tamarin theory files corresponding to these protocols are denoted with the suffix '_simultaneous'. There are variants, where these terms are sent one after another (therefore being able to land at different runs on the devices), but they satisfy the same security properties. Their Tamarin theory files are denoted with the suffix '_sequential'.

6.3 Nest Thermostat

The Nest Thermostat is a thermostat by Nest Labs Inc., which can be connected to a smart phone in order to be able to control the temperature in the home from a smart phone app via Wi-Fi or even remotely. We are focusing on the case where the phone and the thermostat communicate via Wi-Fi. In order to secure the communication between the smart phone and the thermostat, the user has to have a secure Wi-Fi network whose password he needs to type into the thermostat (which includes the functionality of a keyboard on its display) in order to let it join the secure network [9]. Since the user needed to have typed the password into the phone at some point earlier as well, we can view the whole protocol as a key derivation protocol (see Section 6.2), where the human types the password (and if the devices did not find it already on their own, the SSID) securely into both devices D and P, which then use a key derivation function to get a shared key k, which they can use for transmitting messages securely. The thermostat corresponds to D and the phone to P.

In our model, the key derivation protocol does not guarantee confidentiality of the key (as explained in Section 6.2), because the password is guessed by the

adversary after the protocol has finished. In the real world, depending on the security of the password chosen by the human, the adversary might also be able to guess the password correctly. Also, if the Wi-Fi network is used by other devices as well, there might be a compromised device in the system which could leak the password to the adversary. With the password, the adversary can enter the network and therefore read and modify all messages sent between D and P, breaking security. So, the protocol is only as secure as the Wi-Fi network itself. In order to avoid attacks over a compromised device, the human would have to send a different, fresh password to both the thermostat and the phone. With this password, they could generate a new key they could use for secure communication.

6.4 Philips Hue

Philips Hue [10] is a lighting system by Philips. The system mainly consists of a bridge and some light bulbs which can talk to each other. The user can send commands to the bridge (such as changing the color of the light bulbs or turning them on and off) for example via an app on the smart phone. The bridge is connected by a wire to the router, while the phone is connected wirelessly. The situation is therefore similar to the situation of the Nest Thermostat: The communication between the bridge and the phone is as secure as the network they are in. In order to send commands to the bridge, the phone needs to have a token whitelisted by the bridge. The phone has to send the token to the bridge in a time frame of 30 seconds after the human has pressed a button on the bridge. If this was the case, the token is whitelisted and the phone can send commands to the bridge using the token. However, if the network is compromised, obviously the adversary can find out the token and therefore is able to insert any commands and obviously also read any commands the phone sent.

6.5 Nest Protect

Nest Protect is a smoke detector by Nest Labs Inc., which can be controlled by a smart phone. As opposed to the Nest Thermostat, Nest Protect does not have a display or a functionality to type in a password. Instead, Nest Protect joins the network in the following way (as usual, D denotes the smoke detector and P the phone) [8]: On the back of Nest Protect, there is a QR code which the phone camera can read to get some initial information about the device. Alternatively, the human can also type a short string next to the QR code into the phone. Next, the smoke detector creates an insecure, temporary ad-hoc Wi-Fi network which the phone connects to. As soon as the phone has connected to the network, it sends the SSID and the password of the home Wi-Fi network over this network to the smoke detector which then joins the home Wi-Fi network. After this setup phase is finished, the phone also connects back to the home Wi-Fi network. Therefore, both devices D and P are connected to a secure network where they can now send messages to each other.

However, there is a security problem with this protocol, namely the fact that

the password of the home Wi-Fi network is sent insecurely from P to D. If we modeled the protocol in Tamarin, confidentiality and non-injective agreement of the key would obviously not be satisfied, since the adversary is able to change or get the password and therefore has full access to the network. In reality, the ad-hoc network is open only about a minute or two (during the setup process), so in order for the adversary to break the security of the system, he would need to log into the network in exactly that short time frame, which lowers the probability of such an attack happening when setting up Nest Protect. However, it is still a possibility and therefore, this way of connecting Nest Protect to the home network is insecure.

Chapter 7

(Im)possibility Results in General Topologies

Up to now, we have dealt with impossibility and possibility results in a topology consisting of three participants, including a human. Examples of possibility results are just protocols themselves and examples of impossibility results are Lemmas 3 and 4. In this chapter, we want to find an algorithm which, for a general topology $\tau = (V, E, \eta, \mu)$ consisting of an arbitrary number of nodes, finds out what the strongest channels a protocol could provide between two designated nodes A and B are. In a second step, we want to extend the algorithm to also output a protocol providing these channels.

The devices are connected to each other by channels in $C_{net} = \{\rightarrow, \bullet\!\!\rightarrow, \rightarrow\!\!\bullet, \bullet\!\!\rightarrow\!\!\bullet\}$, so, unrestricted channels, which are either insecure, authentic, confidential or secure, as defined in Section 2.2. Formally, that means that $\mu = E \rightarrow C_{net}$.

The definitions of what it means that a protocol *provides* insecure, authentic, confidential or secure channels between two nodes are found in [1] and are analogous to the definitions in Section 2.3 (insecure channels without any other guarantees are called *communication channels* in these definitions in [1]). If a certain channel can be provided by some protocol between two nodes, we call this channel *providable* between these two nodes. We denote the set of the four providable channels mentioned above by P_{net}.

The four channels in C_{net} can be ordered by channel strength. Channel c_1 is stronger than channel c_2 if for every protocol P that provides a channel from P_{net}, the protocol obtained by replacing c_2 by c_1 provides the same channel. The secure channel is the strongest, the insecure channel is the weakest and authentic and confidential channels are incomparable with each other. The order of the channels by channel strength is seen in a Hasse diagram in Figure 7.1.

We assume that the devices have enough computing power to perform cryptography, so, their capabilities are unrestricted. Formally, $\forall C \in V : \eta(C) = \Sigma$. Note that this implies that there is no human involved in this setting. We also

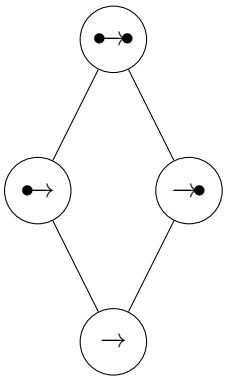


Figure 7.1: Hasse diagram of channel strengths

assume that all agents executing the roles are honest. First, we are going to look at the case where the devices have no shared initial knowledge, afterwards we are going to add that.

We define an equivalence relation $\approx_{A,B}$ on topologies $\tau = (V, E, \eta, \mu)$, where two topologies τ and τ' are equivalent, denoted as $\tau \approx_{A,B} \tau'$, if they both contain the two designated nodes A and B and if the strongest providable channels between A and B are the same. The protocols providing these channels can be different.

7.1 Equivalent Channels

Before presenting the algorithm, let us consider how many different equivalence classes of direct channels between two nodes C and D we have to consider. Direct channels between nodes C and D are equivalent, if the topologies consisting only of C, D and these channels are equivalent with respect to the relation $\approx_{C,D}$.

Secure channels in both directions are equivalent to a secure channel in one direction and any channel from C_{net} in the other direction. The reason is that we can always send a fresh key over the secure channel in order to make the other channel secure. So, if we have a secure channel in one direction and any channel in the other direction, we can just consider both of the channels to be secure channels.

If we have authentic channels in both directions, one of the nodes can send its public key to the other node, which can then send messages securely by encrypting them with the public key. The transmission is authentic by the channel assumption and also confidential because of the encryption, therefore secure. Since one direction is secure, by the argument from above, both are.

If we have confidential channels in both directions, one of the nodes can send a symmetric key to the other one, which can then send messages securely by encrypting them with this key. The transmission is confidential by the channel assumption and also authentic because the key is confidential and therefore the

encryption with this key authenticates the message. Since one direction is secure, both are.

Having an authentic channel from C to D and a confidential channel from D to C (so, the dots on the arrows are both on C's side) is equivalent to a setting where one of the channels is insecure. If the confidential channel is replaced by an insecure one, C can just send its public key to D authentically and D then encrypts everything sent over the insecure channel with it, therefore making the channel confidential. If the authentic channel is replaced by an insecure one, D can send a symmetric key confidentially to C which can then use it to encrypt messages sent over the insecure channel, making it authentic.

Therefore, if we have channels in both directions, we have to consider only four different possibilities: Either both channels are secure (both channels have dots on both sides, depicted as $\bullet\!\!\not\leftrightarrow\!\!\bullet$) or there is an authentic channel from C to D and a confidential channel from D to C (both channels have a dot on C's side, depicted as $\bullet\!\!\not\leftrightarrow$) or there is a confidential channel from C to D and an authentic channel from D to C (both channels have a dot on D's side, depicted as $\not\leftrightarrow\!\!\bullet$) or both channels are insecure (both channels have no dots, depicted as $\leftrightarrow$). Additionally, there are also the four cases of single channels from C to D ($\rightarrow$, $\bullet\!\!\rightarrow$, $\not\rightarrow\!\!\bullet$ and $\bullet\!\!\not\rightarrow\!\!\bullet$) and the four cases of single channels from D to C ($\leftarrow$, $\leftarrow\!\!\bullet$, $\bullet\!\!\not\leftarrow$ and $\bullet\!\!\not\leftarrow\!\!\bullet$).

Remark 1. Whenever we merge nodes or create new channels during the steps described below, it might happen that we get into a case where we get two channels in the same direction between two nodes. If one channel is stronger than the other, we can just forget about the weaker one. If there is an authentic and a confidential channel, we can replace them by a secure channel by simulating the sending of a message m over the secure channel by sending a key k over the confidential channel and $\{m\}_k$ over the authentic channel. Also, if we get channels in two directions which are equivalent to stronger channels, we can replace them by their stronger equivalents. For example, if we get a secure channel from C to D and an insecure channel from D to C, we can replace the insecure channel by a secure one.

7.2 Algorithm

Let us first consider the case where there is no shared initial knowledge between the nodes. We are now given such a topology and we want to find out which channels in P_{net} are the strongest providable channels between two designated nodes A and B. The first part of the algorithm consists of simplifying the topology by performing steps which keep the topologies equivalent, but remove some unnecessary intermediate nodes. The second part consists of an iterative procedure using the remaining nodes to deduce which are the strongest providable channels between A and B.

7.2.1 Simplification Steps

We first want to look at steps which simplify the topology but keep it equivalent (with respect to the relation $\approx_{A,B}$) to the original one. We begin with some

trivial deletions of intermediate nodes (nodes other than A or B) together with their adjacent edges. Note that deleting honest nodes trivially never strengthens the strongest providable channel between A and B, because any protocol providing a channel in the new topology can obviously be used in the old topology (which is a supergraph of the new one) as well.

Trivial Steps

There are two trivial steps where we can just delete nodes: If an intermediate node C is connected only to one single other node D (regardless with which channels or in which direction), we can just delete it. The reason is that everything the node sends to D is composed of fresh values, public values or values that it got from D (since this is the only connection it potentially has). But since all these terms might as well just be created by D itself, C is completely useless and all protocols using C can be trivially transformed into protocols not using C. So, by deleting C, we do not weaken the strongest providable channels between A and B.

The other case is when an intermediate node C only has incoming and no outgoing channels. Obviously, in any protocol, a step where something is sent to C is completely useless and can therefore just be omitted. C is therefore not needed for any protocol and can be deleted without weakening the strongest providable channels between A and B. We can even extend this case and delete all intermediate nodes which neither have a path to A nor a path to B. Again, sending any information to such a node is useless and therefore, the node can be deleted.

Merging Nodes

We are now introducing an important concept, which we will use for the analysis of other channel types later, namely the merging of two nodes with a bidirectional secure channel between them.

If we have two nodes C and D which have secure channels in both directions between each other ($\bullet\!\!\leftrightarrow\!\!\bullet$), we can merge them into one node. Each channel adjacent to one of the nodes stays the same but is now adjacent to the new merged node. This keeps the topology equivalent to the old one, since C and D could share anything between them securely and also use the adjacent channels of one another, since they could just always send the message securely to the other node or receive it securely from the other node. Remember that after the merge, there might be cases such as two channels from the new node to another node. Such issues can be resolved as described in Chapter 7.1.

Note that this step could theoretically be done at any point in our algorithm where we find such nodes with a bidirectional secure channel between them. However, in Chapter 7.4, when we define rules to directly construct a protocol, the merging step would make things too complicated. So, if we only want to know about the existence of a protocol, we might merge nodes, but if we want to find the protocol itself, then the algorithm does not perform this optional step.

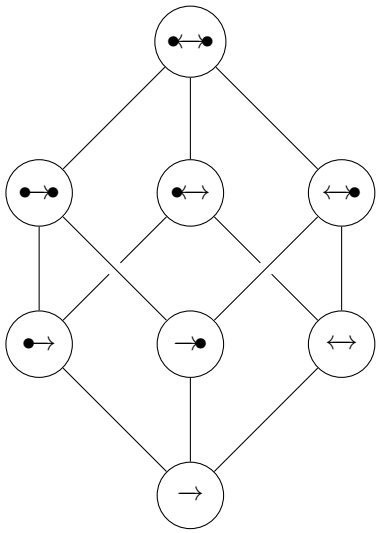


Figure 7.2: Hasse diagram of channel strengths including bidirectional channels

7.2.2 Basic Steps on Three Nodes

In this section, we are going to look at basic steps which we can perform on a topology of three nodes C, D and E. The basic idea is that we are given the channels between C and E as well as between D and E and we want to see what are the strongest channels between C and D we can derive from this while still keeping the topology with the derived channel and the topology without the derived channel equivalent with respect to the relation $\approx_{C,D}$. Later, we are going to perform these steps in our iterative procedure.

Relay Nodes

Let us look at what happens if the node E has at least an incoming and an outgoing channel (such nodes will be referred to as relay nodes). Because of this condition, we definitely have a path from C to D via E or a path from D to C via E. Because of symmetry, let us just assume without loss of generality that there is a path from C to D via E. For the channels between C and E, we have eight possibilities: There are four possibilities (insecure, authentic, confidential and secure) of single channels from C to E and four possibilities of bidirectional channels (no dots, dots on C's side, dots on E's side or dots on both sides). The same eight possibilities we get between E and D, so $8^2 = 64$ in total. Our goal is now to derive channels between C and D given the channels between C and E and between E and D. We therefore want to find out for each of the 64 possibilities which channels between C and D we get.

The eight possible channels ordered by channel strength actually form a lattice. The Hasse diagram is shown in Figure 7.2. Note that it corresponds to the three-dimensional cube.

First, consider the case where between C and E, there are secure channels in

	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow\!\!\bullet$	$\bullet\!\!\rightarrow\!\!\bullet$	$\leftrightarrow$	$\bullet\!\!\leftrightarrow$	$\leftrightarrow\!\!\bullet$	$\bullet\!\!\leftrightarrow\!\!\bullet$
$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$
$\bullet\!\!\rightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$
$\rightarrow\!\!\bullet$	$\rightarrow$	$\rightarrow$	$\rightarrow\!\!\bullet$	$\rightarrow\!\!\bullet$	$\rightarrow$	$\rightarrow$	$\rightarrow\!\!\bullet$	$\rightarrow\!\!\bullet$
$\bullet\!\!\rightarrow\!\!\bullet$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow\!\!\bullet$	$\bullet\!\!\rightarrow\!\!\bullet$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow\!\!\bullet$	$\bullet\!\!\rightarrow\!\!\bullet$
$\leftrightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
$\bullet\!\!\leftrightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\leftrightarrow$	$\bullet\!\!\leftrightarrow$	$\leftrightarrow$	$\bullet\!\!\leftrightarrow$
$\leftrightarrow\!\!\bullet$	$\rightarrow$	$\rightarrow$	$\rightarrow\!\!\bullet$	$\rightarrow\!\!\bullet$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow\!\!\bullet$	$\leftrightarrow\!\!\bullet$
$\bullet\!\!\leftrightarrow\!\!\bullet$	$\rightarrow$	$\bullet\!\!\rightarrow$	$\rightarrow\!\!\bullet$	$\bullet\!\!\rightarrow\!\!\bullet$	$\leftrightarrow$	$\bullet\!\!\leftrightarrow$	$\leftrightarrow\!\!\bullet$	$\bullet\!\!\leftrightarrow\!\!\bullet$

Table 7.1: Resulting channels in a relay situation

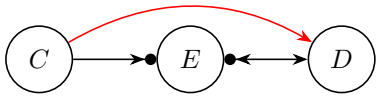


Figure 7.3: Example of a basic relay step

both directions ($\bullet\!\!\leftrightarrow\!\!\bullet$). Here, E can be merged with C and the resulting channels between C and D are just the ones that were between E and D. In the same way, if there are secure channels in both directions between E and D, E can be merged with D and the resulting channels between C and D are the ones that were between C and E. Because secure channels in both directions are the strongest connection between two nodes, for any combination of channels between C and E and E and D, the resulting channels (after the transformation) between C and D must be the same or weaker than the resulting channels if we replace the channels between C and E with two secure channels. We know that in this case, the resulting channels are just the channels between E and D. So, for any combination of channels, the resulting channels must be the same or weaker than the channels between E and D. Obviously, we can analogously find out that the resulting channels must be the same or weaker than the channels between C and E. So, to sum up, for any combination of channels between C and E and between E and D, the resulting channels between C and D are at most the greatest lower bound of the channels between C and E and between E and D. We can easily see that we trivially achieve the greatest lower bound by just relaying everything sent from C to E on to D (and the other way round if there is a path in both directions).

In Table 7.1, we can see the resulting channel between C and D for all combinations of channels between C and E and between E and D. Figure 7.3 shows an example of such a basic step where there are certain channels between C and E as well as between E and D and we get the red channel from C to D as a result.

Note that if we are only interested in the existence of a protocol and not the protocol itself, we might also delete such a relay node E after having derived the channel between C and D, if, in the whole topology, it is only connected to C and D. But in general, we keep the node E and its channels in the system.

Key Distribution Nodes

Let us have a look at nodes E with only outgoing channels to C and D and no incoming channels. First of all, if there is no channel between C and D, the node E does not help and we can just leave the system as it is.

If the channels from E to C and from E to D are both authentic, then we cannot use any information sent from E to C or D and we can just leave the system as it is. The reason is that if we add channels from D and from C back to E (these channels can be seen as confidential by the channel equivalencies in Section 7.1), we are in a situation where E is a relay node. If we look at which resulting channels are constructed between C and D, we see that these are only insecure channels in both directions. So, also when we get rid of these channels back to E and return to the situation where E only has outgoing channels, this is the best we can achieve (we deleted channels, so we certainly cannot achieve anything better). So, E cannot increase the security of the channels between C and D and obviously cannot create new channels from C to D or from D to C, so it does not change the system.

For the case where the two outgoing channels from E are confidential or the case where we have a secure channel from E to C and an insecure channel from E to D (or vice versa), we can apply the same reasoning to deduce that also in these cases, E does not help. This obviously also applies to all cases where the channels are weaker than in the mentioned cases, so, all cases where at least one of the channels is insecure.

Let us look at the case where there is a confidential channel from E to C and an authentic channel from E to D. Here, E can send a secret key via the confidential channel to C and the corresponding public key via the authentic channel to D. If there is a channel from D to C, we can get a confidential channel from D to C: D just encrypts the message it wants to send to C with the public key (since the public key was sent authentically, D knows that at most C can decrypt the message) and sends it to C. Since the message is encrypted, the adversary does not get any information about the message and therefore, we can view this as a confidential channel from D to C. If there is a channel from C to D, we can get an authentic channel from C to D: C just signs the message it wants to send to D with the secret key and sends it to D. D knows that only C could have signed the message and so the channel is authentic. In other words, we can add a dot on C's side to the channels between C and D.

Now, we want to prove that we cannot get stronger channels than these. If we add channels from D and C back to E, we can use E as a relay node and see that as resulting channels between C and D, we exactly get a confidential channel from D to C and an authentic channel from C to D. If we remove the channels incoming to E again, we certainly cannot get more than these channels between C and D. Since E does not create any channels between C and D, obviously, these channels are only obtained if channels in the respective directions are already available.

Therefore, the following result is obtained.

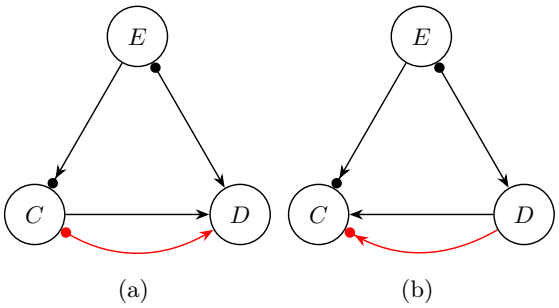


Figure 7.4: Two examples of upgrading a channel by using a public/secret key pair

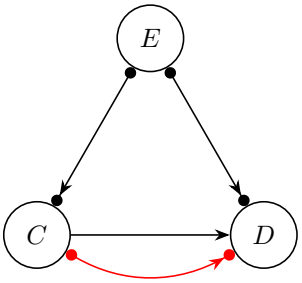


Figure 7.5: Example of upgrading a channel by using a shared key

Lemma 5. *If we have a node E which only has a confidential channel to C and an authentic channel to D, we can add dots on C's side to all channels between C and D. In more detail, we can get an authentic channel from C to D if there is an insecure channel from C to D (seen in Figure 7.4a) and a confidential channel from D to C if there is an insecure channel from D to C (seen in Figure 7.4b). Note that if we already have a confidential channel from C to D, we even get a secure channel from C to D, analogously, if we already have an authentic channel from D to C, we get a secure channel from D to C.*

If we replace either the confidential channel from E to C or the authentic channel from E to D (but not both) by a secure channel, it is clear that we can still use the same protocol (sending the public and the secret key). Also, the proof works in exactly the same way and gives the same result. We again get exactly the same channels.

If the channels from E to C and from E to D are both secure, then we can just send a key securely to both of them, which they can use for encryption. We insert a secure channel from C to D if there is a channel from C to D and a secure channel from D to C if there is a channel from D to C. This step is visualized in Figure 7.5. Since a secure channel is the strongest channel, we already know that we cannot achieve anything stronger.

This covers all possible cases (obviously, C and D are symmetric, so we can always switch them) for channels from E to exactly two nodes. Together with the relay nodes, we found all useful three-node-settings where we can upgrade

or create channels between two nodes using the channels these two nodes have with the third one. Obviously, in case E only has incoming channels from C and D, we have no way of upgrading or creating any channels between C and D.

7.2.3 Iterative Procedure

After the simplification steps, we are left with a topology where all nodes (except maybe A and B) are adjacent to at least two other nodes and have at least one outgoing channel. Furthermore, all nodes have a path to either A or B.

Our procedure now works as follows: For every node E in the system, we go over all pairs of neighbors C and D and for each of those pairs, we are looking at only the topology consisting of C, D and E. We are then in one of the cases described above: Either, E has only incoming channels from C and D (in this case we cannot do anything) or E is a relay node between C and D or it has only outgoing channels to C and D. In the latter two cases, we apply the basic steps from above to upgrade or create channels between C and D.

After we have iterated over all nodes and for each node over all pairs, we look at each pair of nodes again and, if possible, upgrade the channels according to the channel equivalencies from Section 7.1. After that, we end up with a topology with more and stronger channels. We take this new topology and repeat the whole process again, iterating over all nodes. We repeat the process until finally, a fixed point is reached, where the topology is not changed anymore after having iterated over all nodes. In this case, the procedure is finished.

The complete algorithm is therefore summarized as follows.

Algorithm 1. *A topology $\tau = (V, E, \eta, \mu)$ with two designated nodes $A, B \in V$, $\forall C \in V : \eta(C) = \Sigma$ and $\mu = E \to C_{net}$ is given as an input. After applying the simplification steps, the iterative procedure from above is run on the remaining topology until a fixed point is reached. The algorithm then outputs the channels between A and B in the fixed point.*

We need to prove that the channels output by Algorithm 1 are the strongest providable channels in P_{net} between A and B in the original topology.

First of all, we need to prove that the fixed point and the original topology are equivalent. Note that the direct channels between A and B in the fixed point are trivially providable between A and B: The protocol just sends the message over the direct channels and therefore has provided channels with the security property of the direct channels. Therefore, if the original topology is equivalent to the fixed point, the channels are also providable in the original topology. We are not going to prove this in full detail. Basically, we need to ensure that the local transformation steps (which keep the small subtopologies of two or three nodes equivalent) can be composed properly to ensure equivalence of the whole topologies. This problem is tackled in more detail in Section 7.4.

The question remains whether these are the *strongest* providable channels. If we get a secure channel from A to B, from B to A or bidirectionally (depend-

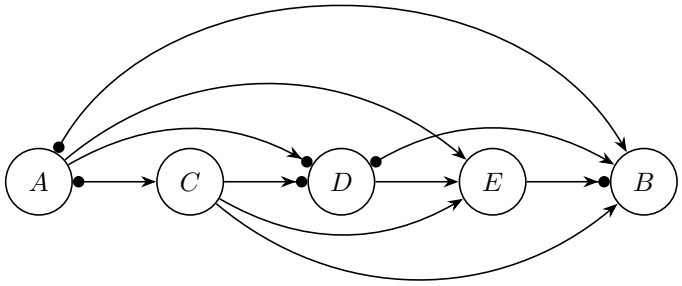


Figure 7.6: Example of a totally ordered topology with five nodes

ing on the existence of paths in a certain direction between A and B), we are already sure that we cannot provide anything stronger. It becomes interesting when we only have weaker channels between A and B in the fixed point.

We introduce the following terminology regarding dots. A dot is called *outgoing* from some node C if it belongs to an authentic or secure channel outgoing from C. It is called *incoming* to node C if it belongs to a confidential or secure channel incoming to C. Both incoming or outgoing dots can also be called dots *adjacent* to C.

We are going to prove the correctness of the algorithm on different topologies, beginning with simple ones and becoming more and more general until we cover all topologies.

Totally Ordered Topologies

Consider the case where there are only channels in one direction and no cycles in our topology. Furthermore, the nodes and the directed channels form a total order. We run our procedure on this topology and get a fixed point, which is a complete directed graph. All nodes can be drawn from left to right in one line where there is a channel from a node C to a node D if and only if C is left of D. An example with five nodes is seen in Figure 7.6. So, the nodes form a totally ordered set and the graph is complete. Consider A to be the leftmost and B to be the rightmost node. We want to prove the correctness of the algorithm, stated in the following lemma.

Lemma 6. *If the fixed point found by Algorithm 1 is a totally ordered topology which is fully connected, then the channels output by the algorithm are exactly the strongest providable channels in P_{net} between A and B in the topology given as an input to the algorithm.*

Proof. Assume that in the fixed point, there is no dot on B's side on the channel from A to B (so, the channel is insecure or authentic). To prove that we cannot provide a stronger channel, we are going to merge some nodes, which can only make the strongest providable channel between A and B stronger. So, if we merge all nodes until we are left with only A and B and there is no dot adjacent

to B, we know that no stronger channel is providable.

Consider all nodes which have a confidential channel to B. From these, choose the one closest to B and call it C. If C is now merged with B, the dots adjacent to C will be adjacent to B as well. However, note that C has no authentic outgoing channels (and therefore no outgoing dots), because if it had one to some node D, we can use it together with the confidential channel from C to B to create a confidential channel from D to B. However, since C was the closest node with a confidential channel, this is not possible. If C has any confidential incoming channels from some node E, we know that from the confidential channels from E to C and from C to B, the procedure already derived one from E to B. Therefore, B already has a confidential incoming channel from E and does not get a new one when merging with C. To sum up, by merging C and B, B does not get any new adjacent dot. The process is then repeated, every time choosing the nearest one from the remaining ones, merging it with B and therefore getting rid of a dot. At some point, we have to reach a point where B has no more adjacent dots, because there certainly is at least one node (namely A) from which the channel incoming to B has no dot on B's side. Then, we can merge A with all other nodes still between A and B. It is clear that in all these merges, we never get any new dot on B's side. We therefore end up with a topology of only A and B with B having no adjacent dots and have therefore proven that we cannot provide any stronger channels.

If the algorithm found that there is no dot on A's side on the channel from A to B (so, the channel is insecure or confidential), we apply an analogous strategy. Basically, we do the same thing except with authenticity and confidentiality interchanged. We consider all nodes having an authentic channel to B and choose the one closest to B from those. By an analogous argument, we get that we do not add any other authentic channels to B when merging C with B. So, at some point, all channels incoming to B are not authentic and therefore, B will never get anything authentically. We can argue that we can merge all other nodes into A without changing this fact and we are therefore left with a topology consisting of only A and B with A having no adjacent dots. Again, we have proven that we cannot provide any stronger channels. □

Topologies With Cycles

Let us again consider that the fixed point of the procedure is a fully connected topology. Furthermore, it is totally ordered except for some cycles where several nodes are all connected to each other bidirectionally. Note that still, all nodes lie on a path between A and B. We want to prove the correctness of Algorithm 1 using the following lemma.

Lemma 7. *If the fixed point found by Algorithm 1 is fully connected and totally ordered except for some cycles, then the channels output by the algorithm are exactly the strongest providable channels in P_{net} between A and B in the topology given as an input to the algorithm.*

Proof. In all cycles, we just define an order between the nodes and again draw the topology in one line. The order can be arbitrary up to the restriction that A

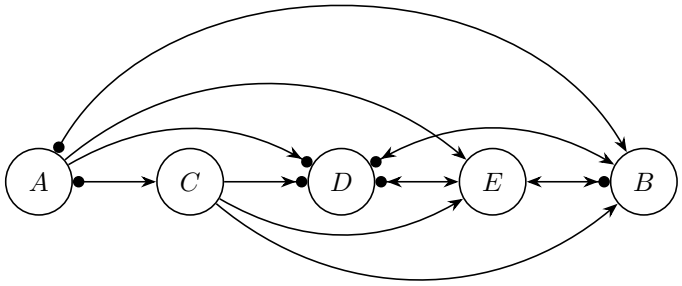


Figure 7.7: Example of a topology with cycles

still needs to be the leftmost and B the rightmost node. The proof then works in exactly the same way as the proof of Lemma 6. Note that if there are paths from A to B and from B to A, all channels are bidirectional and we obviously get a bidirectional channel between A and B as well. This does not change the location of the dots though. □

Figure 7.7 shows an example with five nodes and a cycle between D, E and B. We could have drawn D and E the other way round as well.

Not Fully Connected Topologies

Now, let us consider an even more general case where all nodes lie on a path from A to B or from B to A, including graphs not fully connected after applying the procedure. An example of such a topology is shown in Figure 7.8.

Lemma 8. *If in the topology given as an input to Algorithm 1, all nodes lie on a path from A to B or from B to A, then the channels output by the algorithm are exactly the strongest providable channels in P_{net} between A and B in the topology given as an input to the algorithm.*

Proof. If we have cycles in such a system, we can again order the nodes in a cycle arbitrarily. We therefore get a partial order on the nodes.

If we try to apply the same arguments as for the total order case, we notice that B does not always have a single closest node, it might have several closest nodes instead. The obvious approach is to just pick one of them arbitrarily and merge it first. Again, such a merge does not add any other confidential or authentic (depending on which property we consider) channel to B and we can use the same argument to see that the algorithm is correct also in this case. □

General Topologies

Now that we have proven the correctness of the algorithm for several special cases, let us prove the general theorem, which says that the protocol works correctly on all topologies.

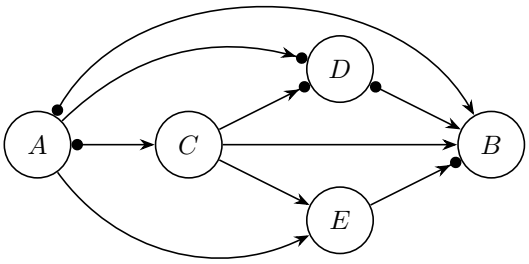


Figure 7.8: Example of a not fully connected topology where all nodes lie on some path between A and B

Theorem 1. *The channels output by Algorithm 1 are exactly the strongest providable channels in P_{net} between A and B in the topology given as an input to the algorithm.*

Proof Sketch. Until now, we looked at all possible topologies where all nodes lie on some path from A to B or from B to A. We call the union of all these nodes the *core* of A and B, denoted as $core(A, B)$. However, there might be some nodes which are outside of the core. If such a node only has incoming paths from the core, it never has any path back (otherwise, it would be part of the core) and therefore, it has no path to either A or B. We already saw that such nodes can initially be deleted in the simplification steps, since they are of no use to any protocol. So, there are no channels from the core to the outside. What is left are nodes which only have outgoing paths to the core. Such a node C could send keys to the core which can then be used to upgrade a channel. However, we still have to prove that the channels C has (in the fixed point) to nodes in the core are the strongest providable channels (otherwise, it might appear that it cannot send a key while in reality it could). Note that we can see such a system between C and some node D inside the core again as an instance of a topology where we applied the procedure in order to get a channel from C to D. We now want to prove for each D (inside the core) that C is connected to that we cannot get any stronger channel from C to D. This is actually the same problem as our original one, just with different designated nodes. C and D obviously have a core itself, namely all nodes that lie on some path from C to D, denoted as $core(C, D)$.

So, for the path between C and D, there might again be a node outside of $core(C, D)$ which could send keys to it. Note that nodes from $core(A, B)$ can only send keys to nodes which are still inside this core, because there are no channels from the core to the outside. So, nodes of $core(A, B)$ might upgrade channels lying somewhere between C and D only if those channels are inside $core(A, B)$, but these are channels we already dealt with. So, a node outside of $core(C, D)$ which could send keys to some nodes inside $core(C, D)$ and which we have not yet considered cannot be part of $core(A, B)$. Instead, it needs to be outside both cores.

In Figure 7.9, we can see such a recursive situation. We have nodes A and

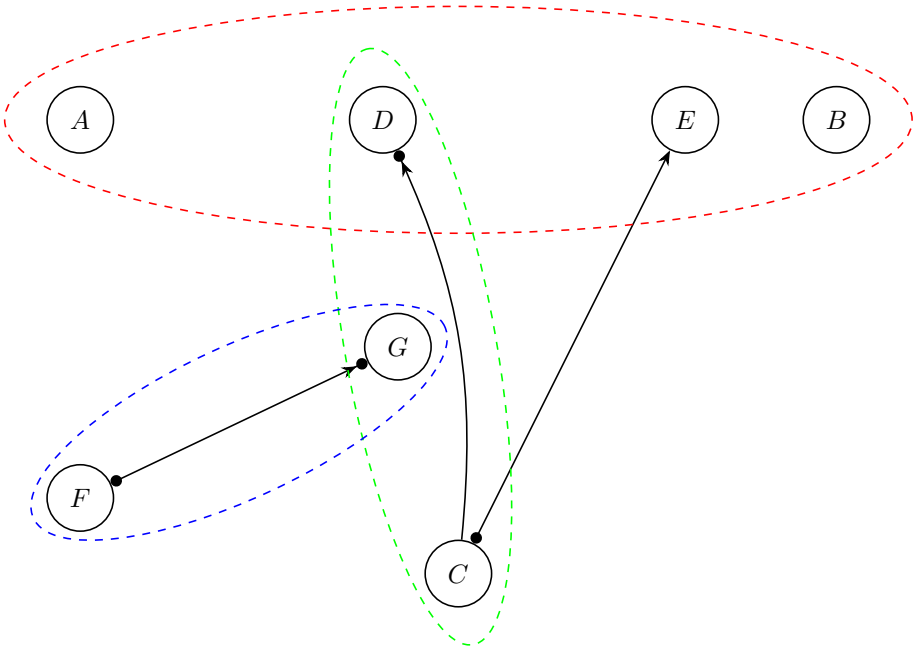


Figure 7.9: Cores of different recursion levels

B, whose core is depicted as a dashed red ellipse. There is a node C which has some channels to $core(A, B)$, say, to nodes D and E (and maybe to others as well). For all these channels, we again have a core (in the figure, $core(C, D)$ is depicted in green), where there can again be a node F having some channel to some node G inside this core. The nodes F and G again have a core (depicted in blue) and so on.

The problem we have is therefore recursive. By going over all possible recursive branches, we get a recursion tree where a vertex represents a pair of nodes in our topology of which we want to calculate the strongest channels between them. To avoid confusion, 'vertices' denotes the nodes of the recursion tree and 'nodes' denotes the nodes of the topology. In Figure 7.10, the (incomplete) recursion tree corresponding to the (incomplete) recursion from Figure 7.9 is depicted. In all recursive steps, we have to consider only nodes which are not part of any core of the ancestors of the current vertex in the recursion tree. Also, in each step, at least one new node (namely the node outside of the core of which we want to analyze the paths to the core) is added to the union of the cores. Since we have a finite number of nodes, the recursion is bound to stop at some point where we have two nodes we want to analyze and all other relevant nodes are inside the core of the two nodes (the corresponding vertices are leaves in the recursion tree). We have proven the correctness of the algorithm in these cases in Lemma 8 already, so we can be sure that the channel between the two considered nodes is the strongest providable one. In the topologies corresponding to the parents of the leaves in the recursion tree, we can therefore delete the cores of all their children and just consider their core, again corresponding to a situation where Lemma 8 can be applied. Recursively, we can apply the same

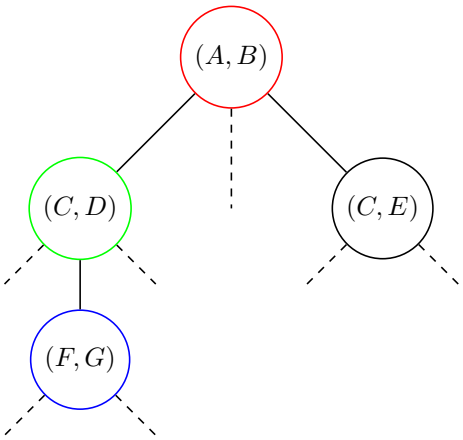


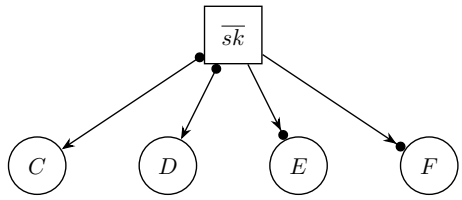
Figure 7.10: Recursion tree

steps until we are at the root of the recursion tree, which corresponds to our initial situation with A and B being the designated nodes and where all nodes not in $core(A, B)$ are deleted. We can therefore just apply Lemma 8 to this topology. Thus, the algorithm also works for general topologies. □

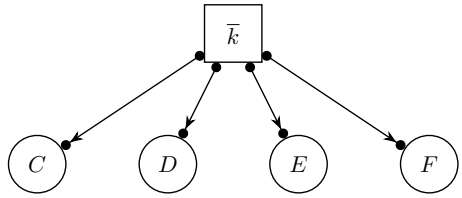
7.3 Adding Shared Initial Knowledge

Let us try to find a way to extend the communication topology model in order to represent private shared initial knowledge between several nodes. We consider two types of shared knowledge: Some nodes might share a key which they can use to encrypt messages and send them securely among each other. In the other case, some nodes know a secret key and other nodes know the corresponding public key, which they can use to send messages to those nodes confidentially, while the nodes knowing the secret key can use it to sign messages and therefore send them authentically.

We can transform these two types of shared initial knowledge into some virtual nodes by introducing for each shared key a new node having secure channels to all nodes sharing this key and for each public/secret key pair a new node having confidential channels to the nodes who are supposed to have the secret key and authentic channels to all nodes who are supposed to know the corresponding public key. This new topology can simulate the shared initial knowledge by having the virtual nodes distribute the keys. However, the topologies are not the same. If the virtual node is connected to more than two nodes, it could distribute different keys for each pair of nodes, while having shared initial knowledge among more than two nodes means that all nodes know exactly the same key. For our algorithm, it does not make any difference whether the keys are different or the same, because we do not yet consider the exact protocol and possible composability issues. So, the shared initial knowledge is still transformed into such a virtual node, which is drawn as a rectangle as an indication that it is not a real node but rather a representation of shared initial knowledge. In



(a) E and F know $\overline{sk}$, C and D know $pk(\overline{sk})$



(b) All nodes know $\overline{k}$

Figure 7.11: The two kinds of shared initial knowledge

the algorithm, it can be treated as a normal node though, upgrading channels between the nodes it is connected to. It is important that the channels which are drawn from a rectangular node to a regular node are not real channels and can never be upgraded during the procedure.

In Figure 7.11, the two types of knowledge are depicted as such rectangular nodes. The bar over k and sk will be explained later.

7.4 Deriving Security Protocols

With Algorithm 1, we can find out the strongest providable channels between A and B. However, we did not yet see how such a protocol providing them looks like. In this section, we will see how we can derive a security protocol directly from the steps the algorithm performs.

First, we are going to define some notation: Sending a message m from C to D using direct channels already given in the original topology is denoted by $i(C, D, m)$, $a(C, D, m)$, $c(C, D, m)$ or $s(C, D, m)$, depending on the security property of the channel (insecure, authentic, confidential or secure). For derived channels which are not there initially, we are going to use capital letters for the security property.

Next, we are going to define rules for the derivation of these channels. The rules are split into three categories and correspond to the steps Algorithm 1 does in the iterative procedure: First, there are rules concerning only two nodes, which model the channel equivalencies dealt with in Chapter 7.1. The second category are relay rules, corresponding to the step of relaying a message from one node to another via a third one. The third category are the rules which correspond to a setting where a node distributes keys to two other nodes.

Only Two Nodes

If we have an authentic and a confidential channel at the same time from C to D, we can send a message securely by first sending a key over the confidential channel and then the message encrypted with this key over the authentic channel. Until now, we just replaced the two channels by a secure one without further thinking about it. However, in a real protocol, it is different whether we need to do this detour or whether we just have a secure channel directly. As a rule, we write this as $S(C, D, m) = C(C, D, k) \cdot A(C, D, \{m\}_k)$, meaning that we can send an arbitrary message m securely from C to D by sending a key k confidentially from C to D and then m encrypted with k over the authentic channel. The dot between the two channels represents sequential composition, so, sending the two terms one after the other. Note that capital letters are used here for all channels, since the confidential and the authentic channel could be derived channels as well.

For the case where we have a secure channel from D to C and an insecure channel from C to D, we can send a message securely from C to D by sending a key securely from D to C and then the encrypted message from C to D. In our notation, this is written as $S(C, D, m) = S(D, C, k) \cdot I(C, D, \{m\}_k)$. The insecure channel can also be an authentic or confidential channel, in which case the rule is changed accordingly.

If we have two authentic channels in both directions, we can send a message from C to D securely by sending a public key from D to C and then sending the encrypted message from C to D. This is written as $S(C, D, m) = A(D, C, pk) \cdot A(C, D, \{m\}_{pk})$. Of course, we analogously get a secure channel in the other direction.

If we have two confidential channels in both directions, we can send a message from C to D securely by sending a symmetric key from D to C and then sending the encrypted message from C to D. This is written as $S(C, D, m) = C(D, C, k) \cdot C(C, D, \{m\}_k)$. Here, we can again get the channel in the other direction as well.

If we have an authentic channel from D to C and an insecure channel from C to D, we can send a message confidentially from C to D by sending a public key from D to C and then the encrypted message from C to D. This is written as $C(C, D, m) = A(D, C, pk) \cdot I(C, D, \{m\}_{pk})$.

If we have a confidential channel from D to C and an insecure channel from C to D, we can send a message authentically from C to D by sending a symmetric key from D to C and then the encrypted message from C to D. This is written as $A(C, D, m) = C(D, C, k) \cdot I(C, D, \{m\}_k)$.

Relay Nodes

Whenever we use a node E to relay anything from C to D, we are just forwarding the message m and use the relay rules accordingly to see what security properties we get. For example, if we have authentic channels from C to E as well as from

E to D, we can derive an authentic channel from C to D. This is written as $A(C, D, m) = A(C, E, m) \cdot A(E, D, m)$.

Key Distribution Nodes

Whenever we send keys to two channels in order to upgrade channels, we can use the following rules. If we have a node E which has secure channels to both C and D and an insecure channel from C to D, E can send a key to both nodes in order to make the insecure channel secure. The rule looks as follows: $S(C, D, m) = S(E, C, k) \cdot S(E, D, k) \cdot I(C, D, \{m\}_k)$. Of course, it also works analogously if the insecure channel was authentic or confidential.

When a node E has a confidential channel to C and an authentic channel to D, we can use them to send a secret key to C and the corresponding public key to D. If we have an insecure channel from C to D, we can send a message authentically from C to D by signing the message, seen in this rule: $A(C, D, m) = C(E, C, sk) \cdot A(E, D, pk(sk)) \cdot I(C, D, \langle m, sign(m, sk)\rangle)$. The term $pk(sk)$ denotes the public key corresponding to the secret key sk. If we have a confidential channel from C to D, we can send a message securely from C to D with this rule: $S(C, D, m) = C(E, C, sk) \cdot A(E, D, pk(sk)) \cdot C(C, D, \langle m, sign(m, sk)\rangle)$. Of course, if we replace any channel from E to C or D by a secure channel (but not both, since then we would be in the case from before again), the rules work analogously.

When the channels are the other way round, i.e., E has an authentic channel to C and a confidential channel to D, we just send the secret key to D and the corresponding public key to C. If we have an insecure channel from C to D, we can send a message confidentially from C to D with this rule: $C(C, D, m) = C(E, D, sk) \cdot A(E, C, pk(sk)) \cdot I(C, D, \{m\}_{pk(sk)})$. If we have an authentic channel from C to D, we can send a message securely from C to D with this rule: $S(C, D, m) = C(E, D, sk) \cdot A(E, C, pk(sk)) \cdot A(C, D, \{m\}_{pk(sk)})$.

Note that if we upgrade a channel, we will always keep the old channel, because it may be enough for the final protocol to use the old channel instead of the upgraded one, which then keeps the protocol simpler, because there is no need to upgrade the channel during the protocol.

So, how do we use these rules? During every step in our algorithm deriving a new channel, we are noting the rule we just used in order to remember how we derived this channel. In the end of our algorithm, we have found a rule for each such channel, particularly also for the channel from A to B or from B to A. For such a channel, we need to have a rule beginning with $S(A, B, m)$ or $S(B, A, m)$ respectively (only if the channel is secure, else we obviously have a different letter there). We then see from which channels and with which messages we derived such a channel and for each channel in this derivation, we see how we derived this one and so on. In the end, we are left with a big concatenation of basic channels (with lower case letters), together building the channel from A to B and trivially being a security protocol, the different terms (bordered by dots) being the individual protocol steps. We need to be careful with the names of the keys and messages used in the rules: Each time we go one layer deeper,

we need to make sure that any new keys introduced are renamed to avoid name clashes with keys already in use.

7.4.1 Shared Initial Knowledge

If we include topologies with shared initial knowledge, we obviously need additional rules. First of all, we want to make sure that every time a certain key coming from this knowledge is used, it is the same one. So, we need to define a different type of variable which is not (such as the ones we saw before) just a placeholder for any key, but a variable which globally needs to refer to exactly the same key whenever it is used. We denote such keys with a bar over their name, so, e.g. $\overline{k}$ for a shared key and $\overline{sk}$ and $pk(\overline{sk})$ for a public/secret key pair. We also need to define a term (used to construct channels) which denotes that a certain node knows a certain key. This is written as $knows(C, \overline{k})$, which means that node C knows key $\overline{k}$.

The rules we define correspond exactly to the rules of outgoing channels. If two nodes C and D share a key and there is an insecure channel from C to D, we can use this key to make the channel secure. The rule looks as follows: $S(C, D, m) = knows(C, \overline{k}) \cdot knows(D, \overline{k}) \cdot I(C, D, \{m\}_{\overline{k}})$. Note that here, $\overline{k}$ represents any variable of this global type. Obviously, we can also have an authentic or confidential channel from C to D and use an analogous rule.

When C knows a secret key and D knows the corresponding public key and there is an insecure channel from C to D, we can send a message authentically from C to D with this rule: $A(C, D, m) = knows(C, \overline{sk}) \cdot knows(D, pk(\overline{sk})) \cdot I(C, D, \langle m, sign(m, \overline{sk})\rangle)$. If we have a confidential channel from C to D, we can send a message securely from C to D with this rule: $S(C, D, m) = knows(C, \overline{sk}) \cdot knows(D, pk(\overline{sk})) \cdot C(C, D, \langle m, sign(m, \overline{sk})\rangle)$.

When D knows a secret key and C the corresponding public key and there is an insecure channel from C to D, we can send a message confidentially from C to D with this rule: $C(C, D, m) = knows(D, \overline{sk}) \cdot knows(C, pk(\overline{sk})) \cdot I(C, D, \{m\}_{pk(\overline{sk})})$. If we have an authentic channel from C to D, we can send a message securely from C to D with this rule: $S(C, D, m) = knows(D, \overline{sk}) \cdot knows(C, pk(\overline{sk})) \cdot A(C, D, \{m\}_{pk(\overline{sk})})$.

7.4.2 Tagging

With our algorithm, we get a protocol consisting of several sub-protocols which themselves consist of sub-protocols and so on. Each of the rules we just described is such a sub-protocol which creates some channel, e.g. by sending a key to two nodes which use it for encryption. However, just because these sub-protocols satisfy some security property, that does not automatically mean that the composed protocol also satisfies it. When the protocol uses the same encryption in several different places, the adversary could potentially use that to insert a message somewhere in a different context. That is why we want the sub-protocols to be independent. One way to achieve this is by using protocol tags. Each sub-protocol has its own unique tag which is included in the encrypted message the protocol sends. What this looks like exactly is not of great

importance. We just assume that every time a sub-protocol encrypts (symmetrically or asymmetrically) or signs a message, it instead encrypts or signs a pair consisting of its tag and the message. Every time a message is decrypted, it is first checked that the tag is really the one which was expected and after decryption, the tag is thrown away. The rules are therefore extended accordingly.

For example, the rule $S(C, D, m) = A(D, C, pk) \cdot A(C, D, \{m\}_{pk})$ is changed to $S(C, D, m) = A(D, C, pk) \cdot A(C, D, \{\langle t, m\rangle\}_{pk})$, where t is a tag unique to an instance of this rule. Similarly, the rule $A(C, D, m) = C(E, C, sk) \cdot A(E, D, pk(sk)) \cdot I(C, D, \langle m, sign(m, sk)\rangle)$ is changed, which yields the new rule $A(C, D, m) = C(E, C, sk) \cdot A(E, D, pk(sk)) \cdot I(C, D, \langle m, sign(\langle t, m\rangle, sk)\rangle)$. Such a tag is inserted in all rules where encryption or signing plays a role. Actually, these are all rules except for the relay rules.

The formal proof of why this works is beyond the scope of this thesis. However, in [2], it is shown for a different model that if we have several protocols using the same cryptographic primitives (which is the case in our system, since we always assume that all nodes use the same algorithm for encryption and signing), we can compose the protocols using the tagging described above. There, that is proven in a setting of two participants and only using symmetric encryption and considering secrecy. However, it should be easy to extend the proof to an arbitrary number of participants, more primitives such as asymmetric encryption and signing as well as authenticity as a security goal.

The theorem from [2] we want to use is stated intuitively as follows:

Theorem 2. *Assume we have a protocol P consisting of two sub-protocols P_1 and P_2, each of which is performed by a different participant in parallel. This protocol is used to establish a key between two participants. Furthermore, there is a protocol Q using a pre-shared key and consisting of two sub-protocols Q_1 and Q_2, each again performed by a different participant in parallel. Assume that this protocol Q keeps a value s secret. The protocol W is defined as the sequential composition of P_1 and Q_1 performed by one participant and the sequential composition of P_2 and Q_2 performed by the other one, both in parallel, where P_1 and P_2 are tagged differently than Q_1 and Q_2. Then, the protocol W also keeps the value s secret, given that neither P nor Q leak the key.*

Essentially, in W, the pre-shared key of protocol Q is replaced by the key sharing algorithm P.

Now, obviously, this is a proof in a different model, so it does not directly prove that the tagging works in our model. However, intuitively, we can find a correspondence between our model and the model from [2]. A rule such as $S(C, D, m) = S(E, C, k) \cdot S(E, D, k) \cdot I(C, D, \{m\}_k)$ can be translated into two protocols P and Q as follows: $S(E, C, k)$ corresponds to P_1, giving C, the first participant, the key k. $S(E, D, k)$ corresponds to P_2, giving D, the second participant, the key k. P corresponds to the two protocols being executed in parallel (in our model, they are executed sequentially, but it is easy to see that they could also be executed in parallel). $I(C, D, \{m\}_k)$ corresponds to the protocol Q, which has the goal of keeping m secret. Q_1 corresponds to C encrypting and sending m, while Q_2 corresponds to D receiving and decrypting

m. It is easy to see that the preconditions, namely that neither P nor Q reveal the key, are satisfied. Also, Q does keep m secret. By using the theorem from [2], the protocol W using the key shared by P in protocol Q (so, essentially, the whole protocol on the right side of the rule, is secure, if P and Q are tagged differently. This can be achieved by tagging the message sent with a new tag specifically for this protocol. We therefore send $I(C, D, \{\langle t, m\rangle\}_k)$ as proposed above. For all other rules using encryption or signing, we can find a similar intuitive correspondence.

7.4.3 Name Identifiers

If a relay rule which results in an authentic channel is used, we want to include the name identifier of the sender. Imagine that C and C' both send a message authentically via E to D by just relaying the messages over authentic channels. The adversary could then intercept both messages sent from E to D and interchange them such that E believes that the message coming from C actually came from C' and vice versa. That problem is solved by including the sender.

Analogously, if a relay rule which results in a confidential channel is used, we want to include the name identifier of the receiver. The reason is that there could be a potential confusion if a node E needs to relay two messages (coming from a different node C) to two nodes D and D', where the channel to D is confidential and the channel to D' is insecure. The adversary could interchange messages between C and D such that E might send a message over the insecure channel which is supposed to be confidential, since he does not know the receiver of the message. By including it, this problem is solved.

For simplicity, we include both sender and receiver in messages relayed over authentic, confidential and secure channels. For example, for two secure channels, the rule is rewritten as follows: $S(C, D, m) = S(C, E, \langle C, \langle D, m\rangle\rangle) \cdot S(E, D, \langle C, \langle D, m\rangle\rangle)$.

If a relay rule which results in an insecure channel is used, we do not need to include any name identifier, because the adversary could change anything arbitrarily anyway.

7.4.4 Tagging in Relay Rules

It is still not quite enough if the sender and the receiver are included in the messages during a relay, because if from C to D, two different messages are transmitted securely, the adversary could still interchange them, possibly without D noticing. That is why we still want to include some kind of tagging when using relay rules.

In order to see how exactly we need to tag, we introduce a general transformation of our initial topology. This transformation is only virtual and just used to reason about tagging, it is not representing the real topology. The real topology consists of channels in C_{net}. The security of these channels is not constructed by cryptographic means, but just assumed to be there. We can replace such a channel by an insecure channel and shared initial knowledge as follows:

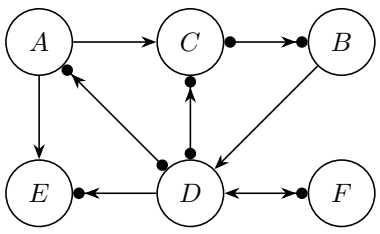
For every authentic channel between two nodes C and D, we replace it by an insecure channel and give C and D shared initial knowledge, namely a secret key to C and the corresponding public key to D. For confidential channels, we do the same, just with the secret and the public key interchanged. For secure channels, we give both channels a shared key. In any case, the channel is replaced by an insecure channel. In the end, we end up with a new topology consisting of only insecure channels and shared initial knowledge, where every channel with a stronger security property is now simulated by an insecure channel and knowledge. The two topologies are equivalent with respect to the relation $\approx_{A,B}$.

In our final protocol consisting of messages sent over direct channels (denoted with a lower case letter), all occurrences of $a(C, D, m)$ are now replaced by $knows(C, \overline{sk}) \cdot knows(D, pk(\overline{sk})) \cdot i(C, D, \langle m, sign(\langle t, m\rangle, \overline{sk})\rangle)$. Analogously, all occurrences of $c(C, D, m)$ are replaced by $knows(D, \overline{sk}) \cdot knows(C, pk(\overline{sk})) \cdot i(C, D, \{\langle t, m\rangle\}_{pk(\overline{sk})})$ and finally, all occurrences of $s(C, D, m)$ are replaced by $knows(C, \overline{k}) \cdot knows(D, \overline{k}) \cdot i(C, D, \{\langle t, m\rangle\}_{\overline{k}})$. Of course, the tags t are unique and occur only once in the whole protocol.

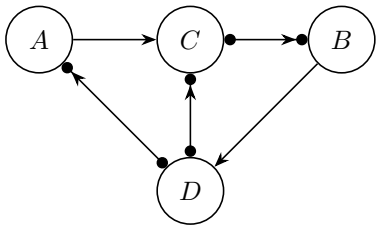
In this new topology, we do not have to think about any additional tagging when using relay rules, since all direct channels are insecure anyway, so, tags are irrelevant, since the adversary could always change them. What is different between the topologies is that now, every time we send something over such an channel secured with our virtual knowledge, we add a tag inside the encryption or signature. If we now go back to the real topology, we see that we need to insert the tag along with the message to be sent, so, for example, we would need to replace $a(C, D, m)$ by $a(C, D, \langle t, m\rangle)$. Essentially, it is enough to just let the algorithm run normally and then, in the final protocol consisting of messages transmitted over direct channels, add a unique tag to all messages transmitted over authentic, confidential or secure channels.

7.5 Example

We are going to conclude this section by an example of the whole procedure on some small topology of six nodes. The goal is to find the strongest providable channels between A and B and the corresponding protocols. The topology looks as follows:



The algorithm starts by applying the simplification steps in order to eliminate unnecessary nodes. In this particular example, it deletes E, because it has no path to either A or B and it also deletes F, because it is only connected to one other node. The simplified topology then looks like this:



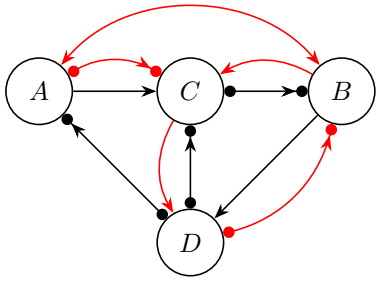
Then, the first iteration of the iterative procedure begins. Assume that the algorithm iterates over the nodes alphabetically. It considers A first, which has only one pair of adjacent nodes. Theoretically, A could serve as a relay node to create an insecure channel from D to C, but since we already have a secure one there, the algorithm does not do anything.

For B, the algorithm creates an insecure channel from C to D by relaying. This deduction is written down as $I(C, D, m) = s(C, B, \langle C, \langle D, m\rangle\rangle) \cdot i(B, D, \langle C, \langle D, m\rangle\rangle)$.

C acts as a relay node to create an insecure channel from A to B using the rule $I(A, B, m) = i(A, C, \langle A, \langle B, m\rangle\rangle) \cdot s(C, B, \langle A, \langle B, m\rangle\rangle)$. Furthermore, it can also create a secure channel from D to B by relaying, using the rule $S(D, B, m) = s(D, C, \langle D, \langle B, m\rangle\rangle) \cdot s(C, B, \langle D, \langle B, m\rangle\rangle)$.

D can create insecure channels between B and C as well as between B and A by rules $I(B, C, m) = i(B, D, \langle B, \langle C, m\rangle\rangle) \cdot s(D, C, \langle B, \langle C, m\rangle\rangle)$ and $I(B, A, m) = i(B, D, \langle B, \langle A, m\rangle\rangle) \cdot s(D, A, \langle B, \langle A, m\rangle\rangle)$. It can also upgrade the channel between A and C by sending keys to both of these nodes with the rule $S(A, C, m) = s(D, A, k) \cdot s(D, C, k) \cdot i(A, C, \{\langle t, m\rangle\}_k)$.

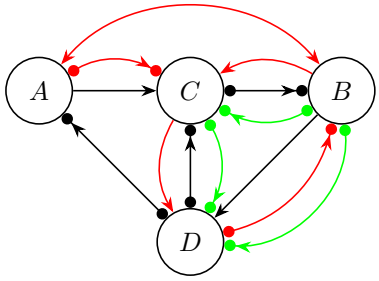
This concludes the first iteration of the algorithm. The resulting channels are drawn in red in the next figure.



Next, as a side step, some channels can be updated by using the 'equivalent channel' rules. In this case, between C and B, D and C as well as D and B, we have the situations that there is a secure channel in one direction and an insecure channel in the other. By using these rules, we can get bidirectional secure channels. Written in our notation, these rules are the following ones:

$$S(C,D,m) = s(D,C,k) \cdot I(C,D,\{\langle t,m\rangle\}_k)$$
$$S(B,C,m) = s(C,B,k) \cdot I(B,C,\{\langle t,m\rangle\}_k)$$
$$S(B,D,m) = S(D,B,k) \cdot i(B,D,\{\langle t,m\rangle\}_k)$$

The new channels are drawn in green in the next picture. Note that we leave the old ones as they are.



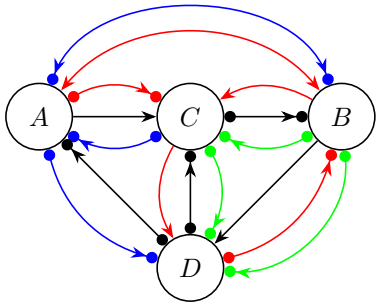
Now, the algorithm proceeds with the second iteration. Using A as a relay does not help anywhere, so it does not add any new channels there.

B however can be used as a relay channel to create insecure channels from C to A as well as from A to D with the rules $I(C,A,m) = s(C,B,\langle C,\langle A,m\rangle\rangle) \cdot I(B,A,\langle C,\langle A,m\rangle\rangle)$ and $I(A,D,m) = I(A,B,\langle A,\langle D,m\rangle\rangle) \cdot i(B,D,\langle A,\langle D,m\rangle\rangle)$.

C can be used as a relay channel to create secure channels from A to D as well as from A to B with the rules $S(A,D,m) = S(A,C,\langle A,\langle D,m\rangle\rangle) \cdot S(C,D,\langle A,\langle D,m\rangle\rangle)$ and $S(A,B,m) = S(A,C,\langle A,\langle B,m\rangle\rangle) \cdot s(C,B,\langle A,\langle B,m\rangle\rangle)$.

D can similarly be used as a relay channel to create secure channels from C to A as well as from B to A with the rules $S(C,A,m) = S(C,D,\langle C,\langle A,m\rangle\rangle) \cdot s(D,A,\langle C,\langle A,m\rangle\rangle)$ and $S(B,A,m) = S(B,D,\langle B,\langle A,m\rangle\rangle) \cdot s(D,A,\langle B,\langle A,m\rangle\rangle)$.

Note that C and D produce both channels that B produces as well, just securely instead of insecurely. So while the algorithm also creates the insecure channels, we are not going to draw them in order to keep the diagram a bit simpler. The other new channels are drawn in blue in the following figure.



Because now, we have secure channels in both directions between all nodes, we know that we cannot get any stronger than that, meaning that this must be a fixed point. From the rules we can now deduce the exact protocol for sending a message securely from A to B.

The top rule we use is the following one:

$$S(A, B, m) = S(A, C, \langle A, \langle B, m\rangle\rangle) \cdot s(C, B, \langle A, \langle B, m\rangle\rangle)$$

We see that the second channel is already an existing one. For the first one, we have to use the rule which created the secure channel between A and C and insert our message $\langle A, \langle B, m\rangle\rangle$ as the m of this rule. This looks as follows:

$$S(A, C, \langle A, \langle B, m\rangle\rangle) = s(D, A, k) \cdot s(D, C, k) \cdot i(A, C, \{\langle t, \langle A, \langle B, m\rangle\rangle\rangle\}_k)$$

Since these are all existing channels, we can put everything together and get the following protocol:

$$\begin{aligned} S(A, B, m) = \; & s(D, A, k) \cdot s(D, C, k) \\ & \cdot i(A, C, \{\langle t, \langle A, \langle B, m\rangle\rangle\rangle\}_k) \cdot s(C, B, \langle A, \langle B, m\rangle\rangle) \end{aligned}$$

The last step we need to perform is to add unique tags to all channels which are not insecure. This makes the protocol look as follows:

$$\begin{aligned} S(A, B, m) = \; & s(D, A, \langle u, k\rangle) \cdot s(D, C, \langle v, k\rangle) \\ & \cdot i(A, C, \{\langle t, \langle A, \langle B, m\rangle\rangle\rangle\}_k) \cdot s(C, B, \langle w, \langle A, \langle B, m\rangle\rangle\rangle) \end{aligned}$$

In words, A and C get the key k from D and use it to transfer the message securely over the insecure channel from A to C. The message is then sent from C to B over the existing secure channel.

In the other direction, we can send a message securely from B to A by using the following top rule:

$$S(B, A, m) = S(B, D, \langle B, \langle A, m\rangle\rangle) \cdot s(D, A, \langle B, \langle A, m\rangle\rangle)$$

The first channel is then broken down itself in the following way:

$$S(B, D, \langle B, \langle A, m\rangle\rangle) = S(D, B, k) \cdot i(B, D, \{\langle t, \langle B, \langle A, m\rangle\rangle\rangle\}_k)$$

The first message needs to be resolved as follows:

$$S(D, B, k) = s(D, C, \langle D, \langle B, k\rangle\rangle) \cdot s(C, B, \langle D, \langle B, k\rangle\rangle)$$

Putting everything together, we get

$$\begin{aligned} S(B, A, m) = \; & s(D, C, \langle D, \langle B, k\rangle\rangle) \cdot s(C, B, \langle D, \langle B, k\rangle\rangle) \\ & \cdot i(B, D, \{\langle t, \langle B, \langle A, m\rangle\rangle\rangle\}_k) \cdot s(D, A, \langle B, \langle A, m\rangle\rangle) \end{aligned}$$

Adding tags, we get

$$\begin{aligned} S(B,A,m) = \; & s(D,C,\langle u,\langle D,\langle B,k\rangle\rangle\rangle)\cdot s(C,B,\langle v,\langle D,\langle B,k\rangle\rangle\rangle) \\ & \cdot i(B,D,\{\langle t,\langle B,\langle A,m\rangle\rangle\rangle\}_k)\cdot s(D,A,\langle w,\langle B,\langle A,m\rangle\rangle\rangle) \end{aligned}$$

In words, first, D sends a key securely to B via C, then, B sends to D the message encrypted with that key insecurely and then, D sends the message securely on to A.

Chapter 8

Conclusion

We have extended the communication topology model from [1] by new channel properties (replay-free channels and invariant channels) as well as introduced two types of restrictions (small and password channels). We have then analyzed settings involving a human H, a smart device D and a smart phone P and have seen for which topologies there is a protocol which establishes a shared key between D and P and for which there is none. Considering some real-world examples of Internet of Things products, we found out that there are still security issues which need to be tackled in order to safely live in an environment where more and more devices are connected to the Internet.

Moreover, we have found an algorithm which takes a topology consisting of an arbitrary number of devices and outputs the strongest channel (insecure, authentic, confidential or secure) a protocol could provide between two designated nodes A and B in the topology. We have then extended the algorithm such that it actually finds a protocol providing such a channel.

Bibliography

[1] D. Basin, M. Schläpfer, and S. Radomirović. A complete characterization of secure human-server communication, 2015.

[2] S. Ciobâcă and V. Cortier. Protocol composition for arbitrary primitives. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, pages 322–336, July 2010.

[3] Bluetooth Special Interest Group. Simple pairing whitepaper, August 2006.

[4] IEEE. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements (IEEE Std 802.11i-2004)*. Institute of Electrical and Electronics Engineers, Inc., July 2004.

[5] D. P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, October 1996.

[6] B. Kaliski. PKCS #5: Password-Based Cryptography Specification. RFC 2898, RFC Editor, September 2000.

[7] G. Lowe. A hierarchy of authentication specifications. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE, 1997.

[8] How do I connect my Nest Protect to Wi-Fi and my Nest account? `https://nest.com/support/article/How-do-I-connect-my-Nest-Protect-Wi-fi-and-my-Nest-Account`. Accessed: March 19, 2015.

[9] A step-by-step guide to setup on the Nest Learning Thermostat. `https://nest.com/support/article/A-step-by-step-guide-to-setup-on-the-Nest-Learning-Thermostat`. Accessed: March 19, 2015.

[10] Philips Hue. `http://www.meethue.com`. Accessed: March 19, 2015.

[11] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 78–94, June 2012.

[12] Tamarin theory file repository for this thesis. `http://www.infsec.ethz.ch/education/studentProjects/former/dannys.html`. Accessed: March 19, 2015.

[13] T. Zgraggen. E-voting and secure human-server communication. Master's thesis, ETH Zurich, September 2014.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Secure Communication Topologies in the Internet of Things

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Schweizer	Danny

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 19.03.2015	D. Schweizer

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.