Fortified Universal Composability: Taking Advantage of Simple Secure Hardware Modules

Brandon Broadnax[1], Alexander Koch[1], Jeremias Mechler[1], Tobias Müller[2], Jörn Müller-Quade[1], Matthias Nagel[1]

[1] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[2] FZI Research Center for Information Technology

Abstract. Adaptive security is the established way to capture adversaries breaking into computers during secure computations. However, adaptive security does not prevent *remote hacks* where adversaries learn and modify a party's secret inputs and outputs. We initiate the study of security notions which go beyond adaptive security. To achieve such a strong security notion, we utilize realistic simple remotely unhackable hardware modules such as air-gap switches and data diodes together with isolation assumptions. Such hardware modules have, to the best of our knowledge, not been used for secure multi-party computation so far. As a result, we are able to construct protocols with very strong composable security guarantees against remote hacks, which are not provided by mere adaptive security. We call our new notion *Fortified UC* security.

Using only very few and very simple remotely unhackable hardware modules, we construct protocols where mounting remote attacks does not enable an adversary to learn or modify a party's inputs and outputs unless he hacks a party via the input port *before* it has received its (first) input (or gains control over *all* parties). Hence, our protocols protect inputs and outputs against all remote attacks, except for hacks via the input port while a party is waiting for input. To achieve this level of security, the parties' inputs and outputs are authenticated, masked and shared in our protocols in such a way that an adversary is unable to learn or modify them when gaining control over a party via a remote hack.

It is important to note that the remotely unhackable hardware modules applied in this work are based on substantially weaker assumptions than the hardware tokens proposed by Katz at EUROCRYPT '07. In particular, they are not assumed to be physically tamper-proof, can thus not be passed to other (possibly malicious) parties, and are therefore not sufficient to circumvent the impossibility results in the Universal Composability (UC) framework. Our protocols therefore rely on well-established UC-complete setup assumptions in tandem with our remotely unhackable hardware modules to achieve composability.

Keywords: universal composability, remotely unhackable hardware modules

1 Introduction

In the field of multi-party computation, one distinguishes between *static* and *adaptive* corruptions. In the static setting, parties may only be corrupted prior to the start of the protocol. In the adaptive corruption model, first proposed by [5], the adversary is able to corrupt parties throughout the entire protocol execution. As a consequence, an adversary mounting *remote hacks* (e.g. sending computer viruses) may be able to learn and modify a party's secret inputs and outputs even if the protocol is adaptively secure.

To prevent this, we introduce new protocols based on (few and simple) remotely unhackable hardware modules and isolation assumptions which are motivated from practice: A protocol party could be (temporarily) isolated from the network and may therefore not be hacked remotely (during this time). For instance, a party may use *data diodes* (unidirectional channels) or disconnect itself via *air-gap switches*. Furthermore, a party may have additional hardware modules at its disposal such as a simple encryption unit that only implements a specific public key encryption scheme. Such hardware modules with very limited functionality can be implemented securely as fixed-function circuits and formally verified for correctness. They can therefore be assumed to be resilient against remote hacking. In particular, an adversary can only corrupt such modules if he has direct physical access.

In order to adequately capture the advantages provided by remotely unhackable hardware modules, we propose a new framework—called *Fortified UC*—based on the UC framework [4]. In our new framework one distinguishes between *physical attacks* and *online attacks*. Physical attacks model adversaries physically tampering or replacing hardware. Online attacks model adversaries mounting remote attacks, e.g. by exploiting software bugs. Contrary to physical attacks, online attacks give the adversary control over a party only if the party is currently *online* and not assumed to be an unhackable hardware module. A party's current *online state* is determined by the type and state of its *channels*, e.g., state of its air-gap switches. The hardware modules used in a protocol and their interconnections are part of what we call the *protocol architecture*.

Utilizing only very few simple unhackable hardware modules, we construct protocols that protect against all online attacks i) mounted *after* a party received its (first) input and ii) mounted *before* a party received input if the attack comes from the "outside", i.e. from all channels except one at a party's input port. More specifically, the parties in our protocols are disconnected from the outside while waiting for input and can therefore not be corrupted via online attacks from the outside at that point. After receiving input, the parties authenticate, mask and share their secrets in such a way that mounting online attacks gives the adversary control over a party but not the ability to *learn the inputs or outputs of a party, nor to modify them* unless he gains control over *all* parties. This stands in contrast to adaptive UC security where an adversary may learn and modify the inputs and outputs of corrupted parties after they received input. Although erasing parties seem necessary for such a strong protection, we show that this assumption can be dropped using an appropriate protocol architecture.

We stress that our remotely unhackable hardware modules should not be confused with the tamper-proof hardware tokens proposed by [16]. Our hardware modules are based on substantially weaker assumptions. In particular, they can be tampered with if one has direct physical access to them. They cannot be passed to other (possibly malicious) parties but are only used and trusted by their owner. Our modules are thus not sufficient (and not intended) to circumvent the impossibility results of [6, 7]. Our protocols therefore rely on well-established UC-complete setup assumptions in conjunction with our remotely unhackable hardware modules to achieve composability.

Given these assumptions, our protocols provide the best possible protection against online attacks in a setting where parties cannot be protected while waiting for input.

1.1 Our Contribution

We utilize realistic simple remotely unhackable hardware modules that, to the best of our knowledge, have not been used for secure multi-party computation so far. Our main contributions are:

New Composable Security Framework: We propose a new security framework that, unlike previous frameworks, adequately captures the advantages provided by remotely unhackable hardware modules. As with UC security, our security notion is universally composable (Theorem 2). Furthermore, our security notion is equivalent to adaptive UC security for protocols that do not use any remotely unhackable hardware modules (Theorem 1).

New Protocols With Strong Security Guarantees Against Online Attacks: Using only very few simple remotely unhackable hardware modules, we construct MPC protocols with very strong security guarantees against online attacks: An adversary is unable to learn or modify a party's inputs and outputs by mounting online attacks unless he gains control over a party via the input port *before* the party has received its (first) input (or gains control over *all* parties). We present a construction for non-reactive functionalities (Theorem 3) using only two simple remotely unhackable hardware modules (apart from air-gap switches and data diodes) per party and a protocol for reactive functionalities (Theorem 5) that uses only one additional simple remotely unhackable hardware module. Both constructions can be proven secure in our new framework for adversaries that gain control over all but one parties. We also present an augmentation of these constructions that allow simulation also in the case that all parties are under adversarial control (Theorems 4 and 6). For simplicity, we assume erasing parties. However, we later show how this assumption can be dropped (cf. Section 6).

1.2 Related Work

Adaptive Security, first proposed in [5], captures security against adversaries that can corrupt parties at any point in the protocol. This notion has since received considerable attention in the literature, see e.g. [8, 15, 13, 9]. In contrast to adaptive security where an adversary may learn all secrets of a corrupted party, we achieve that remotely hacking a party after it received its inputs does not leak anything about them at all, except for the case that all parties have been corrupted.

Mobile adversaries [18, 2], a notion strictly stronger than adaptive security, models an adversary taking over a participant – similar in spirit to our framework as "remote hacks/virus attacks" – and possibly undoing the corruption at a later point in time.

Concerning the used *trusted building blocks*, we assume data diodes, which are channels which allow for communication only in one specified direction. [11] analyze the cryptographic power of unidirectional channels as a building block, whereas we use unidirectional channels as a shield against dangerous incoming data packets. [1] make use of other trusted building blocks, such as a secure equality check hardware module, to ensure the correct, UC-secure functioning of a parallel firewall setup in the case of a malicious firewall.

Tamper-proof hardware tokens, first proposed by [16], are an interesting research direction for finding plausible and minimal UC setup assumptions. Along this line of research, [12] showed strong feasibility results of what can be done with these tokens. Moreover, [10] showed that UC security is possible with a constant number of untrusted and resettable hardware tokens. Furthermore, [14] constructed constant-round adaptively secure protocols which allow all parties to be corrupted.

Isolation is a general principle in IT security, with lots of research on isolation through virtualization, see e.g. [17]. Isolation in this way can be seen as a software analog of a trusted, remotely unhackable encryption module. Moreover, there is a wealth of literature on data exfiltration/side channel attacks to air-gaps including attacks based on acoustic, electromagnetic and thermal covert channels, cf. [20], which are not relevant to our work, as they are for protecting against outgoing communication from malicious internal parties, while we use data diodes/air-gap switches for the purpose of not being *hackable* from the outside. As an example, the Qubes OS provides strict separation between application domains, allowing to use a isolated GPG environment in a safe manner [19].

2 The Fortified Universal Composability Framework

In this section, we present our changes to the UC framework.

2.1 Channels

In the UC framework, the model of computation consists of *instances* (ITIs) of interactive Turing machines (ITMs). Communication is modelled via `external write` instructions written on an ITI's outgoing message tape. The instruction takes the sender's code and ID, the receiver's code and ID, one of the receiver's externally writable tapes as well as the message as arguments. A *control function* decides if the instruction is allowed. There are three different ways for machines to communicate: provide input, send a message, give sub-routine output. This is modelled by `external write` instructions targeted at another party's input tape, incoming message tape or subroutine output tape, respectively (cf. [4]).

In order to model protection mechanisms such as *air-gap switches* and *data diodes* (unidirectional communication) as well as the *online state* of a protocol party, we explicitly specify (possibly multiple, uniquely identified) *channels* between ITMs that determine if communication between two ITMs is allowed and in which direction it is allowed. In particular, if there exists no channel between two ITMs then communication is not allowed between them.

Channels can be between (sub-)parties of a protocol or between a (sub-)party and an ideal functionality. In addition, channels can also be between a party and the environment or the adversary. Channels between a party and the environment model the allowed communication with calling parties (from other protocols). Channels between a party and the adversary model possible communication to the "outside world" that can be "delivered" by the adversary.

Channels are modelled on top of the existing communication mechanism of the UC framework. Specifically, each protocol description must include a set of channels involving the protocol parties, which is part of the *protocol architecture* (cf. Section 2.3). The protocol architecture is given to the control function as an additional input. An `external write` instruction is allowed by the control function only if there exists a channel that allows the intended communication between the sending ITI and the receiving ITI. Otherwise, an `external write` instruction is silently dropped.

In our framework, we have three kinds of channels: *standard channels* that permanently allow bi-directional communication as well as two kinds of *enhanced channels*: *air-gap switches* and *data diodes*.

Enhanced Channels. In our framework, we want to capture possible security gains resulting from being isolated by forbidding certain communication and hence corruption ("remote hacking") by the adversary. To this end, we introduce two kinds of enhanced channels:

1. *Data diodes* that allow communication in one direction only.
2. *Air-gap switches* that can be *connected* or *disconnected* by the party that operates them. Disconnected air-gap switches allow no data transmissions at all. Connected ones allow bi-directional communication. Each air-gap switch has an *initial* connection state determined by the protocol architecture.

In order to model the current state of air-gap switches, we introduce a special *air-gap switch status tape* for each party containing the identifiers of each of its air-gap switches as well as its current state. A party can change the current state of each of its air-gap switches by writing on this tape. The control function gets the contents of each air-gap switch status tape as an additional input.

Communication between $\mathcal{A}$ and $\mathcal{Z}$ and $\mathcal{A}$ and Ideal Functionalities. As in the UC framework, the adversary and the environment may freely interact with each other. The same applies to the communication between the adversary and ideal functionalities. Formally, we always assume standard channels between these ITMs which are given to the control function in addition to the protocol architecture. Communication between these ITMs is therefore independent of the given protocol architecture.

Terminology. Let μ and μ' be two ITMs. If there is a data diode between μ and μ' in the direction of μ' then we say that "μ is connected to μ' via data diode". If there is an air-gap switch operated by μ to μ' then we say that "μ is connected to μ' via air-gap switch". Likewise, we say that "μ is connected to μ' via a standard channel" if there is a standard channel between μ and μ'. If there is a channel C between μ and the adversary we say that "μ is connected to the adversary via C". Likewise, if there is a channel C between μ and the environment we say that "μ is connected to the environment via C". Furthermore, we say that "μ can send messages (or provide input or give output) to μ' via C" or that "μ' can receive messages (or input or output) from μ via C" if C is a channel between μ and μ' that allows the respective `external write` instruction.

Conventions for Graphical Depiction of Architectures. Main parties are represented by boxes with rounded corners, sub-parties and ideal functionalities by cornered ones. Boxes with bold lines and grey background denote that the sub-party is unhackable. Standard channels are denoted by lines, data diodes by —▷|— and air-gap switches by _/_ and _⧸_ (initially (dis)connected). Dashed lines denote standard channels to other parties that are not shown.

2.2 Online State

Online State of Channels to the Environment. The environment $\mathcal{Z}$ may, upon each activation, mark each channel that exists between $\mathcal{Z}$ and a protocol party either online or offline. For this, we introduce a special *channel marking tape* containing the identifiers of each channel to $\mathcal{Z}$ and the current markings. $\mathcal{Z}$ can change the current marking of each channel to $\mathcal{Z}$ by writing on this tape. The control function gets the contents of this tape as an additional input. As the environment embodies other, concurrently executed protocols this mechanism reflects the online state of the calling parties being implicitly incorporated in the environment. In addition, for each channel to $\mathcal{Z}$, $\mathcal{Z}$ is informed upon each activation if it can receive *output* from that channel. (Cf. the proof of the composition theorem (Theorem 2) in Section 3 where these two abilities of $\mathcal{Z}$ will be very important.)

Online State of Protocol Parties. A (sub-)party P of protocol π is *online via* C if C is a channel such that one of the following holds:

1. P can receive messages from the adversary via C.
2. P can receive output from an ideal functionality $\mathcal{F}$ via C.
3. P can receive output/input via C from a sub-party/calling party M and M is online via C' and C' is a channel between M and an ITM $\mu \neq P$.
4. P can receive input from the environment $\mathcal{Z}$ via C and $\mathcal{Z}$ has marked the channel C online.

If none of the above holds, P is *offline via C*. If there exists no channel such that P is online via that channel, P is *offline*. If P is not offline, P is *online*.

Intuitively, (1) models a party who is able to receive messages from the "outside world" and is therefore online. (2) models a party who is able to receive messages from a trusted third party $\mathcal{F}$ that "lives" somewhere in the outside world.[3] For instance, $\mathcal{F}$ could be a public bulletin board, a common reference string, or a trusted party evaluating a specific function. (3) models a party being *transitively* online via connections to other parties who are online. (4) models a party being transitively online via connections to a calling party from another protocol. Note that each party has an *initial online state* prior to invocation depending on the protocol architecture (in particular, the initial connection states of air-gap switches) and how $\mathcal{Z}$ initially marked the respective channels.

Status Report to the Adversary. Each time the adversary $\mathcal{A}$ is activated, he gets informed via which channels each party is online. This is called the *status*. As will be described in Section 2.3, $\mathcal{A}$ will be able to gain control over (hackable) parties when they are online.

Example 1. See Fig. 1a on Page 8 for a graphical depiction.

Consider an environment $\mathcal{Z}$ that permanently marks the channel to P_1 online and the channel to P_2 $\mathsf{offline}$. P_1 disconnects its air-gap switch to $\mathcal{Z}$ as soon as it has received input. Later, P_1 connects its air-gap switch to the adversary $\mathcal{A}$ at a specific point, say, after having erased its input.

Q_1 is always online (being connected to the ideal functionality $\mathcal{F}$ via a standard channel, cf. (2) on Page 6). The same holds for P_2 (being connected to the adversary $\mathcal{A}$ via standard channel, cf. (1) on Page 6). Therefore, Q_2 is also always online (being connected to P_2 via a standard channel, cf. (3) on Page 6). P_1 is online before receiving its input (being connected to the environment $\mathcal{Z}$ via a connected air-gap switch and $\mathcal{Z}$ has marked the channel to P_1 online, cf. (4) on Page 6), offline immediately afterwards, and online again after having erased its input (having connected its air-gap switch to the adversary $\mathcal{A}$, cf. (1) on Page 6). M's online state is the same as P_1's (being connected to P_1 via a standard channel, cf. (3) on Page 6).

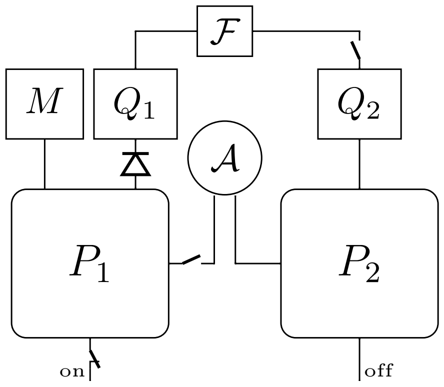
2.3 Corruption Model

We distinguish between two kinds of corruption: *physical attacks* and *online attacks*. Physical attacks model an adversary physically tampering with a party's hardware. Online attacks model remote hacks (e.g. sending computer viruses).

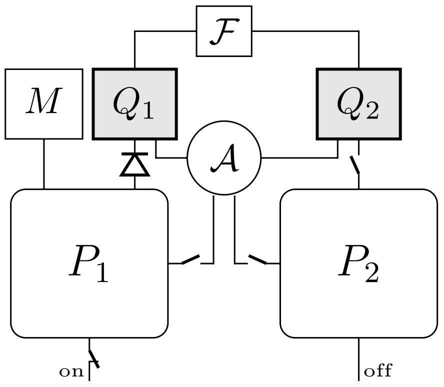
In a nutshell, the adversary is allowed to carry out physical attacks only *prior* to the start of the protocol execution.[4] Online attacks, on the other hand, can

[3] Note that it may be necessary to disable a party being online via a channel to specific functionalities such as signature cards in order to adequately model them. This can be done by, e.g., allowing functionalities to mark their channels to parties $\mathsf{offline}$ or online (like $\mathcal{Z}$). For simplicity, we do not consider this mechanism in this work.

[4] This captures adversaries secretly tampering with a party's hardware while that party is absent or models a malicious party setting up its own hardware. We note



(a) Architecture for Example 1



(b) Architecture for Example 2

Fig. 1: Architectures for Example 1 and 2.

be carried out throughout the protocol execution. Unlike physical attacks, online attacks only take effect if the targeted party is online and assumed to be *hackable*.

In the following, we describe our new corruption model in more detail.[5]

In our framework, parties can be either *hackable* or *unhackable*. The protocol architecture specifies which parties are hackable or unhackable (cf. Section 2.3).

Let $\mathcal{P}$ be the set of main parties of a protocol π. At the first activation[6], the adversary $\mathcal{A}$ may only send a `physical-attack` instruction that enables him to gain control over parties regardless of the protocol architecture. Formally, $\mathcal{A}$ writes $(\texttt{physical-attack}, \mathcal{M})$, where $\mathcal{M} \subseteq \mathcal{P}$, on his outgoing message tape. Each $P \in \mathcal{M}$ and all of their sub-parties are then connected to the adversary via a standard channel and all air-gap switches controlled by and data diodes coming from these parties are replaced with standard channels. From then on, $\mathcal{A}$ has full control over all $P \in \mathcal{M}$ and *all* of their sub-parties.[7]

From the second activation on, the adversary may not send a `physical-attack` instruction anymore. $\mathcal{A}$ may send `online-attack` instructions that enable $\mathcal{A}$ to gain control over *hackable* parties when they are online. Formally, if $\mathcal{A}$ writes $(\texttt{online-attack}, P)$ on his outgoing message tape and P is a (sub-)party of π that is online and hackable, then a standard channel between P and $\mathcal{A}$ is created and all air-gap switches controlled by P are connected. P then sends its entire

that our constructions for up to $N-1$ corrupted parties (cf. Theorems 3 and 5) can be shown to be secure even if physical attacks are allowed throughout the execution.

[5] Note that the following describes the behavior of protocol parties in the real model upon receiving corruption messages. As in the UC framework, in ideal protocols the behavior upon party corruption is determined by the ideal functionality.

[6] As in the UC framework, the first ITI to be invoked by the environment in our framework is $\mathcal{A}$ (cf. Definition 2 for the Fortified UC execution experiment).

[7] Formally, $\mathcal{A}$ can send these parties instructions by writing `external write` instructions. Note that since standard channels have been created between these parties and $\mathcal{A}$, these `external write` instructions are always allowed by the control function.

local state to $\mathcal{A}$. From then on, $\mathcal{A}$ has full control over P. If P is *unhackable*, then this instruction is ignored.

If $\mathcal{A}$ has gained control over a (sub-)party P through one of the above instructions, we say that P is "corrupted".

Finally, if a (sub-)party P is corrupted, then each ideal functionality which is connected to P is informed about P being corrupted through a special message $(\texttt{corrupt}, P)$ that is written on its incoming message tape. Also, each main party immediately informs the environment after being corrupted.

Example 2. See Fig. 1b on Page 8 for a graphical depiction.

Consider an environment $\mathcal{Z}$ that permanently marks its channel to P_1 online and to P_2 offline. On receiving input, P_1 disconnects its air-gap switch to $\mathcal{Z}$. P_2 connects its air-gap switches to the adversary and Q_2 upon receiving input.

At his first activation, the adversary $\mathcal{A}$ may write $(\texttt{pyhsical-attack}, \mathcal{M})$, $\mathcal{M} \subseteq \{P_1, P_2\}$. If, e.g., $\mathcal{M} = \{P_1\}$, then $\mathcal{A}$ gains control over P_1 and M as well as (the unhackable) party Q_1. Alternatively, from the second activation on, $\mathcal{A}$ may gain control over P_1 before P_1 has received its input by writing $(\texttt{online-attack}, P_1)$ (because P_1 is online via the channel to $\mathcal{Z}$ at this point). $\mathcal{A}$ may also choose to "skip" P_1 by writing $(\texttt{online-attack}, M)$ but not $(\texttt{online-attack}, P_1)$. This way, $\mathcal{A}$ can still gain control over P_1 after P_1 has received its input because P_1 is online via the channel to M (because a standard channel between M and $\mathcal{A}$ has been created). Moreover, $\mathcal{A}$ cannot gain control over P_2 through an $\texttt{online-attack}$ instruction before P_2 has received its input (because P_2 is offline up to this point).

"Tainting" Unhackable Parties. Consider an unhackable party E that is connected to a hackable party M via air-gap switch and to the adversary $\mathcal{A}$ via air-gap switch. E's air-gap switch to M is connected only if E's air-gap switch to $\mathcal{A}$ is disconnected. M is only connected to E. Therefore, $\mathcal{A}$ cannot gain control over M (through an $\texttt{online-attack}$ instruction) since M is offline. However, it should be intuitively possible for $\mathcal{A}$ to gain control over M since otherwise E would act as a "perfect firewall" for M.

In order to do so, $\mathcal{A}$ may send $\texttt{taint}$ instructions. Formally, if $\mathcal{A}$ writes $(\texttt{taint}, P)$ on his outgoing message tape and P is a (sub-)party of π that is online and unhackable, then a standard channel between P and $\mathcal{A}$ is created. This way, $\mathcal{A}$ can gain control over M since M is now online via the air-gap switch to E if that air-gap switch is connected.

Combination of Parties. In the UC framework, parties may be combined by giving them the same PID or the same value in a component of the PID (PID-wise corruption). Intuitively, combined parties are processes running on the same physical machine and therefore may only be corrupted together . In our framework, two parties $P = (\texttt{pid}_1 || \ldots || \texttt{pid}_l)$ and $P' = (\texttt{pid}'_1 || \ldots || \texttt{pid}'_m)$ are *combined* if i) $\texttt{pid}_1 = \texttt{pid}'_1$ and ii) P, P' are connected via standard channels only and iii) P, P' are both either hackable or unhackable. If P, P' are combined then any $(\texttt{online-attack}, \mu)$ or $(\texttt{taint}, \mu)$ instruction such that $\mu \in \{P, P'\}$

affects both parties. We will later (implicitly) combine dummy parties with their respective calling party in the constructions presented in this work.

Remark 1. Note that our corruption model gives the adversary lots of freedom. In particular, the adversary is still able to freely control a party corrupted via an `online-attack` instruction even in the case that such a party is offline via all channels specified by the protocol architecture (as is the case for party M after P_1 has received its input in Example 2 on Page 9).

This is because we grant the adversary standard channels to parties corrupted via `online-attack` instructions. Intuitively, this models the ability of a corrupted device to communicate with the outside world via side-channels. Allowing the adversary to corrupt a party (via `online-attack` instructions) if a party is only online via channels to a *tainted* party can also be seen as exploiting side channels. Also, the adversary always knows which parties are online and can gain control over a party even if that party is not connected to the adversary but, e.g., online via some channel to a sub-party that is online. Our corruption model therefore captures the vulnerabilities implied by being online in a very pessimistic way. This has the advantage of making the security notion of our framework both strong and simple at the same time.

Protocol Architecture. The *protocol architecture* of a protocol π is the set of all channels involving the parties of π and, in addition, a specification of the initial connection state of each air-gap switch that exists in that set and for each party in π also a specification of whether that party is hackable or unhackable.

2.4 Interface Modules and Fortified Functionalities

Recall our security goal: The adversary should be unable to learn or modify a party's inputs and outputs (i.e. results of the MPC) via `online-attack` instructions i) carried out *after* a party received its (first) input (unless *all* parties are corrupted) and ii) carried out before a party received input if the online attack "comes from the outside". To model this goal, we introduce *interface modules*, an appropriate *ideal-model protocol architecture* and *fortified functionalities.*

Interface Modules. In order to achieve the above-mentioned level of security, a party's result of the MPC must remain unmodified and hidden from the adversary $\mathcal{A}$ even if the party is corrupted via an `online-attack` instruction *after* receiving input. This is not possible if a party learns its result and outputs it itself since $\mathcal{A}$ would learn this result if he corrupts the party and could then also instruct the party to output a value that does not equal its result. Furthermore, for reactive tasks, a party corrupted after receiving input (via an `online-attack` instruction) must also not be able to learn or modify its input(s) for the rounds ≥ 2.

Deviating from the UC framework, we therefore allow the main parties to invoke special sub-parties called *interface modules* that are connected to their main party as well as to the environment via channels specified by the protocol architecture. These interface modules may thus give subroutine output to or receive input from the environment subject to the protocol architecture.

Intuitively, interface modules model simple hardware modules connected to, e.g., a PC. During the protocol execution, a user does not trust his PC since it may have been remotely hacked (in particular, the output of his PC may have been altered by a hacker). Instead, he only trusts the unhackable interface modules and, in particular, the outputs given by them (e.g. via a display).

In our constructions, interface modules will be unhackable sub-parties with very limited functionality. We will assume an interface module called *output interface module* (OIM) that is used for ensuring that a party's result of the MPC remains unmodified and hidden from the adversary even in the case that the party is corrupted after receiving input. More specifically, a party's result(s) will only be learned by its OIM, which outputs these result(s) instead of the party. For reactive tasks, we will also assume an *input interface module* (IIM) that is used for ensuring that a party's input(s) for the rounds ≥ 2 remain secret and unmodified. Note that in the ideal execution, the ideal functionality may also interact with dummy parties corresponding to interface modules (see Definition 1).

Ideal Protocols. In ideal protocols, each dummy party is connected to the environment and to the ideal functionality $\mathcal{F}$ via channels specified by the ideal protocol's architecture. Recall that, as described in Section 2.3, $\mathcal{F}$ is informed through a special message $(\texttt{corrupt}, P)$, which is written on its incoming message tape, when a party P connected to $\mathcal{F}$ is corrupted.[8]

Denote by $\texttt{SC}(\mathcal{F})$ the ideal protocol where the dummy parties are connected to $\mathcal{F}$ and the environment via standard channels. For a *non-reactive*[9] functionality $\mathcal{F}$, let $\texttt{AG}(\mathcal{F})$ be the ideal protocol where N *hackable* "dummy main parties" $P_1, \ldots, P_N$ are connected to $\mathcal{F}$ via an initially *disconnected* air-gap switch and to the environment via an initially *connected* air-gap switch and additionally N *unhackable* "dummy output interface modules" $\mathrm{OIM}_1, \ldots, \mathrm{OIM}_N$ are connected to $\mathcal{F}$ and the environment via standard channels. (For a graphical depiction of $\texttt{AG}(\mathcal{F})$, see Fig. 2 in Appendix A.) Upon input v, each party P_i disconnects its air-gap switch to the environment, connects its air-gap switch to $\mathcal{F}$, and passes v to $\mathcal{F}$. Each P_i connects its air-gap switch to the environment again upon receiving a special message $\texttt{open}$ from $\mathcal{F}$. Furthermore, if $\mathcal{F}$ is *reactive*, $\texttt{AG}(\mathcal{F})$ additionally contains N *unhackable* "dummy input interface modules" $\mathrm{IIM}_1, \ldots, \mathrm{IIM}_N$ which are connected to $\mathcal{F}$ and the environment via *intitially disconnected* air-gap switch. Each IIM_i connects its air-gap switch to the environment upon receiving $\texttt{open}$ from $\mathcal{F}$.

By construction, $\texttt{AG}(\mathcal{F})$ ensures that each party P_i cannot be corrupted by an $\texttt{online-attack}$ instruction "coming from the outside" prior to receiving input,

[8] Note that the adversary is not allowed to write $\texttt{external write}$ instructions containing the special message $(\texttt{corrupt}, P)$ in order to prevent him from bypassing the corruption rules (e.g. by sending $(\texttt{corrupt}, P)$ to $\mathcal{F}$ during the protocol execution while party P is offline).

[9] For a definition of reactive resp. non-reactive functionalities, see Appendix B.

i.e. each P_i can only be corrupted by an `online-attack` instruction prior to receiving input if it is online via its channel to the environment (which is the case if the environment marks this channel `online`). Note that in the following we will also refer to OIM_i (and IIM_i) as the "dummy OIM (resp. IIM) of P_i"

In the following, we call an ideal functionality $\mathcal{G}$ *standard* if $\mathcal{G}$ i) immediately notifies the adversary upon receiving input from an (honest) party, and ii) is standard corruption[10], and iii) only gives delayed outputs to parties.

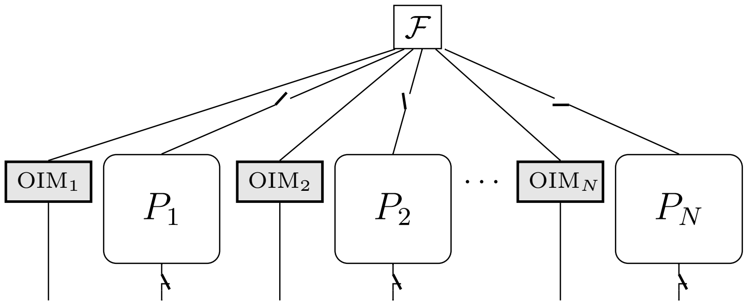


Fig. 2: Architecture of the Ideal Protocol $\mathtt{AG}([\mathcal{G}])$.

Fortified Functionalities. In contrast to functionalities in the adaptive UC security model, *fortified functionalities* do not pass the inputs and outputs of a party P_i corrupted *after* receiving input to the adversary $\mathcal{A}$ and also do not allow him to modify P_i's input and the *output to P_i's dummy* OIM, unless all parties P_j $(j = 1, \dots, N)$ are corrupted. $\mathcal{A}$ can only block an output or instruct the functionality to pass either the computed output or an error symbol $\perp$ to P_i's dummy OIM. If all parties are corrupted, $\mathcal{A}$ learns all inputs and outputs and may modify them arbitrarily (including the outputs to the dummy OIMs).

Definition 1 (Fortified Functionality). *Let $\mathcal{G}$ be a* non-reactive *standard ideal functionality interacting with N parties $P_1, \dots, P_N$ and $\mathcal{A}$. Define the* fortified functionality $[\mathcal{G}]$ *of $\mathcal{G}$ interacting with $P_1, \dots, P_N$, $\mathcal{A}$ and additionally N "dummy output interface modules* (OIMs)*"* $\mathrm{OIM}_1, \dots, \mathrm{OIM}_N$ *as follows:*

- $[\mathcal{G}]$ *internally runs an instance of $\mathcal{G}$.*
- $[\mathcal{G}]$ *initializes a counter $c = 0$.*
- *Upon receiving input from a party P_i, $[\mathcal{G}]$ forwards that input to $\mathcal{G}$.*
- *Each time $\mathcal{G}$ sends a notification to $\mathcal{A}$ upon receiving input from an (honest) party, $[\mathcal{G}]$ forwards that notification to $\mathcal{A}$.*
- $[\mathcal{G}]$ *forwards all delayed outputs of $\mathcal{G}$ to $\mathcal{A}$. Upon confirmation by $\mathcal{A}$, $[\mathcal{G}]$ forwards the output to the dummy* OIM *of the party for which $\mathcal{G}$ intended this output.*

[10] See Appendix B for a definition of standard corruption

– *Upon receiving* $(\texttt{corrupt}, P_i)$*,* $[\mathcal{G}]$ *does the following:*
  - *If* $[\mathcal{G}]$ *has* not yet *received input from* P_i*,* $[\mathcal{G}]$ *increments* c*, marks* P_i *as* corrupted before input *and forwards* $(\texttt{corrupt}, P_i)$ *to* $\mathcal{G}$*.*
  - *If* $[\mathcal{G}]$ *has already received input from* P_i*,* $[\mathcal{G}]$ *increments* c*, marks* P_i *as* corrupted after input *and forwards* $(\texttt{corrupt}, P_i)$ *to* $\mathcal{G}$*.*

– *If* $\mathcal{G}$ *outputs "corrupted" to* P_i *upon receiving* $(\texttt{corrupt}, P_i)$*,* $[\mathcal{G}]$ *forwards this to* P_i*.*[11]

– *Handling Parties* P_i *marked as* corrupted before input*:*
  - *If* $\mathcal{G}$ *sends the input of* P_i *to* $\mathcal{A}$*,* $[\mathcal{G}]$ *forwards that input to* $\mathcal{A}$*. Furthermore, if* $\mathcal{A}$ *sends a modified input value for* P_i*,* $[\mathcal{G}]$ *forwards that value to* $\mathcal{G}$*.*
  - *If* $\mathcal{G}$ *sends an output intended for* P_i *to* $\mathcal{A}$*,* $[\mathcal{G}]$ *sends that output to* $\mathcal{A}$*.* $\mathcal{A}$ *may instruct* $[\mathcal{G}]$ *to pass any output of his choice to* OIM_i*.*

– *Handling Parties* P_i *marked as* corrupted after input*:*
  - *If* $c < N$ *and* $\mathcal{G}$ *sends the input of* P_i *to* $\mathcal{A}$ *after receiving* $(\texttt{corrupt}, P_i)$*, ignore this message. Furthermore, if* $\mathcal{A}$ *sends a modified input value for* P_i*, ignore this value.*
  - *If* $c < N$ *and* $\mathcal{G}$ *sends the output intended for* P_i *to* $\mathcal{A}$*,* $[\mathcal{G}]$ *first notifies* $\mathcal{A}$ *that* OIM_i *is about to receive output.* $\mathcal{A}$ *may then instruct* $[\mathcal{G}]$ *to pass that output or* $\perp$ *to* OIM_i*.*

– *If* $c = N$*, send all inputs and outputs to* $\mathcal{A}$*. In addition,* $\mathcal{A}$ *may determine the outputs of all dummy OIMs in this case.*

– *All other messages between* $\mathcal{A}$ *and* $\mathcal{G}$ *are forwarded.*

– *If* $\mathcal{A}$ *sends* $(\mathsf{output}, \tilde{y}, P_i)$*,* $[\mathcal{G}]$ *outputs* $\tilde{y}$ *to* P_i *if* $[\mathcal{G}]$ *has marked* P_i*.*[12]

Reactive Case. If $\mathcal{G}$ is *reactive*, then $[\mathcal{G}]$ is defined as above except that $[\mathcal{G}]$ additionally interacts with N "dummy input interface modules" $\mathrm{IIM}_1, \ldots, \mathrm{IIM}_N$ as follows: Upon receiving input from an honest party P_i, $[\mathcal{G}]$ forwards that input to $\mathcal{G}$ and sends `open` to the dummy IIM of P_i. $[\mathcal{G}]$ forwards all inputs provided by a party P_i for rounds $u \geq 2$ to the adversary $\mathcal{A}$ if P_i is marked. Furthermore, upon receiving an input provided by the dummy IIM of a party P_i who is marked as corrupted before input, $[\mathcal{G}]$ forwards this input to $\mathcal{A}$ who may then modify it. However, upon receiving an input provided by the dummy IIM of a party P_i who is *not* marked as corrupted before input, $[\mathcal{G}]$ does not forward this input to $\mathcal{A}$ and does not allow $\mathcal{A}$ to modify it.

By construction, $\texttt{AG}([\mathcal{G}])$ captures our desired security goal: i) $[\mathcal{G}]$ ensures that corrupting a party P_i (via an `online-attack` instruction) *after* it received

[11] Note that this output is always allowed by the control function. In particular, it is allowed when a party is corrupted before receiving input because the respective air-gap switch, which is disconnected at that point, is replaced by a standard channel if the party is corrupted via a `physical-attack` instruction or immediately connected if a party is corrupted via an `online-attack` instruction.

[12] Note that the adversary is able to determine what a corrupted party outputs. However, he cannot modify the output of the (unhackable) dummy OIM of a party corrupted after receiving input (unless $c = N$).

its (first) input does not enable the adversary $\mathcal{A}$ to learn or modify P_i's input(s) and result(s) of the MPC (i.e. outputs of P_i's dummy OIM) unless *all* parties P_j ($j = 1, \ldots, N$) are corrupted, and ii) P_i's initially disconnected air-gap switches ensure that $\mathcal{A}$ can only corrupt a party P_i via `online-attack` instructions prior to receiving input if P_i is online via its channel to the environment.

2.5 Notify Transport Mechanism and Activation Instructions

In the UC framework, the adversary is not activated when a party provides input or receives subroutine output from a sub-party and is therefore not able to corrupt it during this communication. In our setting of hacking adversaries, this is undesirable because it does not capture the possibility of parties being remotely hacked when they are online during immediate communication.

As a motivating example, consider a hackable party P that is connected to the environment and the adversary $\mathcal{A}$ via standard channels. Furthermore, P is also connected to an unhackable sub-party P' via a standard channel. Upon receiving input, P sends a message containing secret data (e.g. shares of its input) to P'. P' then sends a notification message to P who immediately *erases* all secret data after being activated again. As this message delivery is *immediate*, i.e. $\mathcal{A}$ is not activated during the communication between P and P', he is unable to corrupt P before P has erased its secret data and sent it to an unhackable sub-party even though P has been *online all the time.*

To address this problem, we introduce a *notify transport mechanism* that activates $\mathcal{A}$ (under certain conditions) upon immediate message delivery.

Notify Transport Mechanism. Let μ, μ' by two ITIs such either

- μ and μ' are protocol (sub-)parties that are not combined, or
- μ is a protocol (sub-)party and μ' is an ideal functionality that is not a fortified functionality, or vice versa.

If μ sends an `external write` instruction addressed to μ' and the control function allows this instruction, then the adversary $\mathcal{A}$ is activated with a *notify transport message* consisting of μ''s ID. Upon activation, $\mathcal{A}$ may forward the notify transport message to the environment $\mathcal{Z}$ or execute an `online-attack` instruction (or do nothing). If activated, $\mathcal{Z}$ may only activate $\mathcal{A}$ again who may then carry out an `online-attack` instruction (or do nothing). Afterwards, the `external write` instruction is carried out.

Note that, upon receiving a notify transport message, $\mathcal{A}$ is not allowed to block the message sent by μ or activate another party.

Why Exclude Fortified Functionalities? The notify transport mechanism does not apply to the communication between the dummy parties and $[\mathcal{G}]$. This ensures that the ideal model adversary is not activated after a dummy party has sent its input to $[\mathcal{G}]$ *before* $[\mathcal{G}]$ receives this input. Note that he would otherwise be able to learn or modify a party's input and output through an `online-attack` instruction at a moment when the party has already received its input.

Activation Instructions. In the UC framework, protocol parties are activated via `external write` instructions. This mechanism cannot be applied to parties that are offline, however. For instance, consider a party P that wants to send messages to multiple parties via data diodes while being offline. In order to do so, P must be activated multiple times. This raises a problem since there is no way to activate P via an `external write` instruction.

In order to address this problem, we allow the adversary to send *activation instructions*. Formally, $\mathcal{A}$ may activate a party P by writing $(\texttt{activate}, P)$ on its outgoing message tape. P will then be activated.

2.6 Fortified UC Emulation

We now define the execution experiment in our framework by applying the rules specified in Sections 2.1 to 2.5 to the UC execution experiment:

Definition 2 (Fortified UC Execution Experiment). *An execution of a protocol σ with* adversary *$\mathcal{A}$ and an* environment *$\mathcal{Z}$ on input $a \in \{0,1\}^*$ and with security parameter $n \in \mathbb{N}$ is a run of a system of ITMs subject to the following restrictions:*

- *First, $\mathcal{Z}$ is activated on input $a \in \{0,1\}^*$. At each activation, $\mathcal{Z}$ may mark each channel that exists between $\mathcal{Z}$ and a protocol party either* online *or* offline*. In addition, for each channel to $\mathcal{Z}$, $\mathcal{Z}$ is informed upon each activation if it can receive output from that channel (cf. Section 2.2).*
- *The first ITI to be invoked by $\mathcal{Z}$ is the adversary $\mathcal{A}$. The corruption model is as specified in Section 2.3.*
- *$\mathcal{Z}$ may invoke a single instance of a* challenge protocol*, which is set to be σ by the experiment. The SID of σ is determined by $\mathcal{Z}$ upon invocation.*
- *$\mathcal{Z}$ may provide inputs to the adversary. In addition, $\mathcal{Z}$ may provide inputs to the parties of σ subject to the protocol architecture (cf. Section 2.1). (Note that among the parties that may receive input from $\mathcal{Z}$ are interface modules, cf. Section 2.4.)*
- *The adversary $\mathcal{A}$ may give subroutine outputs to $\mathcal{Z}$. In addition, $\mathcal{A}$ may send messages to the parties of σ subject to the protocol architecture (Section 2.1). At each activation, $\mathcal{A}$ is given the status (cf. Section 2.2). Moreover, $\mathcal{A}$ may activate a party through* `activate` *instructions (cf. Section 2.5).*
- *Each party of σ may send messages to the adversary, provide inputs to its sub-parties and give subroutine outputs to the parties of which it is a sub-party or to the environment $\mathcal{Z}$ subject to the protocol architecture (Section 2.1). Immediate messages may trigger the notify transport mechanism activating the adversary as specified in Section 2.5.*
- *At the end of the execution experiment, $\mathcal{Z}$ outputs a single bit.*

Denote by $\mathrm{Exec}_{\mathrm{FortUC}}(\sigma, \mathcal{A}, \mathcal{Z})(n, a) \in \{0,1\}$ *the output of the environment $\mathcal{Z}$ on input $a \in \{0,1\}^*$ and with security parameter $n \in \mathbb{N}$ when interacting with σ and $\mathcal{A}$ according to the above definition.*

We now define security in our framework in analogy to the UC framework:

Definition 3 (Emulation in the Fortified UC Framework). *Let π and ϕ be protocols. π is said to emulate ϕ in the Fortified UC framework, denoted by $\pi \geq_{\#\#} \phi$*[13]*, if for every* PPT*-adversary $\mathcal{A}$ there exists a* PPT*-adversary $\mathcal{S}$ such that for every* PPT*-environment $\mathcal{Z}$ there exists a negligible function* negl *such that for all $n \in \mathbb{N}, a \in \{0,1\}^*$ it holds that*

$$|\mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\pi, \mathcal{A}, \mathcal{Z})(n,a) = 1] - \mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\phi, \mathcal{S}, \mathcal{Z})(n,a) = 1]| \leq \mathsf{negl}(n)$$

Let π be a protocol with N *main* parties $P_1, \ldots, P_N$. We will later say that "π emulates ϕ for up to L parties under adversarial control" if emulation holds for all (real-model) PPT-adversaries $\mathcal{A}$ corrupting at most L parties $P \in \{P_1, \ldots, P_N\}$.

3 Properties of the Framework

In this section, we list some important properties of our new security notion.

Definition 4 (Emulation with Respect to the Dummy Adversary). *Define the* dummy adversary *$\mathcal{D}$ as follows: i) When receiving a message (sid, pid, m) from the environment $\mathcal{Z}$, $\mathcal{D}$ sends m to the party with extended identity (pid, sid). ii) When receiving* $(\texttt{physical-attack}, \mathcal{M})$ or $(\texttt{online-attack}, P)$ or $(\texttt{taint}, P)$ or $(\texttt{activate}, P)$ from $\mathcal{Z}$, $\mathcal{D}$ carries out that instruction. iii) When receiving m from the party with PID *pid* and SID *sid*, $\mathcal{D}$ sends (sid, pid, m) to $\mathcal{Z}$. iv) When receiving the instruction `status` from $\mathcal{Z}$, $\mathcal{D}$ sends the status $\mathcal{Z}$.

Let π and ϕ be protocols. π is said to emulate ϕ with respect to the dummy adversary *in the Fortified UC framework if there exists a* PPT*-adversary $\mathcal{S}_{\mathcal{D}}$ such that for every* PPT*-environment $\mathcal{Z}$ there exists negligible function* negl *such that for all $n \in \mathbb{N}, a \in \{0,1\}^*$ it holds that*

$$|\mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\pi, \mathcal{D}, \mathcal{Z})(n,a) = 1] - \mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\phi, \mathcal{S}_{\mathcal{D}}, \mathcal{Z})(n,a) = 1]| \leq \mathsf{negl}(n)$$

As in the UC framework, the dummy adversary is also complete in our framework and our security notion is also transitive:

Proposition 1 (Completeness of the Dummy Adversary). *Let π and ϕ be protocols. Then, $\pi \geq_{\#\#} \phi$ if and only if π emulates ϕ with respect to the dummy adversary in the Fortified UC framework.*

Proof (Idea). The proof is almost identical to the proof in the UC framework (cf. [4]). The only difference is that the environment $\mathcal{Z}_{\mathcal{A}}$, which internally runs a copy of a given adversary $\mathcal{A}$ and environment $\mathcal{Z}$, forwards the status to $\mathcal{A}$ each time $\mathcal{A}$ is activated in $\mathcal{Z}_{\mathcal{A}}$'s internal simulation. Note that $\mathcal{Z}_{\mathcal{A}}$ can obtain the status by sending `status` to the dummy adversary $\mathcal{D}$. $\square$

Proposition 2 (Transitivity). *Let π_1, π_2, π_3 be protocols. If $\pi_1 \geq_{\#\#} \pi_2$ and $\pi_2 \geq_{\#\#} \pi_3$ then it holds that $\pi_1 \geq_{\#\#} \pi_3$.*

[13] Think of "##" as a fence, i.e. part of a fortification.

Proof (Idea). The proof follows from the same argument as in the UC framework [4]. □

Our security notion is equivalent to adaptive UC security for "plain" protocols that do not have unhackable sub-parties. We first formally define plain protocols:

Definition 5 (Plain Protocols). *A protocol π is called* plain *if all of the following hold: i) all (sub-)parties in π are hackable ii) π only uses standard channels iii) π has no interface modules iv) each party in π is combined with all of its sub-parties v) each (sub-)party in π that is not a (dummy) party of an ideal (sub-)protocol is connected to the adversary via a standard channel vi) (sub-)parties in π only make calls to standard functionalities.*

Plain protocols can be naturally interpreted as protocols in the UC framework.

We will later use plain protocols that additionally have air-gap switches between specific ITIs as sub-protocols in our constructions. To this end, we define the following:

Given a plain protocol π, define $\overline{\pi}$ to be identical to π except for the following: All standard channels, except the ones between combined parties and between parties and the environment, are replaced with *initially disconnected air-gap switches*. Upon receiving input, each party connects all of its air-gap switches. We will later use protocols of the form $\overline{\mathtt{SC}(\mathcal{F})}$[14] as sub-protocols in our constructions.

We have the following theorem:

Theorem 1 (Equivalence with UC Emulation for Plain Protocols). *Let π and ϕ be plain protocols. Then,*

$$\pi \geq_{\#\#} \phi \iff \pi \geq_{\mathrm{UC}} \phi \iff \overline{\pi} \geq_{\#\#} \overline{\phi},$$

where $\geq_{\mathrm{UC}}$ denotes UC emulation with respect to adaptive *(PID-wise*[15]*) corruption.*

Proof (Idea). These statements follow from the fact that UC environments can easily simulate Fortified UC environments interacting with plain protocols and vice versa. This is because in a plain protocol the notify transport mechanism is only triggered if a party sends a message to a (standard) ideal functionality (which by convention immediately notifies the adversary upon input. Note that the notify transport mechanism is not triggered during communication between combined parties.) and the online state of a party in a plain protocol can be trivially derived. □

Let ρ^{ϕ} be a protocol that makes subroutine calls to a polynomial number of instances of a protocol ϕ. The following theorem guarantees that if π emulates ϕ in the Fortified UC framework then one can replace subroutine calls to instances of ϕ by subroutine calls to instances of π without losing security.

[14] Note that in the ideal protocol $\overline{\mathtt{SC}(\mathcal{F})}$, each dummy party is connected to $\mathcal{F}$ via an initially disconnected air-gap switch and to the environment via a standard channel.

[15] Recall that the PIDs of combined parties are equal in their first component (cf. Section 2.3). "PID-wise corruption" therefore means that parties with PIDs which are equal in their first component are corrupted together.

Theorem 2 (Composition Theorem). *Let π, ϕ, ρ be protocols. Then,*

$$\pi \geq_{\#\#} \phi \implies \rho^{\pi} \geq_{\#\#} \rho^{\phi}$$

Proof (Sketch). The proof is very similar to the proof of the composition theorem of the UC framework(cf. [4]). The two main differences are the following

The environment $\mathcal{Z}_\pi$, which internally runs a given environment $\mathcal{Z}$, the protocol ρ and all but one of the instances of π or ϕ that are called by ρ and interacts with the dummy adversary $\mathcal{D}$ and either π or ϕ as challenge protocol, behaves as in proof of the composition theorem of the UC framework (cf. [4]) and additionally does the following:

1. $\mathcal{Z}_\pi$ marks each channel to a party in the challenge protocol according to the online state of the respective calling party in ρ in its internal simulation. This ensures that the online states of the parties in the challenge protocol when interacting with $\mathcal{Z}_\pi$ are the same as when run as subroutines of ρ in an interaction with the environment $\mathcal{Z}$.
2. $\mathcal{Z}_\pi$ determines if a party E in its internal simulation who is a calling party of a party P in the challenge protocol is online via a channel C to P by deriving the relevant information from the status reported by the adversary (which contains information about whether P is online via a channel $C' \neq C$ to another ITM $\mu \neq E$) and by checking whether it can receive output[16] via C from P.[17] This ensures that the online state of E when internally run by $\mathcal{Z}_\pi$ is the same as when running in an interaction between ρ and the environment $\mathcal{Z}$.
3. If $\mathcal{Z}$ sends $(\texttt{physical-attack}, \mathcal{M})$ in $\mathcal{Z}_\pi$'s internal simulation, $\mathcal{Z}_\pi$ sends $(\texttt{physical-attack}, \mathcal{M}')$ to $\mathcal{D}$, where the parties in $\mathcal{M}'$ are the main parties of $\mathcal{Z}_\pi$'s challenge protocol who are the respective sub-parties of the parties in $\mathcal{M}$. Furthermore, if $\mathcal{Z}$ sends $(\texttt{online-attack}, P)$ for a party P in $\mathcal{Z}_\pi$'s challenge protocol, $\mathcal{Z}_\pi$ forwards $(\texttt{online-attack}, P)$ to $\mathcal{D}$. Finally, if $\mathcal{Z}$ sends $(\texttt{online-attack}, E)$ for a party E in $\mathcal{Z}_\pi$'s internal simulation, $\mathcal{Z}_\pi$ checks the online state of E in its internal simulation and ignores this instruction if E is offline or internally carries out this instruction if E is online.

The simulator $\mathcal{S}$, which internally runs the dummy adversary $\mathcal{D}$ and copies of the simulator $\mathcal{S}_\pi$ implied by $\pi \geq_{\#\#} \phi$, and interacts with a given environment $\mathcal{Z}$ and the protocol ρ^{ϕ}, behaves as in proof of the composition theorem of the UC framework (cf. [4]) and additionally does the following:

$\mathcal{S}$ keeps track of a "simulated" status as follows:

1. If E is a party who is *not* a party of an instance of ϕ and C is channel between E and an ITM who is also *not* a party of an instance of ϕ, then E is

[16] Recall that for each channel to $\mathcal{Z}_\pi$, $\mathcal{Z}_\pi$ is informed upon each activation if it can receive output from that channel, cf. Section 2.2.

[17] Recall that, by definition, E is online via channel C if and only if E can receive output via C from the sub-party P and P is online via a channel $C' \neq C$ to another ITM $\mu \neq E$, cf. Section 2.2.

online via C in $\mathcal{S}$'s "simulated" status if and only if the status $\mathcal{S}$ receives[18] from the experiment states that E is online via C.

2. At each activation, $\mathcal{S}$ internally hands the copies of the simulator $\mathcal{S}_\pi$ a status that is derived from the status that $\mathcal{S}$ receives from the experiment (by taking the information about the channels involving the parties in the respective instance of ϕ). Afterwards, $\mathcal{S}$ sends the instruction `status` to all copies of $\mathcal{S}_\pi$, receiving a status from each copy.
3. If P is a party of an instance of ϕ and C' is a channel between P and any other ITM, then P is online via C' in $\mathcal{S}$'s "simulated" status if and only if the status reported by the respective copy of the simulator $\mathcal{S}_\pi$ for that instance of ϕ claims P to be online via C'.
4. If E is a party who is *not* a party of an instance of ϕ and C'' is channel between E and a party P who is a party of an instance of ϕ, then E is online via C'' in $\mathcal{S}$'s "simulated" status if and only if the status that $\mathcal{S}$ receives from the experiment states that E is online via C'' *and* the status reported by the respective copy of $\mathcal{S}_\pi$ for that instance of ϕ claims that P is online via a channel $\widetilde{C} \neq C''$ to another ITM $\mu \neq E$.

When the environment $\mathcal{Z}$ sends the instruction `status` to $\mathcal{S}$, then $\mathcal{S}$ reports the "simulated" status to $\mathcal{Z}$. When $\mathcal{Z}$ sends $(\texttt{online-attack}, E)$ for a party E who is *not* a party of an instance of ϕ, then $\mathcal{S}$ checks the online state of E that is implied by the "simulated" status and ignores this instruction if E is offline or carries out the instruction if E is online. When $\mathcal{Z}$ sends $(\texttt{online-attack}, P)$ for a party P that is a party of an instance of ϕ, then $\mathcal{S}$ forwards this message to the respective copy of $\mathcal{S}_\pi$ for that instance.

Theorems 1 and 2 allow for modular composition with UC-secure protocols. This will be used in the subsequent sections where we will construct protocols $\rho^{\overline{\texttt{SC}(\mathcal{F})}}$ such that $\rho^{\overline{\texttt{SC}(\mathcal{F})}} \geq_{\#\#} \texttt{AG}(\underline{[\mathcal{G}]})$ for some fortified functionality $\underline{[\mathcal{G}]}$. Given a protocol π such that $\pi \geq_{\mathrm{UC}} \texttt{SC}(\mathcal{F})$, we can replace all instances of $\overline{\texttt{SC}(\mathcal{F})}$ by instances of $\overline{\pi}$ as follows: By Theorem 1 we have that $\overline{\pi} \geq_{\#\#} \overline{\texttt{SC}(\mathcal{F})}$. Hence, $\rho^{\overline{\pi}} \geq_{\#\#} \rho^{\overline{\texttt{SC}(\mathcal{F})}}$ by Theorem 2. Therefore, $\rho^{\overline{\pi}} \geq_{\#\#} \texttt{AG}([\mathcal{G}])$ by transitivity (Proposition 2).

Remark 2 (Further Discussion of the Composition Theorem).

1. Not giving the environment the possibility to learn if it can receive output via a channel between the environment and a party does not lead to a composable security notion. As an example, consider a two-party protocol π that only uses standard channels. In particular, parties P_1, P_2 in π are always online. Consider a protocol π' that is identical to π except that the parties are connected to the environment via initially-connected air-gap switches. Each party in π' disconnects its air-gap switch to the environment upon receiving input. Before giving output, a party connects its air-gap switch to

[18] Recall that, at each activation, the adversary $\mathcal{S}$ gets informed via which channels each party is online, cf. Section 2.2.

the environment again. It is easy to see that π emulates π' according to this modified notion (where environments are not able learn if they can receive output).

Now, consider a protocol ρ^π that consists of two parties E_1, E_2 making subroutine calls to one instance of π, i.e. P_i is a sub-party of E_i in ρ^π. The protocol architecture of ρ^π is such that each E_i is offline via all channels except to P_i. It holds that ρ^π does not emulate $\rho^{\pi'}$. This is because the parties E_i in ρ^π are online but offline in $\rho^{\pi'}$. Hence, an environment who instructs the dummy adversary $\mathcal{D}$ to send an `online-attack` instruction to E_i can easily distinguish these two protocols by observing if that party becomes corrupted or not.

2. Stipulating that a party is online if it can receive input from the environment, i.e. not giving the environment the possibility to modify the online state of its channels to the parties (by marking the channel), neither leads to a composable security notion. As an example, consider a two-party protocol π where each party P_i is connected to the environment via a standard channel. Furthermore, both parties are connected to the adversary via initially-connected air-gap switches. Let π' be identical to π except that all air-gap switches to the adversary are initially disconnected. It is easy to see that π emulates π' according to this modified notion (where a party is online if it can receive input from the environment).

   Now, consider a protocol ρ^π that consists of two parties E_1, E_2 making subroutine calls to one instance of π, i.e. P_i is a sub-party of E_i in ρ^π. The protocol architecture of ρ^π is such that each E_i is offline via all channels except to P_i. By construction, the parties P_i in ρ^π are still initially online. However, the parties P_i in $\rho^{\pi'}$ are initially offline. Hence, an environment who instructs the dummy adversary $\mathcal{D}$ to send an `online-attack` instruction to P_i can easily distinguish these two protocols by observing if that party becomes corrupted or not.

3. Only giving the adversary the current online state of a party as opposed to the information via which channels a party is online does also not lead to a composable security notion. As an example, consider a two-party protocol π where the environment is connected to the parties P_1, P_2 via initially-connected air-gap switches (i.e. the environment operates these air-gap switches). Furthermore, both parties in π are connected to the adversary via initially-disconnected air-gap switches. Let π' be identical to π except that all air-gap switches to the adversary are initially-connected. It is easy to see that π emulates π' according to this modified notion (where the adversary is only given the current online state of each party).

   Now, consider a protocol ρ^π that consists of two parties E_1, E_2 making subroutine calls to one instance of π, i.e. P_i is a sub-party of E_i in ρ^π. Each E_i is connected to the environment and adversary via initially-connected air-gap switches. On any input, each E_i disconnects its air-gap switch to the environment. On input 0, each E_i disconnects its air-gap switch to the adversary but lets its air-gap switch to P_i remain connected. In contrast, on input 1, each E_i disconnects its air-gap switch to P_i but lets its air-gap switch to the adversary remain connected.

It holds that ρ^π does not emulate $\rho^{\pi'}$. This can be argued as follows: Consider the environment $\mathcal{Z}$ interacting with the dummy adversary $\mathcal{D}$ that randomly chooses a bit b, hands b to E_1 as input, and then instructs $\mathcal{D}$ to send an `online-attack` instruction to E_1. By construction, the party E_1 will then be corrupted or not depending on the input b. More specifically, E_1 will be corrupted if $b = 1$ and remain uncorrupted otherwise (since P_1 in π is offline). However, in the protocol $\rho^{\pi'}$, E_1 is always online regardless of its inputs. This is because E_1 is either online via its channel to the adversary or to the party in π' who, by construction, is always online. Therefore, a potential simulator interacting with $\rho^{\pi'}$ who only gets the online state of the parties cannot decide if the online attack on E_1 should be carried out or ignored.

4 Construction for Non-Reactive Functionalities

In this section, we present a construction for realizing the fortified functionality $[\mathcal{G}]$ of every *non-reactive* (standard adaptively well-formed[19]) functionality $\mathcal{G}$.

The broad idea is to have the parties $P_1, \ldots, P_N$ send *encrypted shares* of their inputs via data diodes in an *offline sharing phase* and subsequently use these shares to compute the desired function in an *online compute phase*. This, however, cannot be done straightforwardly. To begin with, the parties are not able to retrieve public keys themselves in the sharing phase since this would necessitate going online, making them susceptible to online attacks. Therefore, each party P_i sends its shares to an unhackable sub-party called *encryption unit* (Enc-unit) via a data diode. The Enc-unit retrieves the public keys and sends encrypted shares to hackable sub-parties of the designated receivers called *buffers* (note that since the parties $P_1, \ldots, P_N$ are offline they cannot receive messages).

Furthermore, each message has to be authenticated so that the adversary cannot change the input of a party by modify the messages it sends. One could do this with an additional unhackable "authentication unit" which signs each ciphertext or have the Enc-unit sign all ciphertexts. However, since we want to use as few and as simple unhackable sub-parties as possible, we take a different approach. Each party P_i sends its shares together with valid *signatures* to its Enc-unit. The verification key is sent, over an intermediary sub-party called *join* (J), to a hackable sub-party called *registration module* (RM) that disconnects itself from J after receiving input and forwards the verification key to a *public bulletin board* via a data diode. Once a party P_i has sent all of its shares, it erases everything except for its own share, its verification key and its decryption key. In order for this sign-then-encrypt approach to be secure, we assume that the PKE scheme is non-malleable (*IND-parallel-CCA-secure*) and that the digital signature is unforgeable (*EUF-naCMA secure*) and also satisfies a property we call *length-normal*, guaranteeing that signatures of messages of equal length are

[19] Cf. Appendix B for a definition of adaptively well-formed functionalities.

also of equal length. This prevents an adversary from learning information of plaintexts based on the length of their ciphertext. Each party P_i is connected to its sub-party J via an *initially disconnected air-gap switch* in order to prevent the adversary from corrupting P_i's RM but not P_i before P_i has received its input.

In the compute phase, the adversary must be prevented from using values that are *different* from the shares sent by the honest parties to the corrupted parties in the sharing phase. Otherwise, he would be able to modify the inputs of the parties who were honest during the sharing phase. The parties P_i therefore not only use the shares they received but also the signatures of these shares and the registered verification keys during the compute phase. The result of the compute phase is a special "error symbol" if not all signatures are valid. Since the signing keys were erased at the end of the sharing phase, the adversary cannot generate new valid signatures for parties P_i corrupted after receiving input. He is also unable to revoke the verification key of such parties since this would require corrupting the respective RM, which is impossible since that party is offline.

Moreover, an adversary could *swap* a message in the sharing phase addressed to (the buffer of) an honest party P_j with a ciphertext of a share and signature received by a corrupted party (by encrypting that tuple with the respective public key). Furthermore, an adversary controlling at least two parties P_i, P_j knows two shares and valid signatures of each party and could use one of these tuples *twice* in the compute phase. To prevent these attacks, a party P_i signs each share *along with the designated receiver's PID*. In addition, a party P_i also includes its own PID in each message it sends to prevent the adversary from reusing messages sent by honest parties for the parties corrupted before receiving input.

Finally, one cannot simply send the result of the compute phase to a party P_i since this would allow the adversary to learn and modify the output of the parties corrupted after receiving input. Instead, we introduce another unhackable sub-party called *output interface module* (OIM). Each party P_i sends not only the shares of its input x_i but also shares of a *random pad* r_i and of a *MAC key* k_i in the sharing phase. Furthermore, each party P_i sends r_i and k_i to its OIM via a data diode. In the compute phase, the parties will then use these shares to compute $(y_i + r_i, \mathrm{Mac}(k_i, y_i + r_i))$, where y_i is the desired output value (of party P_i). Each party then sends its result to its OIM, which will check authenticity by verifying the MAC tag and, if correct, reconstruct and output the value y_i.

In the following, we will take a modular approach and define a functionality $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ that implements the verification of the input values in the compute phase as well as the subsequent multi-party computation on the shares. Using Theorems 1 and 2, we will be able to replace the sub-protocol $\overline{\mathtt{SC}(\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}})}$ in our construction with an existing adaptively UC-secure protocol (cf. Remark 4).

We first define the functionality $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$.

Construction 1 *Let $\mathcal{G}$ be a* non-reactive *standard adaptively well-formed ideal functionality. $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ proceeds as follows, running with parties $P_1, \ldots, P_N$ and adversary $\mathcal{A}$ and parametrized with a digital signature scheme* SIG *and a message authentication code* MAC.

1. *Initialize the Boolean variable* $\mathtt{verify} = \mathtt{true}$.

2. *Upon receiving input from party P_i, store it and send $(\texttt{received}, P_i)$ to $\mathcal{A}$. Upon receiving $(\texttt{confirmed}, P_i)$ from $\mathcal{A}$, mark P_i as* input given.

Consistency Check

3. *Once each party has been marked as* input given, *check if each stored input is of the form $\overline{\mathsf{vk}}_i = (\mathsf{vk}_1^{(i)}, \ldots, \mathsf{vk}_N^{(i)})$, $(s_{ji}, r_{ji}, k_{ji}, \sigma_{ji})$ $(j = 1, \ldots, N)$.*
   (i) If no, set $\texttt{verify} = \texttt{false}$.
   (ii) If yes, check if $\overline{\mathsf{vk}}_1 = \cdots = \overline{\mathsf{vk}}_N$.
      (A) If this does not hold, set $\texttt{verify} = \texttt{false}$.
      (B) Else, set $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n) = (\mathsf{vk}_1^{(1)}, \ldots, \mathsf{vk}_N^{(1)})$. For all $i = 1, \ldots, N$, check if $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, s_{ji}, r_{ji}, k_{ji}, \sigma_{ji}) = 1$ for all $j = 1, \ldots, N$.
         (a) If this does not hold for every i, j, set $\texttt{verify} = \texttt{false}$.
         (b) Else, proceed with Item 4.

Reconstruction and Computation

4. *For each $i = 1, \ldots, N$, compute $x_i = s_{i1} + s_{i2} + \cdots + s_{iN}$, $k_i = k_{i1} + k_{i2} + \cdots + k_{iN}$ and $r_i = r_{i1} + r_{i2} + \cdots + r_{iN}$.*
5. *Internally run $\mathcal{G}$ on input $(x_1, \ldots, x_N)$. Let $(y_1, \ldots, y_N)$ be the output of $\mathcal{G}$. For all $i = 1, \ldots, N$, compute $o_i = y_i + r_i$ and $\theta_i \leftarrow \mathrm{Mac}(k_i, o_i)$.*
6. *If party P_i requests an output, proceed as follows:*
   (i) If $\texttt{verify} = \texttt{false}$, send a private delayed output $\perp$ to P_i.
   (ii) Else, if Item 5 has already been carried out, send a private delayed output (o_i, θ_i) to P_i.
7. *$\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ is standard corruption (cf. Appendix B). Once all parties are corrupted, $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ sends its private randomness to $\mathcal{A}$ (Note that this ensures that $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ is also adaptively well-formed).*
8. *All other messages between $\mathcal{A}$ and $\mathcal{G}$ are ignored.*

Let $\mathcal{F}_{\mathsf{reg}}$ be the public bulletin board functionality (cf. Appendix B.5 for a definition). Let $\mathrm{PKE} = (\mathrm{Gen}_{\mathrm{PKE}}, \mathrm{Enc}, \mathrm{Dec})$ be a public-key encryption scheme, $\mathrm{SIG} = (\mathrm{Gen}_{\mathrm{SIG}}, \mathrm{Sig}, \mathrm{Vrfy}_{\mathrm{SIG}})$ a digital signature scheme and $\mathrm{MAC} = (\mathrm{Gen}_{\mathrm{MAC}}, \mathrm{Mac}, \mathrm{Vrfy}_{\mathrm{MAC}})$ a message authentication code (cf. Appendix B.6 for a definition of these primitives). Given a *non-reactive* standard adaptively well-formed functionality $\mathcal{G}$, we next define our protocol $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$ for realizing $\texttt{AG}([\mathcal{G}])$. (For a definition of $\texttt{AG}([\mathcal{G}])$, see Section 2.4.)

Construction 2 *Define the protocol $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$ as follows:*
Architecture: *See Fig. 3 for a graphical depiction.*

Offline Sharing Phase

Upon input x_i, each party P_i $(i = 1, \ldots, N)$ does the following:

– Disconnect *air-gap switch to the environment.*
– *Generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$, $k_i \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$, $(\mathsf{sgk}_i, \mathsf{vk}_i) \leftarrow \mathrm{Gen}_{\mathrm{SIG}}(1^n)$ and a random pad $r_i \leftarrow \{0,1\}^{p_i(n)}$.*
– *Generate shares $s_{i1} + s_{i2} + \cdots + s_{iN} = x_i$ and $k_{i1} + k_{i2} + \cdots + k_{iN} = k_i$ and $r_{i1} + r_{i2} + \cdots + r_{iN} = r_i$.*
– Connect *air-gap switch to J.*
– *Send (k_i, r_i) to* OIM *and $(\mathsf{pk}_i, \mathsf{vk}_i)$ to J.*

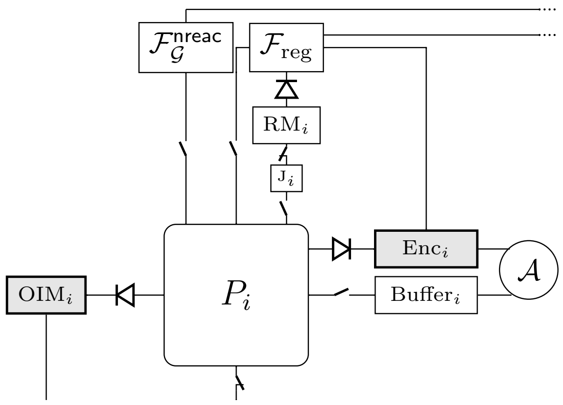

$\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$
$\mathcal{F}_{\text{reg}}$
RM$_i$
J$_i$
Enc$_i$
$\mathcal{A}$
OIM$_i$
P_i
Buffer$_i$

Fig. 3: Architecture of $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$. Each party P_i has 3 hackable sub-parties, called *buffer*, *registration module* (RM) and *join* (J), and 2 unhackable sub-parties, called Enc(-unit) and OIM. Buffer and Enc-unit are connected to the adversary via standard channels. All air-gap switches, except for P_i's air-gap switch to the environment and the RM's air-gap switch to J, are initially *disconnected.*

- *Create signatures* $\sigma_{ij} \leftarrow \text{Sig}(\mathsf{sgk}_i, P_j, s_{ij}, r_{ij}, k_{ij})$ $(j = 1, \ldots, N)$
- *Send* $(P_j, s_{ij}, r_{ij}, k_{ij}, \sigma_{ij})$ $(j \in \{1, 2, \ldots, m\} \setminus \{i\})$ *to* Enc-*unit*
- Erase *everything except for* $(s_{ii}, r_{ii}, k_{ii}, \sigma_{ii})$, vk_i *and* sk_i.

Registration module and J: *On input* $(\mathsf{pk}_i, \mathsf{vk}_i)$ *to J, J forwards the input to RM. RM then* disconnects *air-gap switch to J and registers* pk_i *and* vk_i *by sending these keys to the public bulletin-board functionality* $\mathcal{F}_{\mathsf{reg}}$.

Enc-unit: *Receive a list* $L = \{(P_j, v_j)\}_{j=\{1,\ldots,N\}\setminus\{i\}}$ *from one's main party* P_i. *At each activation, for each* $(P_j, v_j) \in L$, *request* pk_j *belonging to* P_j *from* $\mathcal{F}_{\mathsf{reg}}$. *If retrievable, compute* $c_{ij} \leftarrow \text{Enc}(\mathsf{pk}_j, v_j)$, *send* (P_i, c_{ij})[20] *to buffer of* P_j *and delete* (P_j, v) *from L. Then, go into idle mode.*

Buffer: *Store each message received. On input* `retrieve`, *send all stored messages to one's main party.*

Online Compute Phase

Having completed its last step in the sharing phase, a party P_i *does the following:*

- Connect *air-gap switches to buffer, to* $\mathcal{F}_{\mathsf{reg}}$ *and to* $\mathcal{F}_{\mathcal{G}}$.
- *Request from* $\mathcal{F}_{\mathsf{reg}}$ *all verification keys* $\{\mathsf{vk}_l\}_{l\in\{1,\ldots,N\}\setminus\{i\}}$ *registered by the other parties' registration modules. If not all verification keys can be retrieved yet, go into idle mode and request again at the next activation.*
- *Send* `retrieve` *to buffer and check if the buffer sends at least* $N-1$ *messages. If no, go into idle mode and when activated again send* `retrieve` *and check*

[20] Sending the sender's PID as prefix is not necessary but simplifies the discussion. Note that for (P_i, c) we also say that "c is addressed as coming from party P_i".

again. If yes, check if one has received from each party P_j a set $\mathcal{M}_j = \{(P_j, \tilde{c})\}$ with the following property $(*)$ (Validity Check)*:*
There exists a tuple $(P_j, \hat{s}_{ji}, \hat{r}_{ji}, \hat{k}_{ji}, \hat{\sigma}_{ji})$ and a $(P_j, c) \in \mathcal{M}_j$ such that:

- $\mathrm{Dec}(\mathsf{sk}_i, c) = (P_j, \hat{s}_{ji}, \hat{r}_{ji}, \hat{k}_{ji}, \hat{\sigma}_{ji})$ *and* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, \hat{s}_{ji}, \hat{r}_{ji}, \hat{k}_{ji}, \hat{\sigma}_{ji}) = 1$
- *For all $(P_j, \tilde{c}) \in \mathcal{M}_j$ it holds that either* $\mathrm{Dec}(\mathsf{sk}_i, \tilde{c}) = (P_j, \hat{s}_{ji}, \hat{r}_{ji}, \hat{k}_{ji}, \hat{\sigma}_{ji})$ *or $(P_j, \tilde{c})$ is* "invalid"*, i.e., either decrypts to a tuple $(P_j, \tilde{s}_{ji}, \tilde{r}_{ji}, \tilde{k}_{ji}, \tilde{\sigma}_{ji})$ such that* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, \tilde{s}_{ji}, \tilde{r}_{ji}, \tilde{k}_{ji}, \tilde{\sigma}_{ji}) = 0$*, or decrypts to a tuple $(P', \tilde{s}_{ji}, \tilde{r}_{ji}, \tilde{k}_{ji}, \tilde{\sigma}_{ji})$ such that $P' \neq P_j$, or does not decrypt correctly.*

If this does not hold, send $\perp$ to $\mathcal{F}_{\mathcal{G}}$. Else, send all retrieved verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ as well as all tuples $(\hat{s}_{ji}, \hat{r}_{ji}, \hat{k}_{ji}, \hat{\sigma}_{ji})$ $(j \in \{1, \ldots, N\})$ to $\mathcal{F}_{\mathcal{G}}$.

Online Output Phase
Having completed its last step in the compute phase, a party P_i requests output from $\mathcal{F}_{\mathcal{G}}$ and forwards that output to OIM.

OIM: *Store the first input (k_i, r_i) from one's main party. On second input (o_i, θ_i) or $\perp$ from one's main party, do the following: If the received value equals $\perp$, output $\perp$. Otherwise, check if* $\mathrm{Vrfy}_{\mathrm{MAC}}(k_i, o_i, \theta_i) = 1$ *and output $y_i = o_i + r_i$ if this holds, and $\perp$ otherwise.*

Remark 3. Note that we do not model how to reuse modules such as the registration modules that stay disconnected throughout the protocol execution. In practice, one may assume, e.g., a physical reset mechanism for these modules.

We will prove that $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$ emulates the ideal protocol $\mathtt{AG}([\mathcal{G}])$ in the Fortified UC framework for adversaries corrupting at most $N-1$ parties $P \in \{P_1, \ldots, P_N\}$ under the assumptions that PKE is IND-parallel-CCA-secure, SIG is EUF-naCMA-secure and length-normal and MAC is EUF-1-CMA-secure (cf. Appendix B.6 for a formal definition of these security notions).

Next, we define the following experiment, which will be used in the proof.

Definition 6 (Auxiliary Experiment). *The experiment* $\mathsf{Exp}^{\mathrm{aux}}_{\mathcal{A}(z),\mathrm{PKE},\mathrm{SIG}}(n)$ *is defined as follows: At the beginning, the experiment generates keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$ *and* $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathrm{Gen}_{\mathrm{SIG}}(1^n)$*. On input $1^n, z$ and* pk*, the adversary $\mathcal{A}$ may then* non-adaptively *send queries to a signing oracle* $\mathcal{O}_{\mathrm{Sig}(\mathsf{sgk},\cdot)}$*. Afterwards, the experiment sends* vk *to $\mathcal{A}$. $\mathcal{A}$ may then send a message of the form* $(\mathtt{prf}_1, \mathtt{prf}_2, m)$ *to the experiment. The experiment then computes* $\sigma \leftarrow \mathrm{Sig}(\mathsf{sgk}, \mathtt{prf}_2, m)$*,* $c^* \leftarrow \mathrm{Enc}(\mathsf{pk}, \mathtt{prf}_1, m, \sigma)$*, and sends c^* to $\mathcal{A}$. During the experiment, $\mathcal{A}$ may send a single* parallel *query to a decryption oracle* $\mathcal{O}_{\mathrm{Dec}(\mathsf{sk},\cdot)}$ *subject to the restriction that the query does not contain c^*. At the end of the experiment, $\mathcal{A}$ sends a tuple (m', σ') to the experiment. The experiment then checks if* $\mathrm{Vrfy}_{\mathrm{SIG}}(vk, m', \sigma') = 1$ *and m' has not been sent to* $\mathcal{O}_{\mathrm{Sig}(\mathsf{sgk},\cdot)}$ *before. If this holds, the experiment outputs* 1 *and* 0 *otherwise.*

We have the following lemma.

Lemma 1. *If* PKE *is IND-pCCA-secure and* SIG *EUF-naCMA-secure, then for every* PPT*-adversary $\mathcal{A}$ and all $z \in \{0,1\}^*$, there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\text{aux}}_{\mathcal{A}(z),\text{PKE},\text{SIG}}(n) = 1] \leq \mathsf{negl}(n)$$

Proof (Sketch). Assume there exists an adversary $\mathcal{A}$ that wins in the experiment $\mathsf{Exp}^{\text{aux}}_{\text{PKE},\text{SIG},\mathcal{A}(z)}(n)$ with non-negligible probability. Since PKE is IND-pCCA-secure, one can replace c^* by $c' \leftarrow \text{Enc}(\mathsf{pk}, 0^L)$, where $L = |(\mathtt{prf_1}, m, \sigma)|$, incurring only a negligible loss in $\mathcal{A}$'s success probability. Then, one can directly construct an adversary $\mathcal{A}'$ out of $\mathcal{A}$ that breaks the EUF-naCMA-security of SIG with non-negligible probability. $\mathcal{A}'$ simply internally simulates the experiment $\mathsf{Exp}^{\text{aux}}_{\text{PKE},\text{SIG},\mathcal{A}(z)}(n)$ for $\mathcal{A}$ using his signing oracle and c' for c^*. Once $\mathcal{A}$ sends a tuple (m, σ) to the experiment $\mathsf{Exp}^{\text{aux}}_{\text{PKE},\text{SIG},\mathcal{A}(z)}(n)$, $\mathcal{A}'$ sends (m, σ) to the EUF-naCMA experiment. $\mathcal{A}'$ then wins in the EUF-naCMA experiment if and only if $\mathcal{A}$ wins in the experiment $\mathsf{Exp}^{\text{aux}}_{\text{PKE},\text{SIG},\mathcal{A}(z)}(n)$. □

We will use the above experiment to show that an environment $\mathcal{Z}$ cannot send "fake messages" (P_i, c') to an honest party P_j addressed as coming from a party P_i that has *not* been corrupted before receiving input such that i) c' was not generated by the Enc-unit of P_i and ii) (P_i, c') is accepted by P_j.

Next, we define the simulator for the dummy adversary.

Definition 7 (Definition of the Simulator). *Define the simulator* Sim *interacting with an environment $\mathcal{Z}$ and a fortified ideal functionality $[\mathcal{G}]$ as follows:*

1. *At the beginning,* Sim *internally defines N parties corresponding to the parties in $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$. Throughout the simulation,* Sim *will keep track of the online state of these parties by marking them as* online *or* $\mathsf{offline}$*. At the beginning,* Sim *marks these parties according to the initial online states of the dummy parties in the ideal protocol.*
2. Sim *initializes a Boolean variable* $\mathtt{verify} = \mathtt{true}$.
3. Sim *carries out the* `physical-attack` instruction received from $\mathcal{Z}$ on its first activation. Sim carries out an $(\mathtt{online\text{-}attack}, P_i)$ instruction only if Sim has marked party P_i as online.
4. Each time $\mathcal{Z}$ sends `status`, Sim sends the set of markings of each party.
5. Throughout the simulation, Sim reports the respective notify transport tokens to $\mathcal{Z}$ (note that we will not mention them anymore in the following).
6. Sim generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \text{Gen}_{\text{PKE}}(1^n)$, $k_i \leftarrow \text{Gen}_{\text{MAC}}(1^n)$ and $(\mathsf{sgk}_i, \mathsf{vk}_i) \leftarrow \text{Gen}_{\text{SIG}}(1^n)$ for each party P_i that was not corrupted *before* receiving input.
7. For each i such that party P_i is honest, Sim reports $(\mathtt{registered}, sid', RM_i, \mathsf{pk}_i, \mathsf{vk}_i)$. If $\mathcal{Z}$ answers with "ok", Sim stores $(\mathsf{pk}_i, \mathsf{vk}_i)$ as "registered".
8. Each time Sim is activated by $[\mathcal{G}]$ after an *honest* party P_i received its input, Sim generates $3N$ random strings $s'_{ij}, r'_{ij}, k'_{ij}$, computes $\sigma'_{ij} \leftarrow \text{Sig}(\mathsf{sgk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij})$ $(j = 1, \ldots, N)$ and $c_{ij} \leftarrow \text{Enc}(\mathsf{pk}_j, P_i, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$. Each time party $\mathcal{Z}$ activates the Enc-unit of P_i, Sim reports the respective tuple (P_i, c_{ij}) if pk_j is stored as "registered".

9. Once Sim has reported all (P_i, c_{ij}) $(j = 1, \ldots, N)$ as well as $(\texttt{registered}, sid', RM_i, \mathsf{pk}_i, \mathsf{vk}_i)$ for an honest party P_i, Sim marks P_i as online.
10. If a party P_i is corrupted after receiving input, Sim sends $(s'_{ii}, r'_{ii}, k'_{ii}, \sigma'_{ii}, \mathsf{vk}_i, \mathsf{sk}_i)$ to $\mathcal{Z}$.
11. If $\mathcal{Z}$ sends a value $(\mathsf{pk}_l, \mathsf{vk}_l)$ to $\mathcal{F}_{\mathsf{reg}}$ for a party P_l corrupted before receiving input, Sim stores $(\mathsf{pk}_l, \mathsf{vk}_l)$ as "registered".
12. Each time $\mathcal{Z}$ sends a message addressed to buffer of a party P_i, Sim stores that message as a message "received by P_i".
13. If $\mathcal{Z}$ activates an honest party P_j who is marked as online and has received at least $N - 1$ messages and all vk_l $(l = 1, \ldots, N)$ are stored as "registered", then Sim stores $\overline{\mathsf{vk}}_j = (\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ and reports $(\texttt{received}, P_i)$ to $\mathcal{Z}$. Upon receiving $(\texttt{confirmed}, P_i)$ from $\mathcal{Z}$, Sim marks P_j as $\mathsf{input\ given}$.
14. If $\mathcal{Z}$ sends a tuple consisting of a vector $\overline{\mathsf{vk}}_j$ and $(s'_{lj}, r'_{lj}, k'_{lj}, \sigma'_{lj})$ $(l = 1, \ldots, N)$ as the input to $\mathcal{F}_{\mathcal{G}}$ for a corrupted party P_j, then Sim stores that input (if an input has already been stored for P_j then Sim overwrites it) and, if not done yet, marks P_j as $\mathsf{input\ given}$.
15. Once all parties are marked as $\mathsf{input\ given}$, Sim does the following:
    (i) If not all $\overline{\mathsf{vk}}_i$ $(i = 1, \ldots, N)$ are equal, Sim sets $\texttt{verify} = \texttt{false}$.
    (ii) For each j such that party P_j is honest, Sim checks if the following two conditions hold:
    - P_j has received for each i such that party P_i is *not* corrupted before receiving input the tuple (P_i, c_{ij}), where c_{ij} is the respective ciphertext generated by Sim.
    - P_j has received for each l such that party P_l is corrupted before receiving input a set $\mathcal{M}_l$ fulfilling property $(*)$ (*Validity Check*, see Page 24).

    If at least one of these conditions does not hold, Sim sets $\texttt{verify} = \texttt{false}$.
    (iii) For each tuple consisting of a vector $\overline{\mathsf{vk}}_j$ and $(s'_{lj}, r'_{lj}, k'_{lj}, \sigma'_{lj})$ $(l = 1, \ldots, N)$ which was stored by Sim as the input to $\mathcal{F}_{\mathcal{G}}$ for a corrupted party P_j, Sim checks the following:
    - for each i such that party P_i was *not* corrupted before receiving input, Sim checks if $(s'_{ij}, r'_{ij}, k'_{ij}) = (s_{ij}, r_{ij}, k_{ij})$, where (s_{ij}, r_{ij}, k_{ij}) is the respective tuple generated by Sim. If this does not hold or $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 0$, Sim sets $\texttt{verify} = \texttt{false}$.
    - for each l such that party P_l was corrupted before receiving input, Sim sets $\texttt{verify} = \texttt{false}$ if $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_l, P_j, s'_{lj}, r'_{lj}, k'_{lj}, \sigma'_{lj}) = 0$.
16. Sim *extracts* the input, MAC key and random pad of each party P_l *corrupted before receiving input* by decrypting all ciphertexts coming from $\mathcal{Z}$, and by observing the inputs $\mathcal{Z}$ sends to $\mathcal{F}_{\mathcal{G}}$ for corrupted parties. Sim sends each extracted input to $[\mathcal{G}]$.
17. Once all parties are marked as $\mathsf{input\ given}$ and $\mathcal{Z}$ activates an honest party P_i, then Sim instructs $[\mathcal{G}]$ to
    (i) send the output to P_i's dummy OIM if $\texttt{verify} = \texttt{true}$.
    (ii) output $\perp$ to P_i's dummy OIM if $\texttt{verify} = \texttt{false}$.

18. Once all parties are marked as $\mathsf{input\ given}$ and $\mathcal{Z}$ requests the output of $\mathcal{F}_{\mathcal{G}}$ for a party P_i *corrupted after receiving input*, then
    (i) If $\mathtt{verify} = \mathtt{true}$, Sim generates a random string $\tilde{y}_i \leftarrow \{0,1\}^{p_i(n)}$ and sends $(\tilde{y}_i, \mathrm{Mac}(k_i, \tilde{y}_i))$ to $\mathcal{Z}$.
    (ii) If $\mathtt{verify} = \mathtt{false}$, Sim sends $\perp$ to $\mathcal{Z}$.
19. If $\mathcal{Z}$ sends a message (m', t') addressed to OIM of a party P_i *corrupted after receiving input*, then
    (i) If $\mathcal{Z}$ has not yet requested the output of $\mathcal{F}_{\mathcal{G}}$ for P_i yet, Sim instructs $[\mathcal{G}]$ to output $\perp$ to the dummy OIM of P_i.
    (ii) If $\mathcal{Z}$ has already requested the output of $\mathcal{F}_{\mathcal{G}}$ for P_i and Sim sent $(\tilde{y}_i, \mathrm{Mac}(k_i, \tilde{y}_i))$ (in Item 18) to $\mathcal{Z}$, then
       - If $m' \neq \tilde{y}_i$, Sim instructs $[\mathcal{G}]$ to output $\perp$ to the dummy OIM of P_i.
       - If $m' = \tilde{y}_i$ and $\mathrm{Vrfy}_{\mathrm{MAC}}(k_i, m', t') = 1$, then Sim instructs $[\mathcal{G}]$ to send the output to the dummy OIM of party P_i. Otherwise, Sim instructs $[\mathcal{G}]$ to output $\perp$ to the dummy OIM of P_i.
    (iii) If $\mathcal{Z}$ has already requested the output of $\mathcal{F}_{\mathcal{G}}$ for P_i and Sim sent $\perp$ (in Item 18) to $\mathcal{Z}$, then Sim instructs $[\mathcal{G}]$ to output $\perp$ to P_i's dummy OIM.
20. Once all parties are marked as $\mathsf{input\ given}$ and $\mathcal{Z}$ requests the output of $\mathcal{F}_{\mathcal{G}}$ for a party P_i *corrupted before receiving input*, then
    (i) If $\mathtt{verify} = \mathtt{true}$, Sim sends $(y_i + r_i, \mathrm{Mac}(k_i, y_i + r_i))$ to $\mathcal{Z}$, where y_i is the output of $[\mathcal{G}]$ for party P_i and k_i, r_i are the MAC key and random pad extracted in Item 16.
    (ii) If $\mathtt{verify} = \mathtt{false}$, Sim sends $\perp$ to $\mathcal{Z}$.
21. Sim lets $\mathcal{Z}$ determine the output of the dummy OIM of each party *corrupted before receiving input.*

We now state the theorem:

Theorem 3 (Up to $N-1$ Corrupted Parties, Non-Reactive Case). *Let $\mathcal{G}$ be a* non-reactive *standard*[21] *adaptively well-formed functionality. Assume* PKE *is NM-CPA-secure and* SIG *is EUF-naCMA-secure and length-normal, and* MAC *is EUF-1-CMA-secure. Then it holds that*

$$\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}} \geq_{\#\#} \mathtt{AG}([\mathcal{G}])$$

for up to $N-1$ parties under adversarial control.

Proof. By Proposition 1, it suffices to find a simulator for the dummy adversary.

In the following proof, we will consider a sequence of hybrids $\mathrm{H}_0, \ldots, \mathrm{H}_4$. Starting from the real protocol $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$, we will define ideal protocols that gradually reduce the simulator's abilities (i.e. restrict the set of parties for which he may learn/modify the inputs/outputs). The final hybrid H_4 will be the ideal protocol $\mathtt{AG}([\mathcal{G}])$ and the simulator as defined in Definition 7.

Let $\mathcal{Z}$ be an environment that instructs $\mathcal{D}$ to corrupt at most $N-1$ parties $P \in \{P_1, \ldots, P_N\}$. Let $\mathrm{out}_i(\mathcal{Z})$ be the output of $\mathcal{Z}$ in the hybrid H_i.

In the following, we will say *corrupted before input* and *corrupted after input* for the sake of brevity.

[21] Cf. Section 2.4 for a definition of standard ideal functionalities

Hybrid H_0. Let H_0 be the execution experiment between the environment $\mathcal{Z}$, the dummy adversary $\mathcal{D}$ and the real protocol $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$.

Hybrid H_1. Let H_1 be the execution experiment between the environment $\mathcal{Z}$, the ideal protocol $\mathtt{AG}(\mathcal{F}_1)$ and the ideal-model adversary Sim_1, where $\mathcal{F}_1$ and Sim_1 are defined as follows: Define $\mathcal{F}_1$ to be identical to $[\mathcal{G}]$ except for the following: $\mathcal{F}_1$ hands the adversary the inputs and outputs of *every* party (honest and corrupted) and allows him to determine the outputs of the dummy OIMs of *all corrupted* parties (i.e. all parties corrupted before *and* after input).

Define Sim_1 to be like the simulator in Definition 7 except for the following: In Item 8, Sim_1 reports the ciphertexts as they are generated in the real protocol (in particular, generates shares of the actual inputs). Also, if a party P_i is corrupted after having received input, Sim_1 reports the respective shares as they are generated in the real protocol in Item 10 along with a valid signature and $\mathsf{vk}_i, \mathsf{sk}_i$. In Item 18, if $\mathtt{verify} = \mathtt{true}$, Sim_1 reports $(y_i + r_i, \mathrm{Mac}(k_i, y_i + r_i))$ to $\mathcal{Z}$, where y_i is the output Sim_1 receives for the respective party from $\mathcal{F}_1$ and k_i, r_i are the MAC key and one-time pad generated in Items 6 and 8. If $\mathtt{verify} = \mathtt{false}$, Sim_1 reports $\perp$. In Item 19, if $\mathcal{Z}$ sends a message (m', t') addressed to OIM of a party P_i (corrupted after input), Sim_1 carries out the program of the OIM (using the MAC key and one-time pad generated in Items 6 and 8), computing a value $y' \in \{0,1\}^{p_i(n)} \cup \{\perp\}$, and then instructs $[\mathcal{G}]$ to output y' to P_i's dummy OIM.

Consider the following events:

Let $\mathbf{E}_{\text{fakemess}}$ be the event that there exists an *honest* party P_j that retrieves a tuple (P_i, c') in its buffer such that party P_i is *not* corrupted before input and $\mathrm{Dec}(\mathsf{sk}_j, c') = (P_i, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$ and $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 1$ but either $c' \neq c_{ij}$ or c_{ij} has not been generated yet (by the Enc-unit of party P_i).

Let $\mathbf{E}_{\text{fakeinp}}$ be the event that $\mathcal{Z}$ sends an input $(s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$ for a corrupted party P_j to $\mathcal{F}_{\mathcal{G}}$ such that $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 1$ but $(s'_{ij}, r'_{ij}, k'_{ij}) \neq (s_{ij}, r_{ij}, k_{ij})$, where (s_{ij}, r_{ij}, k_{ij}) was generated by a party P_i that was *not* corrupted before input.

Let $\mathbf{E} = \mathbf{E}_{\text{fakemess}} \cup \mathbf{E}_{\text{fakeinp}}$. It holds that

$$\mathsf{Pr}[\mathrm{out}_0(\mathcal{Z}) = 1 \wedge \neg\mathbf{E}] = \mathsf{Pr}[\mathrm{out}_1(\mathcal{Z}) = 1 \wedge \neg\mathbf{E}]$$

This is because if $\mathbf{E}_{\text{fakemess}}$ does not occur then a message in the buffer of a party P_j that is addressed as coming from a party P_i who was *not* corrupted before input decrypts to a valid message/signature pair if and only if it equals the ciphertext c_{ij} sent by P_i. Moreover, for each corrupted party P_i, since $\mathbf{E}_{\text{fakeinp}}$ does not occur, $\mathcal{Z}$ only sends inputs $(s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$ to $\mathcal{F}_{\mathcal{G}}$ such that either $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 0$ or $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 1$ and $(s'_{ij}, r'_{ij}, k'_{ij}) = (s_{ij}, r_{ij}, k_{ij})$ was generated by party P_i (who was not corrupted before input).

Therefore, it holds that

$$|\mathsf{Pr}[\mathrm{out}_0(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_1(\mathcal{Z}) = 1]| \leq \mathsf{Pr}[\mathbf{E}] \leq \mathsf{Pr}[\mathbf{E}_{\text{fakemess}}] + \mathsf{Pr}[\mathbf{E}_{\text{fakeinp}}]$$

Claim 1: $\mathsf{Pr}[\mathbf{E}_{\text{fakemess}}]$ *is negligible.*
Consider the following adversary $\mathcal{A}$ in the auxiliary experiment $\mathsf{Exp}^{\text{aux}}_{\text{PKE,SIG},\mathcal{A}(z)}(n)$: At the beginning, $\mathcal{A}$ randomly selects a tuple $(i,j) \in \{1,\dots,N\} \times \{1,\dots,N\}$ such that $i \neq j$. $\mathcal{A}$ then simulates hybrid H_0 using the public key pk from the experiment for pk_j in its internal simulation. When $\mathcal{Z}$ gives the party P_i its input x_i, $\mathcal{A}$ generates shares s_{il}, r_{il}, k_{il} of x_i, of a random pad r_i and of a MAC key k_i just like in H_0. $\mathcal{A}$ sends the tuples $(P_l, s_{il}, r_{il}, k_{il})$ for $l \neq j$ to the signing oracle $\mathcal{O}_{\text{Sig}(\mathsf{sgk},\cdot)}$, receiving signatures σ_{il}. After receiving the verification key vk from the experiment, $\mathcal{A}$ uses vk for vk_i in its internal simulation. Using pk, $\mathcal{A}$ encrypts all tuples $(P_i, s_{il}, r_{il}, k_{il}, \sigma_{il})$ $(l \notin \{i,j\})$ and sends them to the respective party in its internal simulation. Once the message (P_i, c_{ij}) is supposed to be sent in the internal simulation, $\mathcal{A}$ sends $(\mathtt{P_i}, \mathtt{P_j}, s_{ij}, r_{ij}, k_{ij})$ to the experiment, receiving c^*. $\mathcal{A}$ then uses (P_i, c^*) for (P_i, c_{ij}) in its simulation. When P_j is activated and is online and has received at least $N-1$ messages, $\mathcal{A}$ sends all ciphertexts addressed as coming from P_i such that $c \neq c^*$ to the decryption oracle $\mathcal{O}_{\text{Dec}(\mathsf{sk},\cdot)}$ (if c^* has not been generated yet, $\mathcal{A}$ sends all ciphertexts addressed as coming from P_i). For each message (P_l, m, σ) he receives from the oracle $\mathcal{O}_{\text{Dec}(\mathsf{sk},\cdot)}$, $\mathcal{A}$ checks if $\text{Vrfy}_{\text{SIG}}(vk, P_j, m, \sigma) = 1$. If this holds for a message (P_l, m', σ'), then $\mathcal{A}$ sends (P_j, m', σ') to the experiment. If during the simulation, P_i is corrupted before input or P_j is corrupted (before or after input) or if no message $\mathcal{A}$ receives from $\mathcal{O}_{\text{Dec}(\mathsf{sk},\cdot)}$ is valid, then $\mathcal{A}$ sends $\perp$ to the experiment.

By construction, it holds that if $\mathbf{E}_{\text{fakemess}}$ occurs and $\mathcal{A}$ has correctly guessed an index (i,j) for which $\mathbf{E}_{\text{fakemess}}$ occurs, then $\mathcal{A}$ sends a message c' to $\mathcal{O}_{\text{Dec}(\mathsf{sk},\cdot)}$ such that $c \neq c^*$ or c^* has not been generated yet and $\text{Dec}(\mathsf{sk}, c') = (P_i, m', \sigma')$ and $\text{Vrfy}_{\text{SIG}}(vk, P_j, m', \sigma') = 1$. Since $\mathcal{A}$ does not send a message of the form $(\mathtt{P_j}, m)$ to the signing oracle $\mathcal{O}_{\text{Sig}(\mathsf{sgk},\cdot)}$, it follows that $\mathsf{Exp}^{\text{aux}}_{\text{PKE,SIG},\mathcal{A}(z)}(n) = 1$. Furthermore, the probability that $\mathcal{A}$ correctly guesses an index (i,j) for which $\mathbf{E}_{\text{fakemess}}$ occurs is at least $1/(N \cdot (N-1))$. Hence,

$$\mathsf{Pr}[\mathsf{Exp}^{\text{aux}}_{\text{PKE,SIG},\mathcal{A}(z)}(n) = 1] \geq \mathsf{Pr}[\mathbf{E}_{\text{fakemess}}]/(N \cdot (N-1))$$

Therefore, since $\mathsf{Pr}[\mathsf{Exp}^{\text{aux}}_{\text{PKE,SIG},\mathcal{A}(z)}(n) = 1]$ is negligible by Lemma 1 and $N \cdot (N-1)$ is polynomial in n, it follows that $\mathsf{Pr}[\mathbf{E}_{\text{fakemess}}]$ is also negligible.

Claim 2: $\mathsf{Pr}[\mathbf{E}_{\text{fakeinp}}]$ *is negligible.*
Consider the following adversary $\mathcal{A}$ against the EUF-naCMA security of SIG: At the beginning, $\mathcal{A}$ randomly selects an index $i \in \{1,\dots,N\}$. $\mathcal{A}$ then simulates hybrid H_0. When $\mathcal{Z}$ gives the party P_i its input x_i, $\mathcal{A}$ generates shares s_{ij}, r_{ij}, k_{ij} of x_i, of a random pad r_i and of a MAC key k_i just like in H_0. $\mathcal{A}$ sends the tuples $(P_j, s_{ij}, r_{ij}, k_{ij})$ $(j \neq i)$ to the signing oracle $\mathcal{O}_{\text{Sig}(\mathsf{sgk},\cdot)}$, receiving signatures σ_{ij}. After receiving vk, $\mathcal{A}$ then uses vk for vk_i, encrypts all tuples $(P_i, s_{ij}, r_{ij}, k_{ij}, \sigma_{ij})$ $(j = 1,\dots,N)$ and sends them to the respective party in its internal simulation. Each time $\mathcal{Z}$ sends a tuple $(s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$ as input for a corrupted party P_j to $\mathcal{F}_{\mathcal{G}}$ such that $(s'_{ij}, r'_{ij}, k'_{ij}) \neq (s_{ij}, r_{ij}, k_{ij})$, $\mathcal{A}$ checks if $\text{Vrfy}_{\text{SIG}}(\mathsf{vk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}) = 1$. If this holds, $\mathcal{A}$ sends $(P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij})$ to the experiment. If during the simulation, P_i is corrupted before input or if no message $\mathcal{A}$ checks is valid, then $\mathcal{A}$ sends $\perp$ to the experiment.

By construction, it holds that if $\mathbf{E}_{\text{fakeinp}}$ occurs and $\mathcal{A}$ has correctly guessed an index i for which $\mathbf{E}_{\text{fakeinp}}$ occurs, then $\mathsf{Exp}^{\text{euf-nacma}}_{\text{SIG},\mathcal{A}(z)}(n) = 1$ because the tuple $(P_j, s'_{ij}, r'_{ij}, k'_{ij}, \sigma_{ij})$ is valid and $(P_j, s'_{ij}, r'_{ij}, k'_{ij}) \neq (P_j, s_{ij}, r_{ij}, k_{ij})$ has not been sent to the signing oracle $\mathcal{O}_{\text{Sig}(\mathsf{sgk},\cdot)}$. Furthermore, the probability that $\mathcal{A}$ correctly guesses an index i for which $\mathbf{E}_{\text{fakeinp}}$ occurs is at least $1/N$. Hence,

$$\Pr[\mathsf{Exp}^{\text{euf-nacma}}_{\text{SIG},\mathcal{A}(z),}(n) = 1] \geq \Pr[\mathbf{E}_{\text{fakeinp}}]/N$$

Therefore, since $\Pr[\mathsf{Exp}^{\text{euf-nacma}}_{\text{SIG},\mathcal{A}(z)}(n) = 1]$ is negligible because SIG is EUF-naCMA-secure by assumption and N is polynomial in n, it follows that $\Pr[\mathbf{E}_{\text{fakeinp}}]$ is also negligible.

Hence, there exist a negligible function negl_1 such that

$$|\Pr[\text{out}_0(\mathcal{Z}) = 1] - \Pr[\text{out}_1(\mathcal{Z}) = 1]| \leq \mathsf{negl}_1(n)$$

Hybrid H_2. Let H_2 be the execution experiment between the environment $\mathcal{Z}$, the ideal protocol $\texttt{AG}(\mathcal{F}_1)$ (again) and the ideal-model adversary Sim_2, where Sim_2 is defined as follows:

Define Sim_2 to be like Sim_1 except for the following: In Item 8, each time Sim_2 is activated by $\mathcal{F}_1$ after an *honest* party P_i received its input, Sim_2 generates N random strings k'_{ij} and computes $\sigma'_{ij} \leftarrow \text{Sig}(\mathsf{sgk}_i, P_j, s_{ij}, r_{ij}, k'_{ij})$ $(j = 1, \ldots, N)$, where the s_{ij} and r_{ij} $(j = 1, \ldots, N)$ are still the shares of the input x_i and a random pad r_i, respectively. Sim_2 then iteratively reports $(P_i, \text{Enc}(\mathsf{pk}_j, P_i, s_{ij}, r_{ij}, k'_{ij}, \sigma'_{ij}))$ $(j \in \{1, \ldots, N\} \setminus \{i\})$ to $\mathcal{Z}$. If a party P_i is corrupted after having received input, Sim_2 sends $(s_{ii}, r_{ii}, k'_{ii}, \sigma'_{ii}, \mathsf{vk}_i, \mathsf{sk}_i)$ to $\mathcal{Z}$ in Item 10. (Note that in Item 18 Sim_2 still uses the MAC key $k_i \leftarrow \text{Gen}_{\text{MAC}}(1^n)$ generated in Item 6 for the output of $\mathcal{F}_{\mathcal{G}}$ to a party P_i corrupted after input (if that output is $\neq \perp$)).

Let $\text{H}_{2,0}, \ldots, \text{H}_{2,N}$ be the execution experiment between the environment $\mathcal{Z}$, the ideal protocol $\texttt{AG}(\mathcal{F}_1)$ and the ideal-model adversary $\mathsf{Sim}_{2,0}, \ldots, \mathsf{Sim}_{2,N}$, respectively, where $\mathsf{Sim}_{2,i}$ is defined as follows:

Define the simulators $\mathsf{Sim}_{2,i}$ to be like Sim_1 except for the following: In Item 8, each time $\mathsf{Sim}_{2,i}$ is activated by $\mathcal{F}_1$ after an *honest* party $P_l \in \{P_1, \ldots, P_i\}$ received its input, $\mathsf{Sim}_{2,i}$ generates N random strings k'_{lj}, computes $\sigma'_{lj} \leftarrow \text{Sig}(\mathsf{sgk}_l, P_j, s_{lj}, r_{lj}, k'_{lj})$ $(j = 1, \ldots, N)$, and iteratively reports $(P_l, \text{Enc}(\mathsf{pk}_j, P_l, s_{lj}, r_{lj}, k'_{lj}, \sigma'_{lj}))$ $(j \in \{1, \ldots, N\} \setminus \{l\})$ to $\mathcal{Z}$. If a party $P_l \in \{P_1, \ldots, P_i\}$ is corrupted after having received input, $\mathsf{Sim}_{2,i}$ sends $(s_{ll}, r_{ll}, k'_{ll}, \sigma'_{ll}, \mathsf{vk}_l, \mathsf{sk}_l)$ to $\mathcal{Z}$ in Item 10.

It holds that

$$\Pr[\text{out}_{2,0}(\mathcal{Z}) = 1] = \Pr[\text{out}_1(\mathcal{Z}) = 1]$$

and

$$\Pr[\text{out}_{2,N}(\mathcal{Z}) = 1] = \Pr[\text{out}_2(\mathcal{Z}) = 1]$$

Assume that there exists a non-negligible function ϵ such that $|\Pr[\text{out}_1(\mathcal{Z}) = 1] = \Pr[\text{out}_2(\mathcal{Z}) = 1]| > \epsilon$. Then there exists an $i^* \in \{1, \ldots, N\}$ such that

$$|\Pr[\text{out}_{2,i^*-1}(\mathcal{Z}) = 1] - \Pr[\text{out}_{2,i^*}(\mathcal{Z}) = 1]| > \epsilon/N$$

Moreover, if party P_{i^*} is not corrupted after input, i.e. if it is corrupted before input or remains honest throughout the execution, then the views of $\mathcal{Z}$ in H_{2,i^*-1} and H_{2,i^*} are identically distributed. Therefore,

$$\begin{aligned}\epsilon/N <& |\mathsf{Pr}[\mathrm{out}_{2,i^*-1}(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_{2,i^*}(\mathcal{Z}) = 1]| \\ =& |\mathsf{Pr}[\mathrm{out}_{2,i^*-1}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input}] \\ & - \mathsf{Pr}[\mathrm{out}_{2,i^*}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input}]|\end{aligned}$$

Consider the following adversary $\mathcal{A}$ against the IND-pCCA security of PKE: At the beginning, $\mathcal{A}$ randomly selects an index $j \in \{1, \ldots, N\} \setminus \{i^*\}$. $\mathcal{A}$ then simulates the experiment H_{2,i^*-1}. When $\mathcal{Z}$ gives the party P_{i^*} its input x_{i^*}, $\mathcal{A}$ generates shares $s_{i^*l}, r_{i^*l}, k_{i^*l}$ of the input x_{i^*}, of a random pad r_{i^*} and of a MAC key k_{i^*} just like in H_{2,i^*-1}. $\mathcal{A}$ additionally generates random strings k'_{i^*l} ($l \in \{1, \ldots, N\}$). $\mathcal{A}$ then generates signatures σ_{i^*j}, σ'_{i^*j} for $(P_j, s_{i^*j}, r_{i^*j}, k_{i^*j})$ and $(P_j, s_{i^*j}, r_{i^*j}, k'_{i^*j})$, respectively, and sends $(P_{i^*}, s_{i^*j}, r_{i^*j}, k_{i^*j}, \sigma_{i^*j})$, $(P_{i^*}, s_{i^*j}, r_{i^*j}, k'_{i^*j}, \sigma'_{i^*j})$ to the experiment, receiving a ciphertext c^*. Note that $\mathcal{A}$'s challenge messages are allowed, i.e. have the same length, because SIG is length-normal. $\mathcal{A}$ then continues simulating the experiment H_{2,i^*-1} using c^* as c_{i^*j} and his decryption oracle to decrypt the ciphertexts in the buffer of P_j that are addressed as coming from the parties corrupted before input but do not equal c^* (the ones that are equal to c^* are ignored. Note that a tuple (P_l, c^*) sent by a party P_l corrupted before input is always invalid since $P_l \neq P_{i^*}$). Note that in $\mathcal{A}$'s internal simulation, party P_{i^*} receives the correct value from $\mathcal{F}_{\mathcal{G}}$ (i.e. $(y_{i^*} + r_{i^*}, \mathrm{Mac}(k_{i^*}, y_{i^*} + r_{i^*}))$ or $\perp$). At the end of the experiment, $\mathcal{A}$ outputs what $\mathcal{Z}$ outputs. If during the simulation, $\mathcal{Z}$ corrupts P_j (before or after input) or if P_{i^*} is *not* corrupted after input, $\mathcal{A}$ sends $\perp$ to the experiment.

Let $\mathrm{output}_b(\mathcal{A})$ denote the output of $\mathcal{A}$ in the IND-pCCA experiment when the challenge bit b is chosen. By construction, assuming party P_{i^*} is corrupted after input, if $\mathcal{A}$ guessed an index j such that party P_j remains honest then it holds that if the challenge bit is 0 the view of $\mathcal{Z}$ in $\mathcal{A}$'s internal simulation is distributed as in the experiment H_{2,i^*-1} and if the challenge bit is 1 the view of $\mathcal{Z}$ in $\mathcal{A}$'s internal simulation is distributed as in the experiment H_{2,i^*}. Moreover, assuming party P_{i^*} is corrupted after input, the probability that $\mathcal{A}$ guesses an index j such that party P_j remains honest is at least $1/(N-1)$. Hence,

$$\begin{aligned}& |\mathsf{Pr}[\mathrm{output}_0(\mathcal{A}) = 1] - \mathsf{Pr}[\mathrm{output}_1(\mathcal{A}) = 1]| \\ =& |\mathsf{Pr}[\mathrm{out}_{2,i^*-1}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input} \wedge \textbf{Guess correct}] \\ & - \mathsf{Pr}[\mathrm{out}_{2,i^*}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input} \wedge \textbf{Guess correct}]| \\ >& \epsilon/(N \cdot (N-1))\end{aligned}$$

This contradicts the IND-pCCA security of PKE.

Hence, there exist a negligible function negl_2 such that

$$|\mathsf{Pr}[\mathrm{out}_1(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_2(\mathcal{Z}) = 1]| \leq \mathsf{negl}_2(n)$$

Hybrid H_3. Let H_3 be the execution experiment between the environment $\mathcal{Z}$, the ideal protocol $\mathtt{AG}(\mathcal{F}_2)$ and the ideal-model adversary Sim_3, where $\mathcal{F}_2$ and Sim_3 are defined as follows:

Let $\mathcal{F}_2$ be identical to $\mathcal{F}_1$ except that now the adversary is allowed to determine the outputs only of the dummy OIMs of the parties *corrupted before input.*

Define Sim_3 to be like Sim_2 except that Item 19 is identical to the same step of the simulator in Definition 7.

Let $\mathbf{E}_{\text{fakeoutp}}$ be the event that $\mathcal{Z}$ sends a message (m', t') to OIM of a party P_i corrupted after input such that $\mathrm{Vrfy}_{\mathrm{MAC}}(k_i, m', t') = 1$ but either P_i has received $\perp$ from $\mathcal{F}_{\mathcal{G}}$ or a tuple (m, t) such that $m' \neq m$ or P_i has not received an output from $\mathcal{F}_{\mathcal{G}}$ yet.

It is easy to see that the following holds:

$$\mathsf{Pr}[\mathrm{out}_2(\mathcal{Z}) = 1 \wedge \neg\mathbf{E}_{\text{fakeoutp}}] = \mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1 \wedge \neg\mathbf{E}_{\text{fakeoutp}}]$$

Therefore, it holds that

$$|\mathsf{Pr}[\mathrm{out}_2(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1]| \leq \mathsf{Pr}[\mathbf{E}_{\text{fakeoutp}}]$$

Claim 3: $\mathsf{Pr}[\mathbf{E}_{\text{fakeoutp}}]$ *is negligible.*
Consider the adversary $\mathcal{A}$ against the EUF-1-CMA-security of MAC. At the beginning, $\mathcal{A}$ randomly selects an index $i \in \{1, \ldots, N\}$. $\mathcal{A}$ then simulates the hybrid H_2. Once $\mathcal{Z}$ expects the output from $\mathcal{F}_{\mathcal{G}}$ for party P_i (if P_i is corrupted after input), $\mathcal{A}$ computes the (padded) result m for this party. If $m = \perp$, $\mathcal{A}$ sends $\perp$ to $\mathcal{Z}$. Otherwise, $\mathcal{A}$ sends m to the MAC oracle $\mathcal{O}_{\mathrm{Mac}(k,\cdot)}$, receiving a tag t. $\mathcal{A}$ then sends (m, t) to $\mathcal{Z}$. If $\mathcal{Z}$ sends a tuple (m', t') to OIM of P_i such that $m' \neq m$, then $\mathcal{A}$ sends (m', t') to the experiment. If during the simulation, P_i is *not* corrupted after input or if $\mathcal{Z}$ sends $\perp$ or a tuple (m', t') such that $m' = m$ to OIM of P_i, then $\mathcal{A}$ sends $\perp$ to the experiment.

By construction, it holds that if $\mathbf{E}_{\text{fakeoutp}}$ occurs and $\mathcal{A}$ correctly guessed an index for which $\mathbf{E}_{\text{fakeoutp}}$ occurs, then $\mathsf{Exp}^{\mathsf{euf\text{-}1\text{-}cma}}_{\mathrm{MAC},\mathcal{A}(z)}(n) = 1$ because (m', t') is valid and $m' \neq m$ has not been sent to $\mathcal{O}_{\mathrm{Mac}(k,\cdot)}$. Moreover, the probability that $\mathcal{A}$ correctly guesses an index for which $\mathbf{E}_{\text{fakeoutp}}$ occurs is at least $1/N$. Hence,

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{euf\text{-}1\text{-}cma}}_{\mathrm{MAC},\mathcal{A}(z)}(n) = 1] \geq \mathsf{Pr}[\mathbf{E}_{\text{fakeoutp}}]/N$$

Therefore, since $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{euf\text{-}1\text{-}cma}}_{\mathrm{MAC},\mathcal{A}(z)}(n) = 1]$ is negligible because MAC is EUF-1-CMA-secure by assumption and N is polynomial in n, it follows that $\mathsf{Pr}[\mathbf{E}_{\text{fakeoutp}}]$ is also negligible.

Hence, there exist a negligible function negl_3 such that

$$|\mathsf{Pr}[\mathrm{out}_2(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1]| \leq \mathsf{negl}_3(n)$$

Hybrid H_4. Let H_4 be the execution experiment between $\mathcal{Z}$, the ideal protocol $\mathtt{AG}(\mathcal{F}_3)$ and the ideal-model adversary Sim_4, where $\mathcal{F}_3$ and Sim_4 are defined as follows: Let $\mathcal{F}_3$ be identical to $\mathcal{F}_2$ except that the adversary is *not* given the inputs and outputs of honest parties anymore. The adversary is only given the inputs and outputs of parties corrupted after input when all parties are corrupted.

Let $\mathrm{H}_{4,0}, \dots, \mathrm{H}_{4,N}$ be the execution experiment between the environment $\mathcal{Z}$, the ideal protocol $\mathtt{AG}(\mathcal{F}_{3,0}), \dots, \mathtt{AG}(\mathcal{F}_{3,N})$ and the adversary $\mathsf{Sim}_{4,0}, \dots, \mathsf{Sim}_{4,N}$, respectively, where $\mathcal{F}_{3,i}$ and $\mathsf{Sim}_{4,i}$ are defined as follows:

Define $\mathcal{F}_{3,i}$ be identical to $\mathcal{F}_2$ except now the adversary is not given the inputs and outputs of the parties $P_l \in \{1, \dots, i\}$ if they are honest or corrupted after input unless all parties are corrupted.

Define the simulators $\mathsf{Sim}_{4,i}$ to be like Sim_3 except for the following: In Item 8, each time $\mathsf{Sim}_{4,i}$ is activated by $\mathcal{F}_{3,i}$ after an *honest* party $P_l \in \{P_1, \dots, P_i\}$ received its input, $\mathsf{Sim}_{4,i}$ generates $3N$ random strings $s'_{lj}, r'_{lj}, k'_{lj}$, computes $\sigma'_{lj} \leftarrow \mathrm{Sig}(\mathsf{sgk}_l, j, s'_{lj}, r'_{lj}, k'_{lj})$ $(j = 1, \dots, N)$, and iteratively reports $(P_l, \mathrm{Enc}(\mathsf{pk}_j, P_l, s'_{lj}, r'_{lj}, k'_{lj}, \sigma'_{lj}))$ $(j \in \{1, \dots, N\} \setminus \{l\})$ to $\mathcal{Z}$. If a party $P_l \in \{P_1, \dots, P_i\}$ is corrupted after having received input, $\mathsf{Sim}_{4,i}$ sends $(s'_{ll}, r'_{ll}, k'_{ll}, \sigma'_{ll}, \mathsf{vk}_l, \mathsf{sk}_l)$ to $\mathcal{Z}$ in Item 10. In Item 18, if $\mathtt{verify} = \mathtt{true}$, then for every corrupted party $P_l \in \{P_1, \dots, P_i\}$, Sim_4 generates a random string $\tilde{y}_l \leftarrow \{0,1\}^{p_l(n)}$ and sends $(\tilde{y}_l, \mathrm{Mac}(k_l, \tilde{y}_l))$ to $\mathcal{Z}$ as the output from $\mathcal{F}_{\mathcal{G}}$, where $k_l \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$ is the MAC key generated in Item 6. If $\mathtt{verify} = \mathtt{false}$, then for every corrupted party, $\mathsf{Sim}_{4,i}$ sends $\perp$ to $\mathcal{Z}$ as the output from $\mathcal{F}_{\mathcal{G}}$.

It holds that

$$\mathsf{Pr}[\mathrm{out}_{4,0}(\mathcal{Z}) = 1] = \mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1]$$

and

$$\mathsf{Pr}[\mathrm{out}_{4,N}(\mathcal{Z}) = 1] = \mathsf{Pr}[\mathrm{out}_4(\mathcal{Z}) = 1]$$

Assume that there exists a non-negligible function ϵ such that $|\mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1] = \mathsf{Pr}[\mathrm{out}_4(\mathcal{Z}) = 1]| > \epsilon$. Then there exists an $i^* \in \{1, \dots, N\}$ such that

$$|\mathsf{Pr}[\mathrm{out}_{4,i^*-1}(\mathcal{Z}) = 1] = \mathsf{Pr}[\mathrm{out}_{4,i^*}(\mathcal{Z}) = 1]| > \epsilon/N$$

Moreover, if party P_{i^*} is not corrupted after input, i.e. if it is corrupted before input or remains honest throughout the execution, then the views of $\mathcal{Z}$ in H_{4,i^*-1} and H_{4,i^*} are identically distributed. Therefore,

$$\begin{aligned}\epsilon/N &< |\mathsf{Pr}[\mathrm{out}_{4,i^*-1}(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_{4,i^*}(\mathcal{Z}) = 1]| \\ &= |\mathsf{Pr}[\mathrm{out}_{4,i^*-1}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input}] \\ &\quad - \mathsf{Pr}[\mathrm{out}_{4,i^*}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input}]|\end{aligned}$$

Consider the following adversary $\mathcal{A}$ against the IND-pCCA security of PKE: At the beginning, $\mathcal{A}$ randomly selects an index $j \in \{1, \dots, N\} \setminus \{i^*\}$. $\mathcal{A}$ then simulates the experiment H_{4,i^*-1}. When $\mathcal{Z}$ gives the party P_{i^*} its input x_{i^*}, $\mathcal{A}$ generates shares s_{i^*l} and r_{i^*l} of x_{i^*} and of a random pad r_{i^*} and generates random strings k'_{i^*l} $(l \in \{1, \dots, N\})$ just like in H_{4,i^*-1}. $\mathcal{A}$ additionally generates random strings s'_{i^*j} and r'_{i^*j} $(l \in \{1, \dots, N\})$. $\mathcal{A}$ then generates signatures $\sigma_{i^*j}, \sigma'_{i^*j}$ for $(P_j, s_{i^*j}, r_{i^*j}, k'_{i^*j})$ and $(P_j, s'_{i^*j}, r'_{i^*j}, k'_{i^*j})$, respectively, and sends $(P_{i^*}, s_{i^*j}, r_{i^*j}, k'_{i^*j}, \sigma_{i^*j})$, $(P_{i^*}, s'_{i^*j}, r'_{i^*j}, k'_{i^*j}, \sigma'_{i^*j})$ to the experiment, receiving a ciphertext c^*. Note that $\mathcal{A}$'s challenge messages are allowed because SIG is length-normal. $\mathcal{A}$ then continues simulating the experiment H_{4,i^*-1} using c^* as c_{i^*j} and his decryption oracle to decrypt the ciphertexts in the buffer of P_j that are addressed as coming from the parties corrupted before input but do not equal c^* (the ones that are equal to c^* are ignored. Note that a tuple (P_l, c^*) sent by a party P_l corrupted before input is always invalid since $P_l \neq P_{i^*}$). Note that in $\mathcal{A}$'s internal simulation, party P_{i^*} receives the correct value from $\mathcal{F}_{\mathcal{G}}$ (i.e. $(y_{i^*} + r_{i^*}, \mathrm{Mac}(k_{i^*}, y_{i^*} + r_{i^*})$ or $\perp$). At the end of the experiment, $\mathcal{A}$ outputs what $\mathcal{Z}$ outputs. If during the simulation, $\mathcal{Z}$ corrupts P_j (before or after input) or if party P_{i^*} is *not* corrupted after input, then $\mathcal{A}$ sends $\perp$ to the experiment.

Let $\mathrm{output}_b(\mathcal{A})$ denote the output of $\mathcal{A}$ in the IND-pCCA experiment when the challenge bit b is chosen. By construction, assuming party P_{i^*} is corrupted after input, if $\mathcal{A}$ guessed an index j such that party P_j remains honest then it holds that if the challenge bit is 0 the view of $\mathcal{Z}$ in $\mathcal{A}$'s internal simulation is distributed as in the experiment H_{4,i^*-1} and if the challenge bit is 1 the view of $\mathcal{Z}$ in $\mathcal{A}$'s internal simulation is distributed as in the experiment H_{4,i^*}. Moreover, assuming party P_{i^*} is corrupted after input, the probability that $\mathcal{A}$ guesses an index j such that party P_j remains honest is at least $1/(N-1)$. Hence,

$$\begin{aligned}
&|\mathsf{Pr}[\mathrm{output}_0(\mathcal{A}) = 1] - \mathsf{Pr}[\mathrm{output}_1(\mathcal{A}) = 1]| \\
=&|\mathsf{Pr}[\mathrm{out}_{4,i^*-1}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input} \wedge \textbf{Guess correct}] \\
&- \mathsf{Pr}[\mathrm{out}_{4,i^*}(\mathcal{Z}) = 1 \wedge \textbf{party } P_{i^*} \textbf{ corrupted after input} \wedge \textbf{Guess correct}]| \\
>& \epsilon/(N \cdot (N-1))
\end{aligned}$$

This contradicts the IND-pCCA security of PKE.

Hence, there exists a negligible function negl_3 such that

$$|\mathsf{Pr}[\mathrm{out}_3(\mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{out}_4(\mathcal{Z}) = 1]| \leq \mathsf{negl}_3(n)$$

Since H_4 is identical an execution between $\mathcal{Z}$, the ideal protocol $\mathtt{AG}([\mathcal{G}])$ and the simulator as defined in Definition 7, it follows that there exists a negligible function negl such that

$$|\mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}, \mathcal{D}, \mathcal{Z}) = 1] - \mathsf{Pr}[\mathrm{Exec}_{\mathrm{FortUC}}(\mathtt{AG}([\mathcal{G}]), \mathsf{Sim}, \mathcal{Z}) = 1]| \leq \mathsf{negl}(n)$$

The statement follows. $\square$

Remark 4. Using Theorems 1 and 2, we can replace $\overline{\mathtt{SC}(\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}})}$ in $\Pi_{\mathcal{G}}^{\mathsf{N-1,nreac}}$ with an adaptively UC-secure protocol, e.g. [8]. Note that this inevitably requires an additional trusted setup assumption (e.g. a common reference string) because our unhackable sub-parties (and $\mathcal{F}_{\mathsf{reg}}$) are not UC-complete.

Remark 5. Note that one can also let a party check each message it receives (in its buffer) right away once it is online without having to wait for at least $N-1$ messages in the buffer. The protocol remains secure if one assumes the stronger assumption that PKE is IND-CCA-secure (cf. Appendix B.6).

Remark 6. Note that if the parties P_i disconnect all their air-gap switches again after receiving output from $\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$ then $\mathcal{A}$ cannot obtain all shares anymore.

4.1 Up to N Parties Under Adversarial Control

One can augment Construction 2 in order to obtain a protocol $\Pi_{\mathcal{G}}^{\mathsf{N},\mathsf{nreac}}$ that is also secure if the adversary corrupts *all* parties at the expense of one additional unhackable sub-party called *decryption unit* (Dec-unit). The main idea in the new construction is that parties do not decrypt ciphertexts themselves but send them to Dec-unit. Each Dec-unit receives the secret key from its main party during the sharing phase. In the compute phase, each Dec-unit accepts a single vector of ciphertexts from its main party. (cf. Fig. 4). Since the Dec-units are

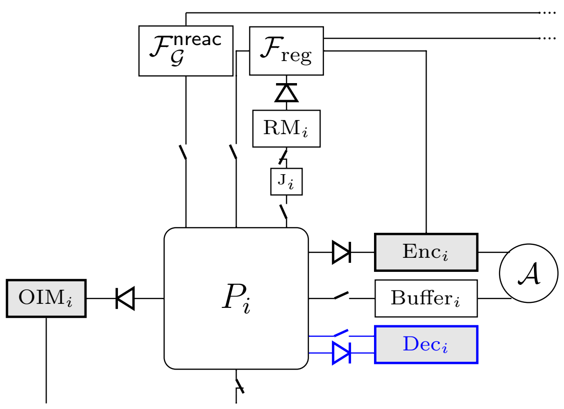

$\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$
$\mathcal{F}_{\mathrm{reg}}$
RM$_i$
J$_i$
P_i
OIM$_i$
Enc$_i$
Buffer$_i$
Dec$_i$
$\mathcal{A}$

Fig. 4: Architecture of $\Pi_{\mathcal{G}}^{\mathsf{N},\mathsf{nreac}}$. Each party P_i $(i = 1, \ldots, N)$ has 3 hackable sub-parties, called *buffer*, *registration module* (RM) and *join* (J), and 3 unhackable sub-parties, called Enc(-unit), Dec(-unit) and OIM. Buffer and Enc-unit are connected to the adversary via standard channels. All air-gap switches, except for P's air-gap switch to the environment and the RM's air-gap switch to J, are initially *disconnected.*

unhackable and do not leak the secret keys, the simulator can report plaintext tuples to $\mathcal{Z}$ in such a way that the shares they contain are consistent with the parties' inputs and outputs even if all parties are corrupted.

Theorem 4 (Up to N Corrupted Parties, Non-Reactive Case). *Let $\mathcal{G}$ be a* non-reactive *standard adaptively well-formed functionality. Assume* PKE, SIG, MAC *are as in Theorem 3. Then it holds that*

$$\Pi_{\mathcal{G}}^{\mathsf{N},\mathsf{nreac}} \geq_{\#\#} \mathtt{AG}([\mathcal{G}])$$

for up to N parties under adversarial control.

The simulator Sim' for the case of up to N parties under adversarial control is identical to the simulator for up to $N-1$ in Definition 7, except for the following: Once *all* parties have been corrupted, Sim', who learns the inputs and outputs of all parties from $[\mathcal{G}]$ in this case, reports plaintext tuples to $\mathcal{Z}$ in such a way that the shares they contain are consistent with the parties' inputs and outputs. Note that $\mathcal{Z}$ cannot check if the tuples it receives from Sim' were encrypted before since it does not have the secret keys.

More specifically, each time Sim' is activated by $[\mathcal{G}]$ after an *honest* party P_i received its input, the simulator Sim' generates $3N$ random strings $s'_{ij}, r'_{ij}, k'_{ij}$, computes $\sigma'_{ij} \leftarrow \mathrm{Sig}(\mathsf{sgk}_i, P_j, s'_{ij}, r'_{ij}, k'_{ij})$ $(j = 1, \ldots, N)$, and reports the ciphertext $(P_i, \mathrm{Enc}(\mathsf{pk}_j, P_i, s'_{ij}, r'_{ij}, k'_{ij}, \sigma'_{ij}))$ $(j \in \{1, \ldots, N\} \setminus \{i\})$ to $\mathcal{Z}$. Furthermore, for each $i = 1, \ldots, N$, Sim' generates random strings $\tilde{y}_i \leftarrow \{0,1\}^n$. Once the last party, denoted by P_{l^*}, is corrupted, Sim' computes for each i the shares $\tilde{s}_{il^*} = x_i + \sum_{j \in \{1,\ldots,N\} \setminus \{l^*\}} s'_{ij}$, and $\tilde{k}_{il^*} = k_i + \sum_{j \in \{1,\ldots,N\} \setminus \{l^*\}} k'_{ij}$ and $\tilde{r}_{il^*} = \tilde{y}_i + y_i + \sum_{j \in \{1,\ldots,N\} \setminus \{l^*\}} r'_{ij}$. Sim' then generates $\tilde{\sigma}_{il^*} \leftarrow \mathrm{Sig}(\mathsf{sgk}_i, P_{l^*}, \tilde{s}_{il^*}, \tilde{r}_{il^*}, \tilde{k}_{il^*})$. When $\mathcal{Z}$ sends a vector of ciphertexts to the Dec-unit of party P_{l^*}, then Sim' checks for each c' contained in that vector if $c' = c^i_{l^*}$ for some i. For each c' for which this holds, Sim' returns the corresponding $(P_i, \tilde{s}_{il^*}, \tilde{r}_{il^*}, \tilde{k}_{il^*}, \tilde{\sigma}_{il^*})$. For each c' for which this does not hold, Sim' returns $\mathrm{Dec}(\mathsf{sk}^*_l, c')$.

5 Construction for Reactive Functionalities

In this section, we present a construction for realizing the fortified functionality of every *reactive* (standard adaptively well-formed) ideal functionality. The new construction is a direct generalization of Constructions 1 and 2.

For reactive functionalities, a new problem arises because a protocol party is online after the first round. The input(s) for the next round(s) can therefore not just be given to a party since it may be corrupted. We therefore need to find a way to insert the input(s) for the rounds $u \geq 2$ into the protocol without allowing a party to learn or modify them.

To this end, we introduce an additional unhackable sub-party called *input interface module* (IIM) that acts as the counterpart of the OIM for inputs. Let $\mathrm{R} \in \mathbb{N}$ be the number of rounds. In the sharing phase, each party P_i generates $2\mathrm{R}$ random pads $r_i^1, \ldots, r_i^{\mathrm{R}}, t_i^1, \ldots, t_i^{\mathrm{R}}$ and shares them as before. Also, each party P_i pads its (first) input $\tilde{x}_i^1 = x_i^1 + t_i^1$ and computes a MAC tag of it. Then, each party P_i sends the R random pads $r_i^1, \ldots, r_i^{\mathrm{R}}$ as well as the MAC key k_i to the OIM and the other R random pads $t_i^1, \ldots, t_i^{\mathrm{R}}$ and the MAC key k_i to the IIM. As before, each random pad is shared with the other parties along with signatures on these shares, the PID of the designated receiver as well as the *number of the round* in which this share is to be used. Note that the latter prevents an adversary from re-using shares from earlier rounds.

In each compute phase, the parties will use their shares and padded inputs to compute the desired padded output values for that round and MAC tags of these padded output values *along with a prefix indicating this being an output and the round number.* Verification and reconstruction of the output values is then done as before using the OIM. Note that since the prefix contains the round number, the OIM is able to reject results from *earlier* computation phases.

As before, each input to the compute phase has to be verified before the desired padded output values are computed. Now, however, not only the signatures of the shares are verified but also the MAC tags of the padded inputs. In order to obtain the MAC tags for the padded inputs for the rounds $u \geq 2$, the respective input has to be inserted into the protocol via the IIM. The IIM then applies a one-time pad on each input it receives and computes a MAC tag of the padded input *along with a prefix indicating this being an input and the round number.* It then sends the computed tuple to the party. This way, a party will be able to continue the computation without learning the inputs for the rounds $u \geq 2$. Note that due to the prefix containing the round number, the adversary cannot use padded inputs of *earlier* rounds. Also note that since the prefix indicates inputs/outputs, an adversary cannot send a padded *input* to the OIM.

As before, we will take a modular approach and define an ideal functionality $\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$ that implements the verification of the input values in the compute phase as well as the multi-party computation on the shares and padded inputs.

We first define the functionality $\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$.

Construction 3
Let $\mathcal{G}$ be a reactive standard adaptively well-formed ideal functionality with R *rounds.* $\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$ *proceeds as follows, running with parties $P_1, \dots, P_N$ and an adversary $\mathcal{A}$ and parametrized with a digital signature* SIG *and a message authentication code* MAC*.*

1. *Upon receiving $(\texttt{corrupt}, P_i)$, behave like a standard corruption ideal functionality. In addition, forward this message to $\mathcal{G}$.*
2. *Initialize* $\mathrm{R}+1$ *Boolean variables* $\texttt{verify}^0$, $\texttt{verify}^1, \dots, \texttt{verify}^{\mathrm{R}} = \texttt{true}$ *and a counter $u = 1$.*
3. *Upon receiving input from party P_i, store it and send $(\texttt{received}, P_i)$ to $\mathcal{A}$. Upon receiving $(\texttt{confirmed}, P_i)$ from $\mathcal{A}$, mark P_i as* input given*.*
4. *Upon receiving from $\mathcal{A}$ a (modified) input for a party P_l marked as* corrupted*, store that input (if an input has already been stored for P_l then overwrite it) and, if not done yet, mark P_l as* input given*.*

Consistency Check

5. *Once each party has been marked as* input given*, proceed with Item 6 if this is round $u = 1$, else proceed with Item 7.*
6. *Check if every party P_i has sent an input of the form $\overline{\mathsf{vk}}_i = (\mathsf{vk}_1^{(i)}, \dots, \mathsf{vk}_N^{(i)})$, $(t_{ji}, r_{ji}, \sigma_{ji}, k_{ji}, \sigma'_{ji})$ $(j = 1, \dots, N)$.*
   - *i) If no, set $\texttt{verify}^0 = \texttt{false}$.*
   - *ii) If yes, check if $\overline{\mathsf{vk}}_1 = \dots = \overline{\mathsf{vk}}_N$.*
     - *(A) If this does not hold, set $\texttt{verify} = \texttt{false}$.*
     - *(B) Else, set $(\mathsf{vk}_1, \dots, \mathsf{vk}_n) = (\mathsf{vk}_1^{(1)}, \dots, \mathsf{vk}_N^{(1)})$. For all $i = 1, \dots, N$, check if $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, k_{ji}, \sigma'_{ji}) = 1$ for all $j = 1, \dots, N$.*
       - *(a) If this does not hold, set $\texttt{verify} = \texttt{false}$.*
       - *(b) Else, for each $i = 1, \dots, N$, compute and store $k_i = k_{i1} + k_{i2} + \dots + k_{iN}$ and continue with Item 8.*

7. *If* $\texttt{verify}^0 = \texttt{false}$*, do nothing. Else, check if every party* P_i *has sent an input of the form* $(t^u_{ji}, r^u_{ji}, \sigma^u_{ji})$ $(j = 1,\ldots,N)$, $(\tilde{x}^u_i, \tau^u_i)$*. If no, set* $\texttt{verify}^\texttt{u} = \texttt{false}$*. Else, continue with Item 8.*
8. *For all* $i = 1,\ldots,N$*, check if* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, u, P_i, t^u_{ji}, r^u_{ji}, \sigma^u_{ji}) = 1$ *for all* $j = 1,\ldots,N$ *and if* $\mathrm{Vrfy}_{\mathrm{MAC}}(k_i, \texttt{Inp Round u}, \tilde{x}^u_i, \tau^u_i) = 1$*.*
   (a) If this does not hold for all i, j*, set* $\texttt{verify}^\texttt{u} = \texttt{false}$*.*
   (b) Else, proceed with Item 9.

Reconstruction and Computation

9. *For each* $i = 1,\ldots,N$*, compute* $r^u_i = r^u_{i1} + r^u_{i2} + \cdots + r^u_{iN}$ *and* $t^u_i = t^u_{i1} + t^u_{i2} + \cdots + t^u_{iN}$ *and* $x^u_i = \tilde{x}^u_i + t^u_i$*.*
10. *Internally run* $\mathcal{G}$ *on input* $(x^u_i, \ldots, x^u_N)$*. Let* $(y^u_1, \ldots, y^u_N)$ *be the output of* $\mathcal{G}$*. For all* $i = 1,\ldots,N$*, compute* $o^u_i = y^u_i + r^u_i$ *and* $\theta^u_i \leftarrow \mathrm{Mac}(k_i, \texttt{Outp Round u}, y^u_i + r^u_i)$*. Increment counter* u*.*
11. *If party* P_i *requests an output for round* u'*, proceed as follows:*
    (i) If $u \leq u'$*, ignore.*
    (ii) Else, if $\texttt{verify}^0 = \texttt{false}$ *or* $\texttt{verify}^{\texttt{u}'} = \texttt{false}$*, send a private delayed output* $\perp$ *to* P_i*.*
    (iii) Else, send a private delayed output (o^u_i, θ^u_i) *to* P_i*.*
12. *Once all parties are corrupted, send all private randomness used so far as well as the private randomness* $\mathcal{G}$ *sends to* $\mathcal{A}$ *in this case (note that* $\mathcal{G}$ *is adaptively well-formed) to the adversary* $\mathcal{A}$*. (Note that this ensures that* $\mathcal{F}^{\mathsf{reac}}_{\mathcal{G}}$ *is also adaptively well-formed).*
13. *All other messages between* $\mathcal{A}$ *and* $\mathcal{G}$ *are ignored.*

Let $\mathcal{G}$ be a *reactive* standard adaptively well-formed functionality. We next define our protocol for realizing $\mathcal{G}$, which is denoted by $\Pi^{\mathsf{N-1,reac}}_{\mathcal{G}}$.

Construction 4 *Define the protocol* $\Pi^{\mathsf{N-1,reac}}_{\mathcal{G}}$ *as follows:*
Architecture: *See Fig. 5 for a graphical depiction.*
Offline Sharing Phase
Upon input x^1_i*, each party* P_i *does the following:*

– Disconnect *air-gap switch to the environment.*
– *Generate a key pair* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$*, a MAC key* $k_i \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$*, a signature key pair* $(\mathsf{sgk}_i, \mathsf{vk}_i) \leftarrow \mathrm{Gen}_{\mathrm{SIG}}(1^n)$ *and random pads* $t^1_i, t^2_i, \ldots, t^{\mathrm{R}}_i \leftarrow \{0,1\}^n$ *and* $r^1_i, r^2_i, \ldots, r^{\mathrm{R}}_i \leftarrow \{0,1\}^{p_i(n)}$*.*
– *Generate shares* $t^u_{i1} + t^u_{i2} + \cdots + t^u_{iN} = t^u_i$*,* $r^u_{i1} + r^u_{i2} + \cdots + r^u_{iN} = r^u_i$ $(u = 1,\ldots,\mathrm{R})$ *and* $k_{i1} + k_{i2} + \cdots + k_{iN} = k_i$*.*
– *Connect air-gap switch to J and to* IIM*.*
– *Send* (k_i, r^u_i) *to the* OIM *and* (k_i, t^u_i) $(u = 1,\ldots,\mathrm{R})$ *to the* IIM*.*
– *Send* $(\mathsf{pk}_i, \mathsf{vk}_i)$ *to the registration module via J and to* IIM*.*
– *Create signatures* $\sigma^u_{ij} \leftarrow \mathrm{Sig}(\mathsf{sgk}_i, u, P_j, t^u_{ij}, r^u_{ij})$ *and* $\sigma'_{ij} \leftarrow \mathrm{Sig}(\mathsf{sgk}_i, P_j, k_{ij})$ $(j = 1,\ldots,N; u = 1,\ldots,\mathrm{R})$*.*
– *Compute* $\tilde{x}^1_i = x^1_i + t^1_i$ *and* $\tau^1_i \leftarrow \mathrm{Mac}(k_i, \texttt{Inp Round 1}, \tilde{x}^1_i)$

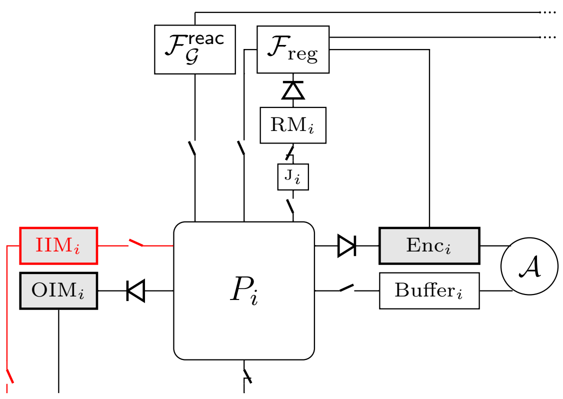

$\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$
$\mathcal{F}_{\mathrm{reg}}$
RM$_i$
J$_i$
IIM$_i$
OIM$_i$
P_i
Enc$_i$
Buffer$_i$
$\mathcal{A}$

Fig. 5: Architecture of $\mathit{\Pi}_{\mathcal{G}}^{\mathsf{N-1,reac}}$. Each party P_i ($i = 1, \dots, N$) has 3 hackable sub-parties, called *buffer*, *registration module* (RM) and *join* (J), and 3 unhackable sub-parties, called Enc(-unit), OIM and IIM. Buffer and Enc-unit are connected to the adversary via standard channels. All air-gap switches, except for P's air-gap switch to the environment and the RM's air-gap switch to J, are initially *disconnected*.

- *Let* $\bar{t}_{ij} = (t_{ij}^1, t_{ij}^2, \dots, t_{ij}^{\mathrm{R}})$, $\overline{r}_{ij} = (r_{ij}^1, r_{ij}^2, \dots, r_{ij}^{\mathrm{R}})$ *and* $\overline{\sigma}_{ij} = (\sigma_{ij}^1, \sigma_{ij}^2, \dots, \sigma_{ij}^{\mathrm{R}})$. *Send* $(j, \bar{t}_{ij}, \overline{r}_{ij}, \overline{\sigma}_{ij}, k_{ij}, \sigma'_{ij})$ $(j \in \{1, \dots, \mathrm{R}\} \setminus \{i\})$ *to the* Enc*-unit*
- Erase *everything except for the tuple* $(\bar{t}_{ii}, \overline{r}_{ii}, \overline{\sigma}_{ii}, k_{ii}, \sigma'_{ii})$ *and* $(\tilde{x}_i^1, \tau_i^1)$ *and* $\mathsf{vk}_i, \mathsf{sk}_i$.

Registration module and J: *On input* $(\mathsf{pk}_i, \mathsf{vk}_i)$ *to J, J forwards the input to RM. RM then* disconnects *air-gap switch to J and registers* pk_i *and* vk_i *by sending these keys to the public bulletin-board functionality* $\mathcal{F}_{\mathsf{reg}}$.

Enc-unit: *Receive a list* $L = \{(P_j, v_j)\}_{j=\{1,\dots,N\}\setminus\{i\}}$ *from one's main party* P_i. *At each activation, for each* $(P_j, v_j) \in L$, *request* pk_j *belonging to* P_j *from* $\mathcal{F}_{\mathsf{reg}}$. *If retrievable, compute* $c_{ij} \leftarrow \mathrm{Enc}(\mathsf{pk}_j, v_j)$, *send* (P_i, c_{ij}) *to the buffer of* P_j *and delete* (P_j, v) *from* L. *Then, go into idle mode.*

Buffer: *Store each message received. On input* `retrieve`, *send all stored messages to one's main party.*

First Online Compute Phase

Having completed its last step in the sharing phase, each party P_i *does the following:*

- Connect *air-gap switches to the* $buffer$, *to* $\mathcal{F}_{\mathsf{reg}}$ *and to* $\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$.
- *Request all verification keys* $\{\mathsf{vk}_l\}_{l\in\{1,\dots,N\}\setminus\{i\}}$ *from* $\mathcal{F}_{\mathsf{reg}}$ *registered by the other parties'* registration modules. *If not all verification keys can be retrieved yet, go into idle mode and request again at the next activation.*

- *Send* `retrieve` *to the buffer and check if the buffer sends at least $N-1$ messages. If no, go into idle mode and when activated again send* `retrieve` *and check again.*
  If yes, check if one has received from each party P_j a set $\mathcal{M}_j = \{(P_j, \tilde{c})\}$ with the following property:
  There exists a tuple $(P_j, \hat{\hat{t}}_{ji}, \hat{\hat{r}}_{ji}, \hat{\hat{\sigma}}_{ji}, \hat{k}_{ji}, \hat{\sigma}'_{ji})$, where $\hat{\hat{t}}_{ji} = (\hat{t}^1_{ji}, \hat{t}^2_{ji}, \ldots, \hat{t}^{\mathrm{R}}_{ji})$, $\hat{\hat{r}}_{ji} = (\hat{r}^1_{ji}, \hat{r}^2_{ji}, \ldots, \hat{r}^{\mathrm{R}}_{ji})$ and $\hat{\hat{\sigma}}_{ji} = (\hat{\sigma}^1_{ji}, \hat{\sigma}^2_{ji}, \ldots, \hat{\sigma}^{\mathrm{R}}_{ji})$, and an element $(P_j, c) \in \mathcal{M}_j$ such that
  - $\mathrm{Dec}(\mathsf{sk}_i, c) = (P_j, \hat{\hat{t}}_{ji}, \hat{\hat{r}}_{ji}, \hat{\hat{\sigma}}_{ji}, \hat{k}_{ji}, \hat{\sigma}'_{ji})$ *and* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, u, P_i, \hat{t}^u_{ji}, \hat{r}^u_{ji}, \hat{\sigma}^u_{ji}) = 1$ $(u = 1, \ldots, \mathrm{R})$ *and* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, \hat{k}_{ji}, \hat{\sigma}'_{ji}) = 1$
  - *For every $(P_j, \tilde{c}) \in \mathcal{M}_j$ it holds that either* $\mathrm{Dec}(\mathsf{sk}_i, \tilde{c}) = (P_j, \hat{\hat{t}}_{ji}, \hat{\hat{r}}_{ji}, \hat{\hat{\sigma}}_{ji}, \hat{k}_{ji}, \hat{\sigma}'_{ji})$ *or $(P_j, \tilde{c})$ is "invalid", i.e., either decrypts to $(P_j, \tilde{\tilde{t}}_{ji}, \tilde{\tilde{r}}_{ji}, \tilde{\tilde{\sigma}}_{ji}, \tilde{k}_{ji}, \tilde{\sigma}'_{ji})$ such that either* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, u, P_i, \tilde{t}^u_{ji}, \tilde{r}^u_{ji}, \tilde{\sigma}^u_{ji}) = 0$ *for some u or* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}_j, P_i, \tilde{k}_{ji}, \tilde{\sigma}'_{ji}) = 0$, *or decrypts to $(P', \tilde{\tilde{t}}_{ji}, \tilde{\tilde{r}}_{ji}, \tilde{\tilde{\sigma}}_{ji}, \tilde{k}_{ji}, \tilde{\sigma}'_{ji})$ where $P' \neq P_j$, or $\tilde{c}$ does not decrypt correctly.*

  If this does not hold, send $\perp$ to $\mathcal{F}^{\mathtt{reac}}_{\mathcal{G}}$. Else, send all verification keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$ as well as all tuples $(\hat{t}^1_{ji}, \hat{r}^1_{ji}, \hat{\sigma}^1_{ji}, \hat{k}_{ji}, \hat{\sigma}'_{ji})$ $(j \in \{1, \ldots, N\}$ and $(\tilde{x}^1_i, \tau^1_i)$ to $\mathcal{F}^{\mathtt{reac}}_{\mathcal{G}}$.
- *Instruct the* IIM *to* connect *its air-gap switch to $\mathcal{Z}$.*

Subsequent Online Compute Phases
Upon receiving an input x^u_i in round u, each IIM *does the following:*

IIM: *Initially, set $u = 2$. Compute $\tilde{x}^u_i = x^u_i + t^u_i$ and $\tau^u_i \leftarrow \mathrm{Mac}(k_i, \texttt{Inp Round u}, \tilde{x}^u_i)$ and send $(\tilde{x}^u_i, \tau^u_i)$ to one's main party. Increment u.*

- *Party P_i then sends $(\hat{t}^u_{ji}, \hat{r}^u_{ji}, \hat{\sigma}^u_{ji})$ $(j \in \{1, \ldots, N\}$ and $(\tilde{x}^u_i, \tau^u_i)$ to $\mathcal{F}^{\mathtt{reac}}_{\mathcal{G}}$.*

Online Output Phases
Having completed its last step in the compute phase in round u, a party P_i requests output from $\mathcal{F}^{\mathtt{reac}}_{\mathcal{G}}$ and forwards that output to OIM.

OIM: *Initially, set $u = 1$ and store the first input $(k_i, (r^1_i, \ldots, r^R_i))$ from one's main party. On subsequent inputs (o^u_i, θ^u_i) or $\perp$ from one's main party, do the following: If the received value equals $\perp$, output $\perp$. Otherwise, check if* $\mathrm{Vrfy}_{\mathrm{MAC}}(k_i, o^u_i, \theta^u_i) = 1$ *and output $y^u_i = o^u_i + r^u_i$ if this holds, and $\perp$ otherwise. Always increment u.*

We are now ready to state our theorem for reactive functionalities. The proof is similar to the proof of Theorem 3 and therefore omitted.

Theorem 5 (Up to $N-1$ Corrupted Parties, Reactive Case). *Let $\mathcal{G}$ be a reactive standard adaptively well-formed functionality. Let* PKE *and* SIG *be as in Theorem 3 and assume that* MAC *is EUF-CMA-secure. Then it holds that*

$$\Pi_{\mathcal{G}}^{\mathsf{N-1,reac}} \geq_{\#\#} \mathtt{AG}([\mathcal{G}])$$

for up to $N-1$ parties under adversarial control.

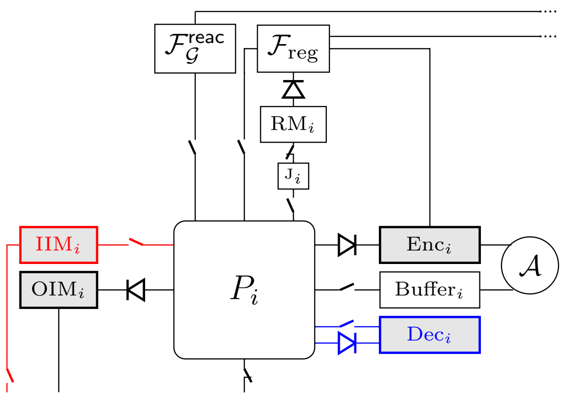
5.1 Up to N Parties Under Adversarial Control

With the same augmentation as described in Section 4.1, one can obtain a protocol $\Pi_{\mathcal{G}}^{\mathsf{N,reac}}$ that is also secure if the adversary corrupts *all* parties (cf. Fig. 6).

Theorem 6 (Up to N Corrupted Parties, Reactive Case). *Let $\mathcal{G}$ be a reactive standard adaptively well-formed functionality. Let* PKE, SIG, MAC *be as in Theorem 5. Then it holds that*

$$\Pi_{\mathcal{G}}^{\mathsf{N,reac}} \geq_{\#\#} \mathtt{AG}([\mathcal{G}])$$

for up to N parties under adversarial control.

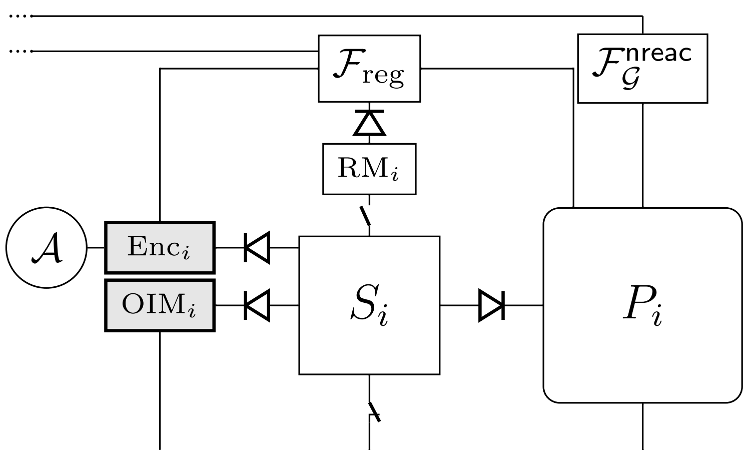


$\mathcal{F}_{\mathcal{G}}^{\mathsf{reac}}$
$\mathcal{F}_{\mathrm{reg}}$
RM$_i$
J$_i$
IIM$_i$
OIM$_i$
P_i
Enc$_i$
Buffer$_i$
Dec$_i$
$\mathcal{A}$

Fig. 6: Architecture of $\Pi_{\mathcal{G}}^{\mathsf{N,reac}}$. Each party P_i $(i = 1, \ldots, N)$ has 3 hackable sub-parties, called *buffer*, *registration module* (RM) and *join* (J), and 4 unhackable sub-parties, called Enc(-unit), Dec(-unit), OIM and IIM. Buffer and Enc-unit are connected to the adversary via standard channels. All air-gap switches, except for P's air-gap switch to the environment and the RM's air-gap switch to J, are initially *disconnected*.

6 Architectures without Erasure

We can obtain the results in Theorems 3 to 6 without relying on erasure by introducing an additional *hackable* interface module S that is connected to its main party P via a data diode and to the environment via an initially-connected air-gap switch (cf. Fig. 7 in Appendix A). S takes the (first) input and carries out the sharing phase. Afterwards, S sends its own shares together with their signatures (and for reactive functionalities also MAC tags) and the secret key to P, who then carries out all further computations. S is never activated again, remains offline, and thus cannot be corrupted though an `online-attack` instruction. Note, however, that S can only be reused in subsequent protocols if it can be reset to its initial state. Such a reset is in line with what is implicitly assumed in large parts of the MPC literature, e.g. in the UC framework, where parties holding secrets cease to exist after protocol execution.



$\mathcal{F}_{\text{reg}}$
$\mathcal{F}_{\mathcal{G}}^{\mathsf{nreac}}$
RM_i
$\mathcal{A}$
Enc_i
OIM_i
S_i
P_i

Fig. 7: Architecture without Erasure (for up to $N-1$ Parties under Adversarial control, Non-Reactive Case).

References

[1] D. Achenbach, J. Müller-Quade, and J. Rill. “Universally Composable Firewall Architectures Using Trusted Hardware”. In: *BalkanCryptSec 2014*.

[2] J. Baron, K. E. Defrawy, J. Lampkins, and R. Ostrovsky. “How to withstand mobile virus attacks, revisited”. In: *PODC 2014*.

[3] B. Broadnax, N. Döttling, G. Hartung, J. Müller-Quade, and M. Nagel. “Concurrently Composable Security with Shielded Super-Polynomial Simulators”. In: *EUROCRYPT 2017*.

[4] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *FOCS 2001*.

[5] R. Canetti, U. Feige, O. Goldreich, and M. Naor. “Adaptively Secure Multi-Party Computation”. In: *STOC 1996*.

[6] R. Canetti and M. Fischlin. “Universally composable commitments”. In: *CRYPTO 2001*.

[7] R. Canetti, E. Kushilevitz, and Y. Lindell. “On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions”. In: *EUROCRYPT 2003*.

[8] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. “Universally composable two-party and multi-party secure computation”. In: *STOC 2002*.

[9] R. Canetti, O. Poburinnaya, and M. Venkitasubramaniam. “Equivocating Yao: constant-round adaptively secure multiparty computation in the plain model”. In: *STOC 2017*.

[10] N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. “Implementing Resettable UC-Functionalities with Untrusted Tamper-Proof Hardware-Tokens”. In: *TCC 2013*.

[11] S. Garg, Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. “Cryptography with One-Way Communication”. In: *CRYPTO 2015*.

[12] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. “Founding Cryptography on Tamper-Proof Hardware Tokens”. In: *TCC 2010*.

[13] C. Hazay, Y. Lindell, and A. Patra. “Adaptively Secure Computation with Partial Erasures”. In: *PODC 2015*.

[14] C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam. “Constant Round Adaptively Secure Protocols in the Tamper-Proof Hardware Model”. In: *PKC 2017*.

[15] Y. Ishai, M. Prabhakaran, and A. Sahai. “Founding Cryptography on Oblivious Transfer - Efficiently”. In: *CRYPTO 2008*.

[16] J. Katz. “Universally Composable Multi-party Computation Using Tamper-Proof Hardware”. In: *EUROCRYPT 2007*.

[17] H. Nemati. “Secure System Virtualization: End-to-End Verification of Memory Isolation”. PhD thesis. Royal Institute of Technology, Stockholm, Sweden.

[18] R. Ostrovsky and M. Yung. “How to Withstand Mobile Virus Attacks (Extended Abstract)”. In: *PODC 1991*.

[19] Qubes OS Project. *Qubes Split GPG*. User Documentation. (Visited on 05/08/2018).

[20] E. Zheng, P. Gates-Idem, and M. Lavin. “Building a virtually air-gapped secure environment in AWS: with principles of devops security program and secure software delivery”. In: *HoTSoS 2018*.

Appendix

A Graphical Depiction of Architectures

This section contains graphical depictions of the architectures of the protocols in Sections 4 to 6. Main parties are represented by boxes with rounded corners, sub-parties and ideal functionalities by cornered ones. Boxes with bold lines and grey background denote that the sub-party is unhackable. Standard channels are denoted by lines, data diodes by —▷|— and air-gap switches by —╱— (initially disconnected) and —⧸— (initially connected). Dashed lines denote standard channels to other parties that are currently not shown. Downward connections from the main party and possibly from the OIMs or IIMs are to the environment (or the calling protocol).

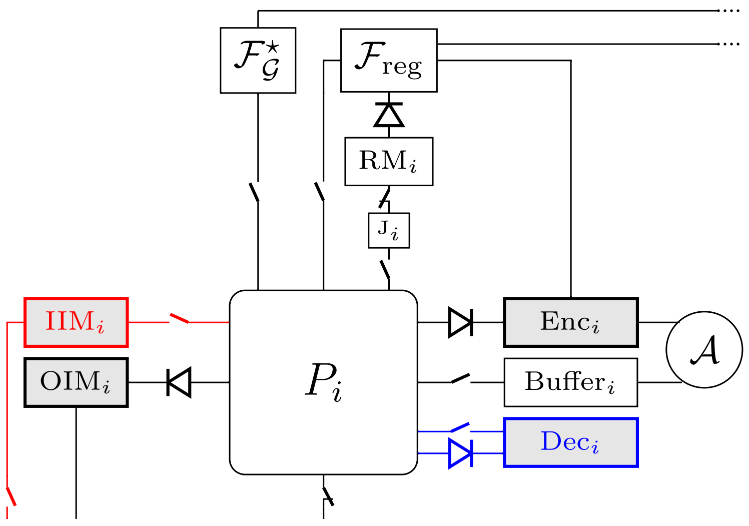


Fig. 8: Architecture for a) non-reactive functionalities and up to $N-1$ parties under adversarial control, if red and blue part absent, b) non-reactive functionalities and up to N parties under adversarial control, if blue part absent, c) reactive functionalities and up to $N-1$ parties under adversarial control, if red part is absent, and d) reactive functionalities and up to N parties under adversarial control, if blue and red is present. Note that $\star \in \{\mathsf{reac}, \mathsf{nreac}\}$ depending on $\mathcal{G}$.

B Definitions

B.1 Reactive / Non-Reactive Functionalities

A *non-reactive* functionality interacts with the parties in a single round, taking at most one input from each party and providing at most one output to each party.

In contrast, a *reactive* functionality may receive inputs and provide outputs in multiple rounds, possibly maintaining state information between rounds.

B.2 Standard corruption

Recall that an ideal functionality $\mathcal{F}$ is standard corruption if it proceeds as follows upon receiving a $(\texttt{corrupt}, P)$ message from $\mathcal{A}$. First, $\mathcal{F}$ marks P as $\mathsf{corrupted}$ and outputs $\texttt{corrupted}$ to P. In the next activation, $\mathcal{F}$ sends to $\mathcal{A}$ all the inputs and outputs of P so far. In addition, from this point on, whenever $\mathcal{F}$ gets an input value v from P, it forwards v to $\mathcal{A}$, who may then send a "modified input value" v' that overwrites v. Also, all output values intended for P are sent to $\mathcal{A}$ instead.

B.3 Standard Functionalities

We call an ideal functionality $\mathcal{G}$ *standard* if $\mathcal{G}$

1. immediately notifies the adversary upon receiving input from an (honest) party, and
2. is standard corruption, and
3. only gives delayed outputs to parties.

B.4 Well-Formed Functionalities

An ideal functionality is called *well-formed* if it consists of a "shell" and a "core". The core is an arbitrary PPT TM. The shell is a TM that acts as a "wrapper" in the following way: All incoming message are forwared to the core except for `corrupt` messages. Furthermore, outputs generated by the core are forwarded by the shell. Furthermore, an ideal functionality is *adaptively well-formed* if it consist of a shell and a core as described above and, in addition, the shell sends the random tape of the core to the adversary if all parties are corrupted at some activation.

B.5 Ideal Public Bulletin Board Functionality

In our constructions (Sections 4 and 5), we make use of the ideal functionality $\mathcal{F}_{\mathsf{reg}}$ that models a public bulletin board.

Definition 8 (Ideal Functionality $\mathcal{F}_{\mathsf{reg}}$). *$\mathcal{F}_{\mathsf{reg}}$ proceeds as follows:*

- *Report: Upon receiving a message $(\texttt{register}, sid, v)$ from party P, send $(\texttt{registered}, sid, P, v)$ to the adversary; upon receiving ok from the adversary, record the pair (P, v). Otherwise, ignore the message.*
- *Retrieve: Upon receiving a message $(\texttt{retrieve}, sid, P_i)$ from some party P_j (or the adversary $\mathcal{A}$), generate a public delayed output $(\texttt{retrieve}, sid, P_i, v)$ to P_j, where $v = \perp$ if no record (P, v) exists.*

Note that in contrast to the usual definition, we allow key revocation in $\mathcal{F}_{\mathsf{reg}}$.

B.6 Cryptographic Primitives

In the following, we define the cryptographic primitives used in this paper along with their required security properties.

Public-Key Encryption Schemes

Definition 9 (Public-Key Encryption Scheme). *Let* $\mathcal{M} \subseteq \{0,1\}^{p(n)}$ *be the message space. A* public-key encryption scheme $\mathrm{PKE} = (\mathrm{Gen}_{\mathrm{PKE}}, \mathrm{Enc}, \mathrm{Dec})$ *consists of three probabilistic polynomial-time algorithms such that:*

1. *The* key-generation algorithm $\mathrm{Gen}_{\mathrm{PKE}}$ *takes as input* 1^n *and outputs a tuple* $(\mathsf{pk}, \mathsf{sk})$. *We call* pk *the* public key *and* sk *the* private key *or* secret key.
2. *The* encryption algorithm Enc *takes as input a public key* pk *and a message* $m \in \mathcal{M}$ *and outputs a ciphertext* c.
3. *The* decryption algorithm Dec *takes as input a private key* sk *and a ciphertext* c *and outputs a message* $m \in \mathcal{M}$ *or a special symbol* $\perp$ *denoting failure.*

We call PKE perfectly correct *if* $\mathsf{Pr}[\mathrm{Dec}(\mathsf{sk}, \mathrm{Enc}(\mathsf{pk}, m)) = m] = 1$ *for any* $m \in \mathcal{M}$ *and for all* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$ *where the probability is over the random choices of* $\mathrm{Gen}_{\mathrm{PKE}}, \mathrm{Enc}$*, and* Dec.

Definition 10 (Indistinguishability Under Parallel Chosen Ciphertext Attack).
The experiment $\mathsf{Exp}^{\mathrm{IND-pCCA}}_{\mathcal{A}(z),\mathrm{PKE}}(n)$ *denotes the output of the following probabilistic experiment: At the beginning, the experiment generates keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$. *On input* 1^n, z *and* pk*, the adversary* $\mathcal{A}$ *chooses two messages* m_0, m_1 *and sends them to the experiment. The experiment then chooses a bit* b *uniformly random from* $\{0,1\}$ *and computes* $c^* \leftarrow \mathrm{Enc}(\mathsf{pk}, m_b)$. *During the experiment* $\mathcal{A}$ *may send a single* parallel *query to the oracle* $\mathcal{O}_{\mathrm{Dec(sk},\cdot)}$. *At the end of the experiment,* $\mathcal{A}$ *sends a bit* $b' \in \{0,1\}$. *The experiment then outputs* 1 *if* $b = b'$*, and* 0 *otherwise. The output of the experiment is replaced by a uniformly random bit* b^* *if during the experiment* $\mathcal{A}$ *queries* $\mathcal{O}_{\mathrm{Dec}(\mathsf{sk},\cdot)}$ *on a vector containing* c^*.

An adversary is called *valid* if he only chooses messages m_0, m_1 such that $|m_0| = |m_1|$ and his parallel query to $\mathcal{O}_{\mathrm{Dec(sk},\cdot)}$ does not contain c^*.

We call a public-key encryption scheme PKE IND-pCCA-secure if for every valid PPT-adversary $\mathcal{A}$ and all $z \in \{0,1\}^*$ there exists a negligible function negl such that

$$\mathsf{Pr}[\mathsf{Exp}^{\mathrm{IND-pCCA}}_{\mathcal{A}(z),\mathrm{PKE}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

Definition 11 (Indistinguishability under Adaptive Chosen Ciphertext Attack). *The experiment* $\mathsf{Exp}^{\mathrm{IND-CCA}}_{\mathcal{A}(z),\mathrm{PKE}}(n)$ *denotes the output of the following probabilistic experiment: At the beginning, the experiment generates a key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}_{\mathrm{PKE}}(1^n)$. *On input* 1^n, z *and* pk*, the adversary* $\mathcal{A}$ *chooses two messages* m_0, m_1 *of equal length and sends them to the experiment. The experiment* *then chooses a bit b uniformly random from $\{0,1\}$ and computes $c^* \leftarrow \mathrm{Enc}(\mathsf{pk}, m_b)$. On input 1^n, z, c^* and pk, the adversary may now make an arbitrary number of queries (not containing c^*) to a decryption oracle $\mathcal{O}_{\mathrm{Dec}(\mathsf{sk},\cdot)}$. At the end of the experiment, $\mathcal{A}$ sends a bit $b' \in \{0,1\}$. The experiment then outputs 1 if $b = b'$, and 0 otherwise. The output of the experiment is replaced by a uniformly random bit b^* if during the experiment $\mathcal{A}$ queries $\mathcal{O}_{\mathrm{Dec}(\mathsf{sk},\cdot)}$ on c^*.*

An adversary is called *valid* if he only chooses messages m_0, m_1 such that $|m_0| = |m_1|$ and does not query $\mathcal{O}_{\mathrm{Dec}(\mathsf{sk},\cdot)}$ on c^* during the experiment.

We call a public-key encryption scheme PKE *IND-CCA-secure* if for every valid PPT-adversary $\mathcal{A}$ and all $z \in \{0,1\}^*$ there exists a negligible function negl such that

$$\Pr[\mathsf{Exp}^{\mathrm{IND-CCA}}_{\mathcal{A}(z),\mathrm{PKE}}(n) = 1] \le \frac{1}{2} + \mathsf{negl}(n)$$

The security definitions of Definitions 10 and 11 can also be equivalently stated as follows. For every valid PPT-adversary $\mathcal{A}$ and all $z \in \{0,1\}^*$ there exists a negligible function negl such that

$$|\Pr[\mathrm{out}_0(\mathcal{A}) = 1] - \Pr[\mathrm{out}_1(\mathcal{A}) = 1]| \le \mathsf{negl}(n)$$

where b denotes the respective experiment's choice bit and $\mathrm{out}_b(\mathcal{A})$ denotes the bit sent to the experiment by $\mathcal{A}$.

Message Authentication Codes

Definition 12 (Message Authentication Code). *A* message authentication code $\mathrm{MAC} = (\mathrm{Gen}_{\mathrm{MAC}}, \mathrm{Mac}, \mathrm{Vrfy}_{\mathrm{MAC}})$ *consists of three probabilistic polynomial-time algorithms such that:*

1. *The* key-generation algorithm $\mathrm{Gen}_{\mathrm{MAC}}$ *takes as input 1^n and outputs a key k. We call k the* MAC key.
2. *The* tag-generation algorithm Mac *takes as input a MAC key k and a message m and outputs a* MAC tag t.
3. *The* verification algorithm $\mathrm{Vrfy}_{\mathrm{MAC}}$ *takes as input a MAC key k, a message m and a presumptive MAC tag t and outputs a bit $b \in \{0,1\}$, with $b = 1$ meaning* valid *and $b = 0$ meaning* invalid.

It is required that for every MAC key $k \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$ and every $m \in \{0,1\}^*$, it holds that $\Pr[\mathrm{Vrfy}_{\mathrm{MAC}}(k, m, \mathrm{Mac}(k,m)) = 1] = 1$, where the probability is over the random choices of $\mathrm{Gen}_{\mathrm{MAC}}, \mathrm{Vrfy}_{\mathrm{MAC}}$ and Mac. (correctness).

Definition 13 (Existential Unforgeability under One Chosen Message Attack for MACs). *We call a message authentication code* MAC *EUF-1-CMA-secure if for every PPT-adversary $\mathcal{A}$ and all $z \in \{0,1\}^*$ there exists a negligible function negl such that*

$$\Pr[\mathsf{Exp}^{\mathrm{EUF-1-CMA}}_{\mathcal{A}(z),\mathrm{MAC}}(n) = 1] \le \mathsf{negl}(n)$$

The experiment $\mathsf{Exp}^{\mathrm{EUF-1-CMA}}_{\mathcal{A}(z),\mathrm{MAC}}(n)$ *denotes the output of the following probabilistic experiment: At the beginning, the experiment generates a key* $k \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$*. On input* 1^n *and* z*, the adversary* $\mathcal{A}$ *may send a single query* m' *to an oracle* $\mathcal{O}_{\mathrm{Mac}(k,\cdot)}$*. Afterwards,* $\mathcal{A}$ *outputs a tuple* (m^*, t^*)*. If* $\mathrm{Vrfy}_{\mathrm{MAC}}(k, m^*, t^*) = 1$ *and* $m^* \neq m'$*, the experiment outputs 1, else 0.*

Definition 14 (Existential Unforgeability under Chosen Message Attack for MACs). *We call a message authentication code* MAC *EUF-CMA-secure if for every* PPT*-adversary* $\mathcal{A}$ *and all* $z \in \{0,1\}^*$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\mathrm{EUF-CMA}}_{\mathcal{A}(z),\mathrm{MAC}}(n) = 1] \leq \mathsf{negl}(n)$$

The experiment $\mathsf{Exp}^{\mathrm{EUF-CMA}}_{\mathcal{A}(z),\mathrm{MAC}}(n)$ *denotes the output of the following probabilistic experiment: At the beginning, the experiment generates a key* $k \leftarrow \mathrm{Gen}_{\mathrm{MAC}}(1^n)$*. On input* 1^n *and* z*, the adversary* $\mathcal{A}$ *may send queries to an oracle* $\mathcal{O}_{\mathrm{Mac}(k,\cdot)}$*. Let* $\mathcal{Q}$ *be the set of all queries. Eventually,* $\mathcal{A}$ *outputs a tuple* (m^*, t^*)*. If* $\mathrm{Vrfy}_{\mathrm{MAC}}(k, m^*, t^*) = 1$ *and* $m^* \notin \mathcal{Q}$*, the experiment outputs 1, else 0.*

Digital Signature Schemes

Definition 15 (Digital Signature Scheme). *A* digital signature scheme $\mathrm{SIG} = (\mathrm{Gen}_{\mathrm{SIG}}, \mathrm{Sig}, \mathrm{Vrfy}_{\mathrm{SIG}})$ *consists of three probabilistic polynomial-time algorithms such that:*

1. *The* key-generation algorithm $\mathrm{Gen}_{\mathrm{SIG}}$ *takes as input* 1^n *and outputs a tuple* $(\mathsf{vk}, \mathsf{sgk})$*. We call* vk *the (public)* verification key *and* sgk *the (private)* signing key *or* signature key*.*
2. *The* signature-generation algorithm Sig *takes as input a signing key* sgk *and a message* m *and outputs a* signature σ*.*
3. *The* verification algorithm $\mathrm{Vrfy}_{\mathrm{SIG}}$ *takes as input a verification key* vk*, a message* m *and a presumptive signature* σ *and outputs a bit* $b \in \{0,1\}$*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*.*

It is required that for every key pair $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathrm{Gen}_{\mathrm{SIG}}(1^n)$ *and every* $m \in \{0,1\}^*$*, it holds that* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}, m, \mathrm{Sig}(\mathsf{sgk}, m)) = 1$ *(correctness).*

Definition 16 (Length-Normal Digital Signatures). *A digital signature scheme* SIG *is* length-normal *if for every key pair* $(\mathsf{vk}, \mathsf{sgk})$ *output by* $\mathrm{Gen}_{\mathrm{SIG}}(1^n)$ *and all* $m, m' \in \{0,1\}^*$ *such that* $|m| = |m'|$ *the following holds: If* $\sigma \leftarrow \mathrm{Sig}(\mathsf{sgk}, m), \sigma' \leftarrow \mathrm{Sig}(\mathsf{sgk}, m')$ *then* $|\sigma| = |\sigma'|$*.*

Definition 17 (Existential Unforgeability under Non-Adaptive Chosen Message Attack for Digital Signature Schemes). *We call a digital signature scheme* SIG *EUF-naCMA-secure if for every* PPT*-adversary* $\mathcal{A}$ *and all* $z \in \{0,1\}^*$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\mathrm{EUF-naCMA}}_{\mathcal{A}(z),\mathrm{SIG}}(n) = 1] \leq \mathsf{negl}(n)$$

The experiment $\mathsf{Exp}^{\mathrm{EUF-naCMA}}_{\mathcal{A}(z),\mathrm{SIG}}(n)$ *denotes the output of the following probabilistic experiment: At the beginning, the experiment generates keys* $(\mathsf{vk}, \mathsf{sgk}) \leftarrow \mathrm{Gen}_{\mathrm{SIG}}(1^n)$*. On input* 1^n *and* z*, the adversary* $\mathcal{A}$ *may send queries to a signing oracle* $\mathcal{O}_{\mathsf{Sig}(\mathsf{sgk},\cdot)}$*. Let* $\mathcal{Q}$ *be the set of all queries. Afterwards on input* 1^n*,* z *and* vk*,* $\mathcal{A}$ *outputs a tuple* (m^*, σ^*)*. If* $\mathrm{Vrfy}_{\mathrm{SIG}}(\mathsf{vk}, m^*, \sigma^*) = 1$ *and* $m^* \notin \mathcal{Q}$*, the experiment outputs 1, else 0.*

C A Short Introduction to the UC Framework

In the following, we give a brief overview of the UC framework. The following is adapted from [3]. For a detailed introduction see [4].

In the UC framework, security is defined by the indistinguishability of two experiments: the *ideal experiment* and the *real experiment*. In the ideal experiment, the task at hand is carried out by dummy parties with the help of an ideal incorruptible entity—called the ideal functionality $\mathcal{F}$. In the real experiment, the parties execute a protocol π in order to solve the prescribed tasks themselves. A protocol π is said to be a (secure) *realization* of $\mathcal{F}$ if no PPT-machine $\mathcal{Z}$, called the *environment*, can distinguish between these two experiments. In contrast to previous simulation-based notions, indistinguishability must not only hold after the protocol execution has completed, but even if the environment $\mathcal{Z}$—acting as the *interactive* distinguisher—takes part in the experiment, orchestrates all adversarial attacks, gives input to the parties running the challenge protocol, receives the parties' output and observes the communication during the whole protocol execution.

The basic model of computation. The basic model of computation consists of a set of (a polynomial number of) instances (ITIs) of interactive Turing machines (ITMs). An ITM is the description of a Turing machine with an additional identity tape, three externally writable input tapes (namely for input, subroutine output[22] and incoming messages) and an outgoing message tape. The latter is jointly used to provide input to any of the three input tapes of another ITM. The tangible instantiation of an ITM—the ITI—is identified by the content of its identity tape. The order of activation of the ITIs is completely asynchronous and message-driven. An ITI gets activated if input, subroutine output or an incoming message is written onto its respective tape. If the ITI writes onto its outgoing

[22] Beware: Despite its name this tape is actually an input tape as it receives subroutine output.

message tape and calls the special `external write` instruction, the activation of this ITI completes. The message must explicitly designate the identity and input tape of the receiving ITI. Each experiment comprises two special ITIs: The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ (in the real experiment) or the simulator $\mathcal{S}$ (in the ideal experiment). The environment $\mathcal{Z}$ is the ITI that is initially activated. If any ITI completes its activation without giving any output, the environment is activated again as a fall-back. If the environment $\mathcal{Z}$ provides subroutine output, the whole experiments stops. The output of the experiment is the output of $\mathcal{Z}$. Without loss of generality, we assume that $\mathcal{Z}$ outputs a single bit only.

The Control Function and Message Delivery. If an ITI writes a message onto its outgoing message tape and calls `external write`, a *control function* decides if the operation is allowed[23]. If so, the experiment proceed as follows: If the receiver is uncorrupted and the designated input tape is either *input* or *subroutine output*, the message is copied to the respective tape of receiver. Else (meaning if the message is intended to be sent to an *incoming message* tape or the receiver is corrupted) the message is delivered to the respective tape of the adversary. This captures the natural intuition that input and subroutine output normally occurs within the same physical party and thus should be authenticated, immediate, confidential and of integrity. In contrast, external communication is only possible through an unreliable network under adversarial the control.

UC Framework Conventions. In the UC framework, many important aspects are unspecified. For example, it leaves open which ITI is allowed to invoke what kind of new ITIs. The conventions stated in the following are probably the mostly used ones and quite natural.

Each party is identified by its party identifier (PID) *pid* which is unique to the party and is the UC equivalent of the physical identity of this party. A party runs a protocol π by means of an ITI which is called the main party of this instance of π. An ITI can invoke subsidiary ITIs to execute sub-protocols. A subsidiary and its parent use their *input/subroutine output* tape to communicate with each other. The set of ITIs taking part in the same protocol but for different parties communicate through their *incoming message* tapes. An instance of a protocol is identified by its session identifier (SID) *sid*. All ITIs taking part in the same protocol instance share the same SID. A specific ITI is identified by its ID $\mathsf{id} = (pid, sid)$.

The (Dummy) Adversary. The adversary $\mathcal{A}$ is instructed by $\mathcal{Z}$ and represents $\mathcal{Z}$'s interface to the network. To this end, all messages from any party to a party that has a different main party and that are intended to be written to an *incoming message* tape are copied to the adversary. The adversary can process the message

[23] N.b.: The control function is another ITI that exists "outside" of the experiment and checks which combination of sender ID, receiver ID and message tape are feasible. For example, only subsidiary ITIs are typically allowed to provide subroutine output to their main ITI. For details see [4].

arbitrarily. The adversary may decide to deliver the message (by writing the message on its own outgoing message tape), postpone or completely suppress the message, inject new messages or alter messages in any way including the recipient and/or alleged sender.

$\mathcal{Z}$ may also instruct $\mathcal{A}$ to corrupt a party. In this case, $\mathcal{A}$ takes over the position of the corrupted party, reports its internal state to $\mathcal{Z}$ and from then on may arbitrarily deviate from the protocol in the name of the corrupted party as requested by $\mathcal{Z}$. This means whenever the corrupted ITI would have been activated (even due to subroutine output), the adversary gets activated with the same input.

Ideal Functionalities and the Ideal Protocol. An ideal functionality $\mathcal{F}$ is a special type of ITM whose instantiations (ITIs) bear a SID but no PID. Hence, it is an exception to the aforementioned identification scheme. Input to and subroutine output from $\mathcal{F}$ is performed through dummy parties. Dummy parties merely forward their input to the input tape of $\mathcal{F}$ and subroutine output from $\mathcal{F}$ to their own outgoing message tape. They share the same SID as $\mathcal{F}$, but additionally have individual party identifiers (PIDs) as if they were the actual main parties of a (real) protocol. The ideal functionality $\mathcal{F}$ is simultaneously a subroutine for each dummy party and conducts the prescribed task. IDEAL($\mathcal{F}$) is called the *(ideal) protocol* for $\mathcal{F}$ and denotes the set of $\mathcal{F}$ together with its dummy parties.

The UC Experiment. Let π be a protocol, $\mathcal{Z}$ an environment and $\mathcal{A}$ an adversary. The UC experiment, denoted by $\mathrm{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(n,a)$, initially activates the environment $\mathcal{Z}$ with security parameter 1^n and input $a \in \{0,1\}^*$. The first ITI that is invoked by $\mathcal{Z}$ is the adversary $\mathcal{A}$. All other parties invoked by $\mathcal{Z}$ are set to be main parties of the challenge protocol π. $\mathcal{Z}$ freely chooses their input, their PIDs and the SID of the challenge protocol. The experiment is executed as outlined above.

Definition of Security. A protocol π is said to *emulate* (or UC-*realize*) another protocol ρ, denoted by $\pi \geq_{\text{UC}} \rho$, if and only if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z} : \mathrm{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(n,a) \overset{\text{c}}{\equiv} \mathrm{Exec}_{\rho,\mathcal{S},\mathcal{Z}}(n,a)$$

holds for all $a \in \{0,1\}^*$ with the probability on the left and on the right being taken over the initial input of $\mathcal{Z}$ and all random tapes of all PPT machines. The adversary $\mathcal{S}$ on the right side is called simulator. Please recall that the experiment silently ensures that the main parties of the challenge protocol are instantiated by π or ρ respectively. Usually, the security of a (real) protocol π is analyzed with respect to an (ideal) protocol IDEAL($\mathcal{F}$) for an ideal functionality $\mathcal{F}$. By abuse of notation we simply write $\pi \geq_{\text{UC}} \mathcal{F}$, i.e.

$$\pi \geq_{\text{UC}} \mathcal{F} \iff \pi \geq_{\text{UC}} \mathrm{IDEAL}(\mathcal{F}) \iff \\ \forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z} : \mathrm{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(n,a) \overset{\text{c}}{\equiv} \mathrm{Exec}_{\mathrm{IDEAL}(\mathcal{F}),\mathcal{S},\mathcal{Z}}(n,a)$$

for all $a \in \{0,1\}^*$. The simulator $\mathcal{S}$ mimics the adversarial behavior to the environment as if this were the real experiment with real parties carrying out the real protocol with real π-messages. Moreover, $\mathcal{S}$ must come up with a convincing internal state upon corrupted parties, consistent with the simulated protocol execution up to this point (dummy parties do not have an internal state).

Protocol Composition. UC security is closed under protocol composition: Let π, ϕ, ρ be protocols. Then,

$$\pi \geq_{\text{UC}} \phi \implies \rho^{\pi} \geq_{\text{UC}} \rho^{\phi}$$