Universally Composable Firewall Architectures using Trusted Hardware[*]

Dirk Achenbach, Jörn Müller-Quade, and Jochen Rill

Karlsruhe Institute of Technology (KIT)
{dirk.achenbach,joern.mueller-quade,jochen.rill}@kit.edu

Abstract. Network firewalls are a standard security measure in computer networks that connect to the Internet. Often, ready-to-use firewall appliances are trusted to protect the network from malicious Internet traffic. However, because of their black-box nature, no one can be sure of their exact functionality.
We address the possibility of actively compromised firewalls. That is, we consider the possibility that a network firewall might collaborate with an outside adversary to attack the network. To alleviate this threat, we suggest composing multiple firewalls from different suppliers to obtain a secure firewall architecture. We rigorously treat the composition of potentially malicious network firewalls in a formal model based on the Universal Composability framework. Our security assumption is trusted hardware. We show that a serial concatenation of firewalls is insecure even when trusted hardware ensures that no new packages are generated by the compromised firewall. Further, we show that the parallel composition of two firewalls is only secure when the order of packets is not considered. We prove that the parallel composition of three firewalls is insecure, unless a modified trusted hardware is used.

Keywords. Formal Models, Firewalls, Universal Composability, Network Security.

1 Introduction

The protection of computer networks against attackers from the Internet is crucial for companies to protect their intellectual property. Network firewalls are used to shield networks against threats from the Internet. The task of a firewall is to inspect the network packets that pass through it and then to decide whether to let them pass. Firewalls use a set of predefined rules to facilitate this decision. These rules may specify static filters, but may also be functions of the history of network traffic.

The protection from attacks from outside the network is a well understood problem and has become the business model of many companies. Firewalls are considered a secure *black box* which protects the network from attacks. However, since many firewall appliances are purchased from third-party vendors, people have no control and insight into their actual functionality. Companies and individuals alike rely on the trustworthiness of firewall appliances. Most firewalls are made of general-purpose hardware with just the same capabilities as any modern computer. In addition, firewalls are often equipped with update mechanisms which make their functionality completely replaceable. It seems naïve to see a firewall as a secure black box. Indeed, as the documents that were leaked by Edward Snowden in 2013 reveal, the National Security Agency has the capability to install backdoors in a number of commercial firewalls: JETPLOW, HALLUXWATER, FEEDTROUGH, GOURMETTROUGH, and SOUFFLETROUGH [1].

The serial concatenation of firewalls does not yield a secure solution if one considers the possibility of actively malicious firewalls. In this paper, we address this problem. We give solutions using trusted hardware. We model our solutions in the Universal Composability framework.

[*]This work was presented at BalkanCryptSec 2014.

1.1 Related Work

To our knowledge, we are first to explicitly model network firewalls in the UC framework.

Network Firewalls The purpose, aim and function of network firewalls is widely understood and agreed upon, e.g. [3, 19, 11]. The informational RFC 2979 [6] defines characteristics of firewalls. Since there is no globally agreed-on standard for what constitutes good and bad network packets however, there is also no complete specification of the function of a firewall.

The security of firewalls or systems of firewalls has mainly been studied under two aspects. One concern is verfiying the correctness and soundness of rule sets. Gouda et al. [8] develop a formal model for verification of distributed rule sets based on trees. They are able to check whether the firewall system accepts or denies a specific class of packets. Ingols et al. [12] check for conflicting or otherwise problematic rules with the aid of Binary Decision Diagrams.

We are not aware of any works that consider the firewall as being malicious.

Universal Composability The Universal Composability (UC) framework [4] was proposed by Canetti in 2001. It follows the "real-world-ideal-world" paradigm and is inherently asynchronous. Katz et. al. [14] recently proposed an extension for synchronous computation. Alternative approaches to the problem of composable security include the reactive simulatability framework [18, 2], the GNUC framework [10]—which aims at being a drop-in replacement for UC—or the Abstract Cryptography paradigm [17, 16]. The UC framework has been used to prove the security of a variety of protocols. An example of such a protocol is the OAuth protocol. Chari et al. [5] present a proof that the OAuth protocol is UC secure, given an ideal SSL functionality.

Secure Hardware Katz [13] uses tamper-proof hardware to realise universally composable multi-party computation. He assumes tamper-proof tokens that can be programmed with an arbitrary program. Such a programmed token is then handed to another party in the protocol, which may then interact with the token. Goldwasser et al. [7] introduce the computational paradigm of one-time programs, i.e. programs that can only be run once, on one input. Of course, such programs cannot exist purely in software, as software can be copied indefinitely. Goldwasser et al. introduce "one-time-memory devices" to create a compiler for one-time programs.

Robust Combiners The idea of mistrusting the implementation of a secure functionality has been studied in the scope of *robust combiners*. A (k, n)-robust combiner combines n candidate implementations of the secure functionality $\mathcal{P}$ in a way that the overall security still holds if at least k implementations are secure [9].

The notion of a robust combiner is not suited for our purposes. The very definition of robust combiners requires a specific and fixed functionality $\mathcal{P}$. However, in the case of firewalls, it is unclear what this functionality precisely is. Informally speaking, the functionality of a network firewall is "filtering all malicious packets". It is not possible to formalise this functionality in a protocol or a program, since, in general, it is not possible to decide whether an arbitrary packet is malicious or not.

Byzantine Fault Tolerance Our constructions are reminiscent of problems in byzantine fault tolerance. However, we use a very different communication structure. In the original Byzantine Generals Problem [15], every party can communicate with every other party. This leads to specific bounds concerning the number of trusted parties needed to achieve fault tolerance. Even when signing messages is possible, in order to allow for m corrupted parties, one still needs at least $(2m + 1)$ trusted parties and $(m + 1)$ rounds of communication. In our case, we do not allow the parties to communicate freely, but only according to the specific structure of the network—we do not allow firewalls to exchange messages with each other. Thus, the results which byzantine fault tolerance research provides are not applicable to our scenario.

1.2 Our Contribution

In this paper we explore the idea of actively malicious firewalls. We present a novel methodology to analyse architectures of multiple firewalls based on the Universal Composability (UC) framework. The UC framework allows us to define security in a natural way while also providing us with a composition theorem. We assume trusted hardware to compose firewalls: modules with simple and fixed functionalities that can compare network packets. We rigorously analyse different approaches. Our analysis reveals subtle attacks on presumably secure architectures. We give ideal functionalities for these architectures which make these weaknesses explicit.

1.3 A Model for Firewalls

We assume a packet-switched local-area network (LAN) in which there are only uncompromised hosts. They are connected through a single uplink to the Internet, in which are potentially compromised hosts. To facilitate an easier discussion, we call machines in the LAN being "inside" and machines on the Internet being "outside". The "inside" is only connected to the "outside" through a firewall (network), whose job is to protect machines "inside" from machines "outside". For ease of exposition, we model communication in networks in one direction only (cf. Section 2).

We assume that each firewall can have any number of *network interfaces* for input and output. The output of a firewall then depends on the packet $p \in P$ it gets as input (where P is the set of all possible packets), its internal state $s \in S$ and the network interface $i \in I$ the packet arrived on.

After processing this information, the firewall then outputs a packet p' on a specific network interface i' and updates its internal state (e.g. outputs a new internal state s'). The functionality of a firewall is defined formally in Definition 1.

Definition 1 (The functionality of an ideal firewall F_j).

$$F_{\mathrm{fw}_j} : P \times I \times S \to (P \cup \bot) \times (I \cup \bot) \times S$$

$$F_{\mathrm{fw}_j}(p, i, s) = \begin{cases} (p', i', s') & \textit{if output is generated,} \\ (\bot, \bot, s') & \textit{else.} \end{cases}$$

We stress that our definition of a firewall functionality is universal. Because it is stateful—it receives its previous state as input, may use it for its computation and outputs an updated state—a firewall may base its output on an arbitrarily long history of incoming and outgoing packets. It may, for example, reconstruct a TCP session. Further, we do not restrict how its output depends on its input. A firewall might for example receive a packet, store it, transform it, and output it much later. Because the functionality processes whole packets including their payload, our definition covers the whole network protocol stack (e.g. Ethernet, IP, TCP, HTTP, HTML).

1.4 The Universal Composability Framework

In this chapter we give a brief review of the Universal Composability (UC) framework. It is a tool for proving the security of multi-party protocols by comparing their execution with an idealised version of the protocol. The framework allows us to model a system of firewalls as a protocol execution and underspecify the concrete functionality of the participating firewalls and only state what an ideal execution would look like.

In the UC framework, participants in a protocol are modeled as Interactive Turing Machines (ITMs). Since there are different definitions of ITMs in literature, we will briefly summarise the definition given by Canetti [4].

Definition 2 (Interactive Turing Machine). *An* Interactive Turing Machine *(ITM) is a multi-tape turing machine with the following tapes. A tape is* externally writeable *(EW), if it can be written by every other turing machine.*

- *an* identity tape *(EW)*
- *a* security parameter tape *(EW)*
- *an* input tape *(EW)*
- *a* communication tape *(EW)*
- *an* output tape
- *a* working tape
- *a* subprocess tape

We call an ITM probabilistic*, if it, in addition, has a* random tape*, which contains a random bitstring of a specific distribution.*

A protocol is a number of interacting ITMs. The execution of a protocol π in the UC framework happens in the context of two additional ITMs: the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. The adversary represents the party which wants to attack the protocol, the environment represents the perception of the execution from an outside point of view.

There are some general restrictions concerning the communication among the participating parties: the adversary and the environment are allowed to communicate freely. In addition, the adversary is allowed to write to the communication tapes of every participant and the environment can write to the input tapes and receive outputs of the parties of the protocol. This captures the notion that the environment represents the external input to the protocol but will not interfere with the protocol itself.

We realise protocol execution in the $\mathcal{F}$-hybrid model of computation: Parties cannot communicate directly, but must use a functionality $\mathcal{F}$ as a proxy, which is modeled as another ITM. $\mathcal{F}$ also communicates with the adversary. The exact behaviour of $\mathcal{F}$ is specified in advance and must reflect the exact nature of the communication link. For example, $\mathcal{F}$ might be set up in a Dolev-Yao fashion, so that $\mathcal{A}$ can intercept and send arbitrary network messages. We will setup $\mathcal{F}$ in a way that reflects our network architecture: The adversary cannot intercept all communication or inject messages into the network at will. It is only possible to send messages on established links as specified in the architecture diagram. However, the adversary can send a special message to the other parties: the *corruption message*. If a party receives a corruption message, it stops executing its own program and instead gives complete control of its functions to the adversary. This includes disclosing its internal state.

The execution of the protocol is turn-based. If an ITM is activated, it can perform computations and write to a tape of any other ITM based on the aforementioned restrictions. Then its turn ends. If an ITM receives input on one of its tapes, it is the next to be activated. The first ITM to be activated is the environment $\mathcal{Z}$.

The output of the whole protocol is the output of $\mathcal{Z}$ and we assume, without loss of generality, that it consists of one bit. The distribution of all outputs of $\mathcal{Z}$ is a random ensemble based on the two parameters z (the input) and k (the security parameter).

Definition 3 (Ensemble of a protocol execution). *We denote the random variable which describes the execution of a protocol π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, input z, security parameter k as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$. The set of random distributions $\{\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in N, z\in\{0,1\}^*}$ is denoted as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

The security of a protocol execution in the UC framework is based on a comparison with an execution of an idealised version of the protocol: the *ideal protocol.* The ideal protocol contains the *ideal functionality* $\mathcal{F}_{\mathsf{ideal}}$ which completely realises the properties of the analysed protocol. In the ideal protocol, all parties only act as dummies which directly give their input to the ideal functionality and receive back their output without performing any computation themselves. The ideal functionality may communicate with the adversary in order to model the influence $\mathcal{A}$ is allowed to have. We call this adversary the "adversary simulator" $\mathcal{S}$. Since the only instance which performs computations is $\mathcal{F}_{\mathsf{ideal}}$, which is ideal by definition, the whole protocol execution is ideal and thus secure. Note that this does not model an absolute security guarantee but a guarantee relative to the defined ideal functionality.

We model the ideal functionality as the same firewall network, but with the adversary removed. For example, when we combine two firewalls, of which one may be compromised, the ideal model we compare our protocol to is just one uncompromised firewall.

Definition 4 (Ideal protocol). *Let $\mathcal{F}_{\mathsf{ideal}}$ be an ideal functionality. Then, the ideal protocol which realises $\mathcal{F}_{\mathsf{ideal}}$ is denoted as $\mathsf{IDEAL}_{\mathcal{F}}$.*

Informally, a protocol π is UC secure if, for every adversary $\mathcal{A}$ and every environment $\mathcal{Z}$, $\mathcal{Z}$ can not distinguish if it is interacting with π or with the ideal protocol implementing π. To capture that notion formally, we define indistinguishability. Because parties may behave indeterministically, their outputs are modeled as distributions. Further, since protocol runs are parameterized (e.g. by the security parameter k), the following definition uses probability ensembles.

Definition 5 (Indistinguishablity). *Two binary ensembles X and Y are* indistinguishable *($X \approx Y$), if $\forall c, d \in N\ \exists k_0 \in N$, so that for all $k > k_0$ and all $a \in \cup_{\kappa \leq k^d} \{0,1\}^\kappa$ holds:*

$$|\Pr(X(k,a) = 1) - \Pr(Y(k,a) = 1)| < k^{-c}$$

Based on that notion, we now formalise the indistinguishability of two protocols in the UC framework. The simulator's job is to simulate the presence of $\mathcal{A}$ to the environment, so that it cannot distinguish the real protocol execution from the idealised version. The security notion requires that there is a successful simulator for every adversary.

Definition 6 (UC emulates). *Let π and ϕ be two protocols. Then π UC emulates the protocol ϕ, if $\forall \mathcal{A}\ \exists \mathcal{S}$, so that $\forall \mathcal{Z}$ holds:*

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$$

We can now formally state when a protocol *realises* a functionality.

Definition 7 (UC realises). *A protocol π (securely)* UC realises *an ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, if π UC emulates the corresponding ideal protocol $\mathsf{IDEAL}_{\mathcal{F}}$.*

If a protocol π realises a given ideal functionality, then we say π is *UC secure.*

The UC framework is a powerful instrument for analyzing the security of protocols because it provides a composition theorem. Informally speaking, the composition theorem states that if π securely realises an ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, one can use π instead of $\mathcal{F}_{\mathsf{ideal}}$ in other protocols without compromising security.

2 Composing Firewalls

In this section, we discuss different architectural solutions to the problem of maliciously acting firewalls and analyse their security in the Universal Composability (UC) framework. To simplify the exposition, we only discuss unidirectional networks. The easiest approach for extending the model to bidirectional communication would be using a independent instance of $\mathcal{F}_{\mathsf{ideal}}$ for each direction and deducing the security of the composed system by using the Composition Theorem. However, this approach would require the protocols for each direction to be independent of each other and not have a joint state. Actual firewall solutions base their decisions on all observed packets (not only those in one direction), however. Thus, the security of the bidirectional extensions of the architectures we discuss has to be proven manually.

We only discuss the security of a single atomic building block for complex firewall architectures. The Composition Theorem of the UC framework provides us with a strong guarantee for networks composed of several building blocks.

2.1 Adversarial Model

We assume an outside adversary who can statically corrupt exactly one firewall in the network. He gains full control over this firewall and can send and receive messages in its name. (Via a GSM link, for example.) Because our constructions are symmetric, our corruption model is equivalent to an adaptive model.

2.2 Trusted Hardware

The UC framework gives strong security guarantees. It is difficult to obtain secure protocols in the plain UC model, however. This problem can be alleviated by using a set-up assumption like a Common Reference String, a Public-Key Infrastructure or trusted hardware. Secure hardware is often modelled as tokens that offer some black box functionality. They cannot be made to deviate from the specified functionality and parties may not learn the value of internally stored values if they are not allowed to do so.

We envision similar network devices for our task. They have two very simple functionalities depending on the direction of the packet flow. In one direction their job is to compare packets that come in from different sources and decide whether to let them pass. In the other direction their job is to split incoming packets and distribute them to several firewalls. Because these "packet comparators" offer only limited functionality, they could be manufactured easily, maybe even by the network owner himself. Also, it would be very hard to hide any backdoors or undocumented functionality in the device. Thirdly, because of its simple functionality, the device need not be able to download updates or even be freely programmable. We envision such a device to be realised as an Application-Specific Integrated Circuit (ASIC). In our security analysis, we assume that the specialised hardware we use cannot be compromised, i.e. is *trusted hardware*.

We formalise the functionality for comparison in Figure 2 and the functionality for splitting in Figure 1. This explicit distinction is solely for the purpose of simplifying the model in the case of uni-directional communication. Practically realising the required functionality at wire speed is not impractical: The trusted device need only compare packets as fast as the slowest of the firewalls can analyse them.

We express the notion of "packet equivalence" with a relation $\equiv$ that we assume to be defined appropriately.

A firewall may change the order of the packets it delivers. Some packets might need to be inspected more closely (Deep Packet Inspection), while others would just be waved through—take for example packages from a voice-over-IP (VoIP) connection. Therefore, it is not sufficient for the trusted hardware to compare packets one-by-one in the order they arrive.

Formally analysing availability is outside the scope of this work.

An idealised description of trusted hardware split

Upon receiving packet p on the interface connected to the outside:

- Output p to every interface connected to a firewall.

Fig. 1. The splitting functionality the trusted hardware must perform. We assume that there is a fixed interface for connecting with the outside network and a number of other interfaces to which the firewalls are connected.

An idealised description of trusted hardware hw

Keep a local cache realised as an unordered list.
Upon receiving packet p on interface i:

- If there is another input interface $j \neq i$, and a corresponding entry (j, q) with $p \equiv q$ in the cache:
  - Remove (j, q) from the cache,
  - output p.
- Otherwise, store (i, p) in the cache.

Fig. 2. The function the trusted hardware must perform. Because network packets may arrive at different times or in a different order, they must be cached in order to be available for comparison. We assume the relation $\equiv$ to be defined appropriately.

2.3 Serial Concatenation of Two Firewalls

An obvious idea is to concatenate two firewalls and compare whether packets that exit the network originally were sent from the outside. This way, no firewall can "make up" packets. This concatenation of firewalls is not secure. We formalise this claim in Appendix A. For the sake of brevity, we only state our result here.

Figure 3 shows a graphical representation of the network architecture of the serial concatenation. fw_1, fw_2, split and hw will be the parties in the corresponding UC protocol.

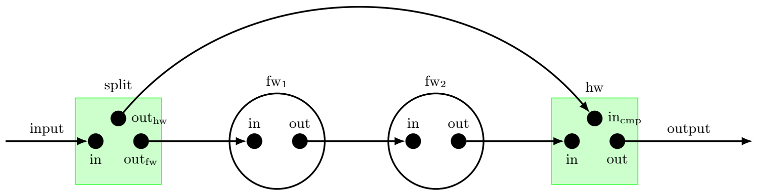

Fig. 3. The serial concatenation of firewalls using secure hardware to compare packets. hw compares whether "what goes in, comes out". split forwards the packet to two components. The connecting arrows represent network cables in a "real" network.

Packets from outside the network always arrive at split first. Parties cannot communicate directly. Instead, we provide them with an ideal functionality for communication. This functionality ensures that parties can only communicate in a way that is fixed by the structure of the network. This is justified, since in an "real" network, components can also only communicate along the network cables.

We omit session IDs from all descriptions of functionalities and protocols. Different instances behave independently. We use the notion of "public delayed output", introduced by Canetti [4].

This means that a message is given to the adversary prior to delivery. The adversary then decides when (or whether) it is delivered.

The ideal functionality of two firewalls $\mathcal{F}_{\mathsf{ideal}}$

- Upon receiving (input, p):
  - Let fw_k be the non-corrupted party; compute $F_{\mathrm{fw}_k}(p, \mathrm{in}, s) = (p', i', s')$. Ask the adversary if p should be delivered. If yes, $p' \neq \bot$ and $i' \neq \bot$, write p' to the output tape of hw. Else, do nothing. Save the new internal state s'.

Fig. 4. The ideal functionality of two firewalls.

The main idea for the ideal functionality is that any firewall architecture, regardless of the amount of different firewalls or their specific rule set, should behave as if the corrupted firewall was not there (see Figure 4).

We only state our result and the idea for the proof here. For details and a full proof, see Appendix A.

Theorem 1. π_{serial} *does* not *UC realise* $\mathcal{F}_{\mathsf{ideal}}$ *in the* $\mathcal{F}_{\mathsf{serial}}$*-hybrid model.*

The idea is that if fw_2 is corrupted, it could output a malicious packet just at the same time this packet arrives at split (sent by the environment). This would force hw to output the packet, even though it was blocked by fw_1.

2.4 Parallel Composition of Two Firewalls

The serial composition of two firewalls is not secure with regard to our envisioned ideal functionality. Better results can be achieved using parallel composition. The idea is that the trusted hardware only accepts a packet if both firewalls accept it. Figure 5 shows this composition. We will now discuss the security of this architecture.

The protocol of the parallel architecture is defined in Definition 8.

Definition 8 (Protocol of the parallel architecture π_{parallel}).

- split*: Upon receiving* (input, p)*: Call* $\mathcal{F}_{\mathsf{parallel}}(\mathsf{send}, \mathrm{out}_1, \mathrm{out}_2, p)$*.*
- fw_k*: Upon receiving* (in, p)*: Calculate* $F_{\mathrm{fw}_k}(p, \mathrm{in}, s) = (p', i', s')$*. If* $p' \neq \bot$ *and* $i' \neq \bot$*, call* $\mathcal{F}_{\mathsf{parallel}}(\mathsf{send}, p', i')$*. Save the new internal state* s'*.*
- hw*: Upon receiving* (in_i, p)*, check if there is an entry* (in_j, q) *with* $i \neq j$ *and* $p \equiv q$ *in the internal storage. If so, write* p *to the output tape and remove both entries. Else, do nothing.*

The functionality describing the network structure is depicted in Figure 6.

We will compare the protocol from Definition 8 with an ideal functionality. The ideal functionality is the same as in the serial case, since the natural approach of defining ideal functionalities only uses the uncorrupted firewall, which again leads to the functionality in Figure 4. However, as in the serial case, the parallel architecture does not realise this functionality.

Theorem 2. π_{parallel} *does not UC realise* $\mathcal{F}_{\mathsf{ideal}}$ *in the* $\mathcal{F}_{\mathsf{parallel}}$*-hybrid model.*

We prove this by describing an attack which cannot be simulated.

Proof. Let, w.l.o.g., fw_1 be honest and fw_2 be corrupted. Also, let p_1 and p_2 be packets that are accepted by fw_1. The environment sends packets p_1 and p_2 to the architecture which the adversary delivers to fw_1. Both packets are accepted by fw_1 and forwarded to hw. Then, the adversary sends packets p_2 and p_1 from fw_2. Since both packets have been accepted and were sent to hw previously (but in reverse order), hw will send out p_2 and p_1—in this order. Thus, the adversary was able to reverse the order of packets. Since the adversary is not allowed to influence the order of packets in the ideal model, there exists no simulator which can simulate this attack. ■

The Internet Protocol explicitly does not give any guarantees about the ordering of packets, since the correct order is encoded in the packet. The packet itself, however, can not be altered by the adversary. Thus, we modify our ideal functionality and explicitly grant the attacker the ability to reorder the outgoing packet stream. The new ideal functionality is described in Figure 7.

Theorem 3. π_{parallel} *UC realises* $\mathcal{F}_{\mathsf{ideal}_2}$ *in the* $\mathcal{F}_{\mathsf{parallel}}$*-hybrid model.*

Proof. To prove the statement, we will give the description of a simulator and show that this simulator can simulate every adversary, so that no environment can distinguish between the real and ideal model. Let w.l.o.g. fw_1 be corrupted and fw_2 be honest. Let $\mathcal{S}$ be a simulator with the following functionality:

– Upon activation, or when given a packet p, simulate the real model and observe its output. If the output of the real model is a packet p', calculate the position of p' in the internal memory structure of the ideal functionality and advise the functionality to deliver the packet on that index. (The case that p' is not found in the internal memory structure of the ideal functionality need not be covered, as is proven below.)

Note that the simulator receives exactly the same input as the adversary in the real model—it can perfectly simulate the communication between the adversary and the environment. Thus, the environment can only distinguish the models based on their output streams. We argue that the output of the real and ideal model are identical. Assume towards a contradiction that they are not.

Let $\{fw_2(S)\}$ denote the set of all packets fw_2 outputs when given the input stream S. There are two possibilities which would cause a difference in output streams:

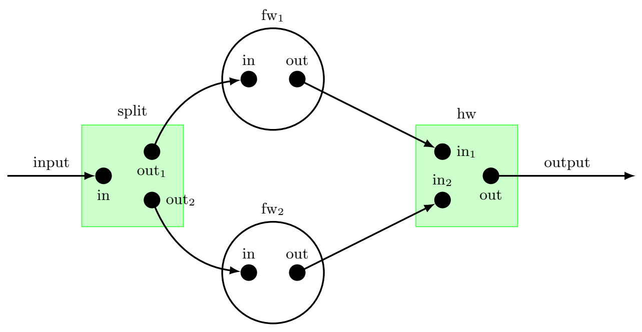

Fig. 5. The parallel composition of two firewalls with trusted hardware. *hw* only accepts packets that are output by both firewalls.

The ideal network function $\mathcal{F}_{\text{parallel}}$

Initialise an empty queue (first-in-first-out) for both fw_1 and fw_2.

- Upon receiving $(\mathsf{send}, \text{out}_1, \text{out}_2, p)$ from split: Push p into the queue for both fw_1 and fw_2.
- Upon receiving $(\mathsf{deliver}, \text{fw}_k)$ from the adversary: Fetch the next packet p from fw_k's queue. Provide public delayed output of (p, in) to fw_k.
- Upon receiving $(\mathsf{send}, \text{out}, p)$ from fw_k: Provide a public delayed output of (p, in_k) to hw.

Fig. 6. The ideal network function representing the parallel concatenation of firewalls with trusted hardware.

The ideal functionality of two firewalls with packet reordering $\mathcal{F}_{\text{ideal}_2}$

- Upon receiving (input, p): Let w.l.o.g fw_1 be the non-corrupted party; compute $F_{\text{fw}_1}(p, \text{in}, s) = (p', i', s')$. If $p' \neq \bot$ and $i' \neq \bot$, save p' in an indexed memory structure m at the next free index. Save new internal state s'. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j)$ from the adversary: If $m[j]$ contains a valid packet, write $(\text{out}, m[j])$ to the output tape of hw and clear $m[j]$; else do nothing.

Fig. 7. The ideal functionality of two firewalls with packet reordering.

Case 1. The adversary in the real model suppressed a packet which did not get suppressed in the ideal model. This is impossible however, since the simulator only advises the ideal functionality to deliver a packet if it observes it being output in its simulation of the real model.

Case 2. The real model outputs a packet which is not output in the ideal world.
Assume that this was the case and let p be that packet. The following conditions have to hold: p has to be in $\{fw_2(S)\}$ and p has to be output by $\mathcal{A}$ (using fw_1). This is true because the trusted hardware will ensure that a packet is only output when both firewalls accept it. For a packet *not* to be output in the *ideal model*, one of the following conditions have to hold:

- p is not in $\{fw_2(S)\}$. This way, p will not be in the internal memory structure of the ideal functionality. Thus, the simulator can not advise the delivery of that packet. This is a contradiction, since we assumed that p was output in the real model, which in turn implies that $p \in \{fw_2(S)\}$.
- $p \in \{fw_2(S)\}$ *and* the simulator did not advise the functionality to deliver p. This is also a contradiction, since we assumed that p was output in the real model. This would cause the simulator to advise the output of p by definition.

We now have shown that the assumption that the environment can observe a difference in packet output stream in the real and the ideal world leads to a contradiction in all cases. This, together with the ability of the simulator to fully simulate the adversary, proves the indistinguishability of the models. ■

2.5 Parallel Composition of Three or More Firewalls

The parallel approach to compose firewalls described above does indeed improve security compared to one single and potentially malicious firewall. There is large class of attacks that become possible when the adversary can selectively suppress packets. Because the parallel architecture with two firewalls has this weakness, we extend the architecture to a quorum of three firewalls.

In the following section, we assume that uncorrupted firewalls in this architecture will have the same behaviour. However, we allow them to disagree on the order of packets.

There is a non-trivial attack on this architecture. When both uncorrupted firewalls both output the same packet p, the adversary can use clever timing to output p from the corrupted firewall directly after the first uncorrupted firewall. The trusted hardware would then observe two p packets on different interfaces and output p. However, a third p packet would arrive from the second uncorrupted firewall. Then, the adversary could output p again. This would cause hw to output p again and thus duplicate the packet. The natural extension of $\mathcal{F}_{\mathsf{ideal}_2}$ to the case of three firewalls already covers this attack. This functionality is depicted in Figure 8.

The other protocols and functionalities ($\pi_{\mathsf{parallel}_3}$ and $\mathcal{F}_{\mathsf{parallel}_3}$) can easily be extended to the case of three firewalls by adding the third firewall as an additional party. We will omit their descriptions here.

The ideal functionality of three firewalls $\mathcal{F}_{\mathsf{ideal}_3}$

- Upon receiving (input, p): Let w.l.o.g. fw_1 and fw_2 be the non-corrupted parties; compute $F_{\mathrm{fw}_1}(p, \mathrm{in}, s) = (p', i', s')$ and $F_{\mathrm{fw}_2}(p, \mathrm{in}, s) = (p'', i'', s'')$. If $p' \neq \bot$ and $i' \neq \bot$, save p' in an indexed memory structure m at the next free index. If $p'' \neq \bot$ and $i'' \neq \bot$, also save p'' in m at the next free index. Save the new internal states. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j)$ from the adversary: If $m[j]$ contains a valid packet, write $(\mathrm{out}, m[j])$ to the output tape of hw and clear $m[j]$; else do nothing.

Fig. 8. The ideal functionality of three firewalls.

The attack described above can also be performed in $\mathcal{F}_{\mathsf{ideal}_3}$. When fw_1 and fw_2 both output the same packet, both will be saved in m. The adversary can now output both packets by giving the right indices.

Theorem 4. *$\pi_{\mathsf{parallel}_3}$ UC realises $\mathcal{F}_{\mathsf{ideal}_3}$ in the $\mathcal{F}_{\mathsf{parallel}_3}$-hybrid model.*

The proof is very similar to the proof of Theorem 3. We omit it here.

It is not acceptable to give an attacker the ability to duplicate packets. We alter the functionality of our trusted hardware slightly to prevent the attack. The functionality is depicted in Figure 9. The idea is that at the moment the hardware outputs a packet, exactly two firewalls must have output this packet before. Then, the hardware can mark this packet as missing from the third firewall. If it arrives eventually, this mark will be removed and no further action will be taken.

The corresponding ideal functionality is depicted in Figure 10. It now continously checks whether the amount of identical packets being given to *hw* matches the amount of identical packets which either one of the uncorrupted firewalls sent. As previously however, we allow the reordering of packets.

Theorem 5. *$\pi_{\mathsf{parallel}_4}$ UC realises $\mathcal{F}_{\mathsf{ideal}_4}$ in the $\mathcal{F}_{\mathsf{parallel}_3}$-hybrid model.*

Proof. The general technique is similar to the technique used in the proof of Theorem 3. See Appendix B for details.

3 Conclusion and Directions for Future Research

In this work, we consider the problem of actively malicious firewalls. We introduce a framework for analysing firewall networks consisting of multiple firewalls based on the Universal Composition

An idealised description of trusted hardware without packet duplication hw_3

Keep a local cache for each incoming interface realised as an unordered list. Upon receiving packet p on interface i:

- Check if the cache of interface i contains an entry $-q$ with $p \equiv q$. If so, delete $-q$ and halt.
- Check if there exists an interface $j \neq i$ with an entry q with $p \equiv q$ in the cache of that interface:
  - Remove q from the cache,
  - output p,
  - add an entry $-p$ to the cache of all other interfaces k with $k \neq i$ and $k \neq j$.
- Otherwise, store p in the cache of interface i.

Fig. 9. The updated functionality of the trusted hardware to prevent packet duplication. The hardware now marks a packet as "missing" if a firewall has not yet delivered it, but two others have.

The ideal functionality of three firewalls without packet duplication $\mathcal{F}_{\mathsf{ideal}_4}$

Initialise three index-based memory structures m_1, m_2 and m_{out}.

- Upon receiving (input, p): Let w.l.o.g. fw_1 and fw_2 be the non-corrupted parties; compute $F_{\text{fw}_1}(p, \text{in}, s) = (p', i', s')$ and $F_{\text{fw}_2}(p, \text{in}, s) = (p'', i'', s'')$. Save the new internal states. Save p' in m_1 and p'' in m_2. Give p to the adversary.
- Upon receiving $(\mathsf{deliver}, j, k)$ $(k \in \{1, 2\})$: If $m_k[j]$ contains a valid packet p''':
  - Check how many times that packet (or an equivalent packet) is in m_{out}. Let that number be n.
  - Check if either m_1 or m_2 (or both) contain that packet at least $n + 1$ times.
  - If so, write (out, p''') to the output tape and to m_{out}.

Fig. 10. The ideal functionality of three firewalls without packet duplication. For every packet, at least one of the firewalls must have sent this packet at least as often as it got passed to hw.

framework. The serial concatenation of firewalls turns out to be insecure. The parallel composition of two firewalls using a trusted packet comparator is secure with regard to an ideal functionality which allows for packet reordering. We show that this positive result can not be directly extended to a 2-out-of-3 quorum decision, since the adversary would be able to duplicate packets. We give a solution for that by describing a slightly different trusted packet comparator. We gave an ideal functionality for the three firewall architecture and prove security with respect to this functionalitiy.

An important open research question is how to rigorously analyse the availability of the approaches we discussed. Intuitively, the parallel approach with three or more firewalls is more available than the one with two firewalls.

Firewall combination strategies other than those we discussed need further exploration. In extension to a generalised k-out-of-n quorum approach, one might consider complex firewall networks in the fashion of multiple parallel and serial stages. Also, how to deduce security guarantees for bidirectional communication (via a "Bidirection Theorem" similar to the Composition Theorem in UC) is an important open problem.

References

1. Interactive graphic: The nsa's spy catalog. Spiegel Online International (December 2013), `http://www.spiegel.de/international/world/a-941262.html`
2. Backes, M., Pfitzmann, B., Waidner, M.: A general composition theorem for secure reactive systems. In: Naor, M. (ed.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 2951, pp. 336–354. Springer Berlin Heidelberg (2004), `http://dx.doi.org/10.1007/978-3-540-24638-1_19`
3. Bellovin, S., Cheswick, W.: Network firewalls. Communications Magazine, IEEE 32(9), 50–57 (1994)
4. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on (oct 2001)
5. Chari, S., Jutla, C.S., Roy, A.: Universally composable security analysis of oauth v2. 0. IACR Cryptology ePrint Archive 2011, 526 (2011)
6. Freed, N.: Behavior of and requirements for internet firewalls. RFC 2979 (2000)
7. Goldwasser, S., Kalai, Y., Rothblum, G.: One-time programs. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008, Lecture Notes in Computer Science, vol. 5157, pp. 39–56. Springer Berlin Heidelberg (2008), `http://dx.doi.org/10.1007/978-3-540-85174-5_3`
8. Gouda, M.G., Liu, A.X., Jafry, M.: Verification of distributed firewalls. In: Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE. pp. 1–5. IEEE (2008)
9. Herzberg, A.: Folklore, practice and theory of robust combiners. Journal of Computer Security 17(2), 159–189 (2009)
10. Hofheinz, D., Shoup, V.: Gnuc: A new universal composability framework. Cryptology ePrint Archive, Report 2011/303 (2011), `http://eprint.iacr.org/`
11. Ingham, K., Forrest, S.: A history and survey of network firewalls. University of New Mexico, Tech. Rep (2002)
12. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Computer Security Applications Conference, 2009. ACSAC'09. Annual. pp. 117–126. IEEE (2009)
13. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) Advances in Cryptology – EUROCRYPT 2007, Lecture Notes in Computer Science, vol. 4515, pp. 115–128. Springer Berlin Heidelberg (2007), `http://dx.doi.org/10.1007/978-3-540-72540-4_7`
14. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 7785, pp. 477–498. Springer Berlin Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-36594-2_27`
15. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (Jul 1982), `http://doi.acm.org/10.1145/357172.357176`
16. Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Moedersheim, S., Palamidessi, C. (eds.) Theory of Security and Applications (TOSCA 2011). Lecture Notes in Computer Science, vol. 6993, pp. 33–56. Springer-Verlag (Apr 2011)
17. Maurer, U., Renner, R.: Abstract cryptography. In: Chazelle, B. (ed.) The Second Symposium in Innovations in Computer Science, ICS 2011. pp. 1–21. Tsinghua University Press (Jan 2011)
18. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. pp. 184–200. IEEE (2001)
19. Schuba, C.L., Spafford, E.H.: A reference model for firewall technology. In: Computer Security Applications Conference, 1997. Proceedings., 13th Annual. pp. 133–145. IEEE (1997)

A Serial Composition of Two Firewalls

We prove the previously stated Theorem 1 here. First, we provide the protocol of the serial architecture and define the ideal network function.

Definition 9 (The protocol of the serial firewall architecture π_{serial}). *The protocol the parties are following is defined as follows:*

– split*: Upon receiving* (input, p)*: Call* $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, \mathrm{out}_{\mathrm{fw}}, \mathrm{out}_{\mathrm{hw}}, p)$*.*
– fw_k*: Upon receiving* (in, p)*: Calculate* $F_{fw_k}(p, \mathrm{in}, s) = (p', i', s')$*. If* $p' \neq \bot$ *and* $i' \neq \bot$*, call* $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, i', p')$*. Save the new internal state* s'*.*
– hw*: Check whether there are two entries* (p, in) *and* $(q, \mathrm{in}_{\mathrm{cmp}})$ *in the local storage (with* $p \equiv q$*). If so, write* p *to the output tape and delete the entries.*

The ideal network function $\mathcal{F}_{\mathsf{serial}}$

Initialize an empty queue (first-in-first-out) for both hw and fw_1.

– Upon receiving $(\mathsf{send}, \mathrm{out}_{\mathrm{fw}}, \mathrm{out}_{\mathrm{hw}}, p)$ from split:
  - Push p into the queue for hw and fw_1.
  - Give the adversary p and let him choose one of the queues. Pull the next packet from that queue. If the adversary chose hw's queue, send $(p, \mathrm{in}_{\mathrm{cmp}})$ to hw, otherwise send (p, in) to fw_1.
– Upon receiving $(\mathsf{send}, \mathrm{out}, p)$ from fw_1: Provide a public delayed output of (p, in) to fw_2.
– Upon receiving $(\mathsf{send}, \mathrm{out}, p)$ from fw_2: Provide a public delayed output of (p, in) to hw.

Fig. 11. The ideal network function representing the serial concatenation of firewalls with special hardware.

We now show that the serial concatenation of firewalls is not secure, even with a trusted comparator. To prove the statement, it suffices to show that there exists an attack which can not be simulated. We describe such an attack. The general idea is that if fw_2 is corrupted, it could output a malicious packet just at the same time this packet arrives at split (sent by the environment). This would force hw to output the packet, even though it was blocked by fw_1.

Theorem 1. π_{serial} *does* not *UC realise* $\mathcal{F}_{\mathsf{ideal}}$ *in the* $\mathcal{F}_{\mathsf{serial}}$*-hybrid model.*

Proof. Let fw_2 be corrupted and fw_1 be honest. Let p be a packet that is blocked by fw_1. The environment inputs p to split. This will cause $(p, \mathrm{in}_{\mathrm{cmp}})$ to be send to hw from split. In its next activation the adversary uses fw_2 to call $\mathcal{F}_{\mathsf{serial}}(\mathsf{send}, \mathrm{out}, p)$ and advises the ideal functionality to deliver (p, in) to hw. hw will now have two identical packets on different interfaces (one from split and one from fw_2) in its storage and output p, even though p has been blocked by fw_1.

There is no simulator which can simulate this attack, since fw_1 will block the packet in the ideal model and the output of fw_2 will not be considered. ■

B Parallel Composition of Three Firewalls

Theorem 5. $\pi_{\mathsf{parallel}_4}$ *UC realises* $\mathcal{F}_{\mathsf{ideal}_4}$ *in the* $\mathcal{F}_{\mathsf{parallel}_3}$*-hybrid model.*

Proof. The proof is similar to the proof of Theorem 3. We argue that the simulator behaves identically to the adversary and that the output of the ideal network is identical to the output of the real network. Let $\mathcal{S}$ be a simulator with the following functionality:

– Upon activation, or when given a packet p, simulate the real model and observe its output. If the output of the real model is a packet p', calculate (for the ideal functionality) the index of the memory structure in which p' is saved as well as its position within the memory. Advise the functionality to deliver the packet on that index. (The case that p' is not found in the internal memory structure of the ideal functionality need not be covered, as is proven below.)

The argument that $\mathcal{S}$ will never mistakenly suppress a packet in the ideal model is identical to Case 1 in the proof of Theorem 3. We need to argue Case 2: It is impossible that $\mathcal{S}$ is unable to schedule a packet it observes in the output of its internal simulation of the real network. Let p be such a packet that, after the input stream S is processed, is written to the output tape of hw in the real model but not to the internal memory structure of $\mathcal{F}_{\mathsf{ideal}_4}$.

Let m_A, m_1 and m_2 be the lists the trusted hardware uses in the protocol for storing the packets output by the firewalls and marking the "negative" packets. Let m_{hw} be the list of all packets it has ever output. Let m'_1, m'_2, m'_{out} be the lists the ideal functionality uses for keeping track of the packets. Let $\|m\|_p$ denote the number of packets p the list m contains. We then define $|m|_p := \|m\|_p - \|m\|_{-p}$.

First, observe that $\mathcal{S}$ only schedules packets it observes in its simulation of the real model. Hence, by the description of hw: $|m_1|_p = |m'_1|_p - |m_{\text{hw}}|_p$ and $|m_2|_p = |m'_2|_p - |m_{\text{hw}}|_p$. Via the argument from Case 1 ($\forall p : |m'_{out}|_p \leq |m_{\text{hw}}|_p$) we have:

$$|m_1|_p \leq |m'_1|_p - |m'_{out}|_p \tag{1}$$
$$|m_2|_p \leq |m'_2|_p - |m'_{out}|_p \tag{2}$$

For p to be output in the real model, one of the following conditions has to hold:

$$|m_A|_p > 0 \text{ and } |m_1|_p > 0 \tag{3}$$
$$|m_A|_p > 0 \text{ and } |m_2|_p > 0 \tag{4}$$
$$|m_1|_p > 0 \text{ and } |m_2|_p > 0 \tag{5}$$

This is true because the trusted hardware will only forward packets which are in at least two of the packet lists. The functionality of *hw* can be restated in the following way: For every packet p which is output, insert a packet $-p$ into the lists of the three firewalls. If there are two packets p and $-p$ in the same list, both cancel each other out.

For p *not* to be written to the internal memory structure of $\mathcal{F}_{\mathsf{ideal}_4}$ in the ideal model, the following condition has to hold:

$$|m'_{out}|_p \geq |m'_1|_p \text{ and } |m'_{out}|_p \geq |m'_2|_p \tag{6}$$
$$\Leftrightarrow |m'_1|_p - |m'_{out}|_p \leq 0 \text{ and } |m'_2|_p - |m'_{out}|_p \leq 0 \tag{7}$$

This again describes the difference between the amount of packages p each individual firewall has output and the amount of packages p which got output in total after processing S.

Concluding the argument, conditions (1) to (5) give us $|m'_1|_p - |m'_{out}|_p > 0$ and $|m'_2|_p - |m'_{out}|_p > 0$, which contradict condition (7). ∎