On the (Ir)Replaceability of Global Setups, or How (Not) to Use a Global Ledger

Christian Badertscher*[1], Julia Hesse[2], and Vassilis Zikas†[3]

[1] *IOHK, Switzerland,* christian.badertscher@iohk.io
[2] *IBM Research – Zurich, Switzerland,* jhs@zurich.ibm.com
[3] *University of Edinburgh, United Kingdom,* vzikas@inf.ed.ac.uk

November 27, 2020

Abstract

In a universally composable framework, a global setup is intended to capture the ideal behavior of a primitive which is accessible by multiple protocols, allowing them to share state. The ledger implemented by blockchain protocols such as Bitcoin is a representative example of such global setup, since the Bitcoin ledger is known to be useful in various scenarios. Therefore, it has become increasingly popular to capture such ledgers as a global setup. One would hope that this allows one to make security statements about protocols that use such a global setup, e.g., a global ledger, which can then be automatically translated into the setting where the setup is replaced by a protocol implementing it, such as Bitcoin.

We show that the above reasoning is flawed and such a generic security-preserving replacement can only work under very (often unrealistic) strong conditions on the global setup. For example, the composable security of Bitcoin, cast as realizing an ideal ledger such as the one by Badertscher *et al.* [CRYPTO'17], is *not* sufficient per se to allow us to replace the ledger by Bitcoin when used as a global setup and to expect that security statements that are made in the global ledger-hybrid world would be preserved.

On the positive side, we provide characterizations of security statements for protocols that make use of global setups, for which the replacement is sound. Our results can be seen as a first guide on how to navigate the very tricky question of what constitutes a "good" global setup and how to use it in order to keep the modular protocol-design approach intact.

*Work done in part while the author was visiting the *Simons Institute for the Theory of Computing*, UC Berkeley.
†Work done in part while the author was visiting the *Simons Institute for the Theory of Computing*, UC Berkeley; work supported in part by Sunday Group.

Contents

1 Introduction

Universally Composable (UC) security [Can01] ensures strong composability guarantees: Informally, a UC secure protocol π remains secure no matter what environment it is placed in, i.e., no matter what other protocols might be executed alongside π; and if π securely realizes a given functionality $\mathcal{F}$—i.e., π emulates the $\mathcal{F}$-dummy protocol ϕ— then π can be used to replace $\mathcal{F}$ in any "context", i.e., within every protocol that might rely on calls to $\mathcal{F}$. This powerful security definition enables a constructive approach to protocols, where protocols can be designed and proved secure assuming access to idealized primitives/functionalities, and then the assumed functionalities can be realized by protocols securely emulating them. However, the power of any security definition lies not only in what it allows us to do, but also, and perhaps more importantly, in answering the following question: How well does this definition capture the execution of cryptographic protocols in reality?

Canetti and Rabin [CR03] pointed out a limitation of the standard universal composition theorem. In a nutshell, the issue is as follows: Assume that a protocol π_1 securely realizes a functionality $\mathcal{F}_1$ in the public-key infrastructure (PKI) model. The UC composition theorem ensures that $\mathcal{F}_1$ can be replaced by π_1 (and its PKI). However, if a protocol makes two calls to two independent instances of $\mathcal{F}_1$, then any replacing instance of π_1 needs to come with its own independent (local to π_1) PKI; in other words, the two replaced instances of π_1 cannot share the same PKI. In fact the issue is more general and applies to any protocol that makes calls to two (or more) functionalities $\mathcal{F}_1$ and $\mathcal{F}_2$. Assuming each of $\mathcal{F}_1$ and $\mathcal{F}_2$ is UC realized by protocol π_1 and π_2, respectively, where each of the π_i's uses a PKI,[1] then in order for us to replace $\mathcal{F}_1$ by π_1 and $\mathcal{F}_2$ by π_2 we need two independent PKIs, each of which is *local* to its protocol! This is a clear mismatch with reality, where we would not create a different PKI for each protocol (instance), but rather people would have one public-key/private-key pair which they would use in multiple protocols.

In addition to pointing out the above mismatch, [CR03] proposed a refinement of the UC composition theorem, termed *UC composition with Joint state*, often referred to as the *JUC theorem.* This theorem allows to tackle the issue for the case where the π_i's and $\mathcal{F}_1$'s are instances of the same protocol and functionality, e.g., we have multiple invocations of a digital signature protocol all of which use the same PKI. However the JUC theorem suffered from its own limitation: It can only be applied if we know in advance (the number and even the session identifiers of) the protocols that will be using the common setup. This is, again, problematic in capturing reality, where a PKI is created by a party registering its public key with a certification authority, without even knowing all the intended uses of it.

The first attempt to resolve the above mismatch in a UC-like manner was made by Canetti *et al.* [CDPW07] with the introduction of *UC security with Global setups*, often referred to as the *GUC framework*. This framework redefined some of the basic features of the UC model of execution which allowed protocols to access the same functionality—and share their state through it—and came equipped with a composition theorem that allowed the protocols used to replace different functionalities to share the same setup. Although the underlying motivation is sound, the GUC model has some inconsistencies (see [BCH+20] for a discussion) which spill into its composition theorem. Furthermore, its redesign of some core UC concepts make it somewhat incompatible with *plain* UC. Motivated by the above, a recent work [BCH+20] proposed a way to equip the (recently updated) plain UC framework with a universal composition theorem that allows the replaced protocols to use the same global setup/functionality, a theorem termed the *UCGS theorem*. Importantly, and unlike GUC, the global setup/functionality in [BCH+20] is not any special object new to the framework. Rather, it is a standard UC functionality; what makes it "global" is just the fact that it is used by

[1] The statement applies also to any non-trivial type of hybrid functionalities whose use might correlate the views of the protocols calling them, e.g., the common reference string (CRS).

different protocols (that are not subprocesses of the same calling protocol). This allows, instead of shifting to a new framework, to equip plain UC with a global-setup composition theorem, thereby automatically preserving the security statement in the vast UC literature.

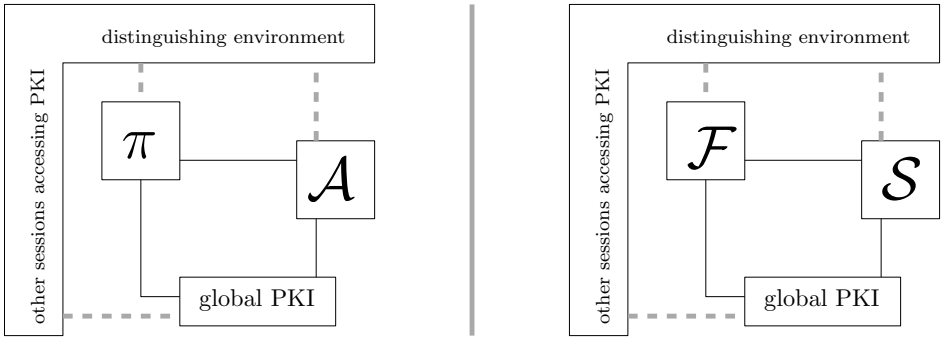
Notwithstanding, despite the cryptographic literature coming a long way in eliminating the mismatch of the UC framework local-functionalities modelling with the reality of (global) PKIs, the following important questions remains unresolved:

> Can we replace a global setup with a (globally accessible) protocol realizing it, and does such a replacement preserve security statements of protocols jointly using the global setup as subroutine?

Arguably, being able to answer such a question is essential for assessing the power of the framework. For example, without an answer to the above question, it is unclear how useful for reality is a statement assuming a global PKI. Indeed, such a PKI would typically not be created by a trusted party, but rather generated by its users by means of a protocol involving a certification authority of a publicly trusted repository. Thus, in order for statements in the (global) PKI-hybrid model [CSV16, PS18, DPS19] to be realistic, one should be able to argue that it is safe to replace the PKI with a protocol as the above. The same holds for statements made w.r.t other commonly used global UC setups in the literature, such as the global ledger [KZZ16, CGL+17, DFH18, DEFM19, DEF+19, EMM19, CGJ19, ACKZ20, KL20], the global clock [KZZ16, BGK+18, DFH18, DEF+19], various flavors of a global random oracles [CJS14, BGK+18, CDG+18] and a global CRS [CKWZ13]. Returning to the above question, the first thing one needs to address is what it means to "realize" a global setup.

To our knowledge, with the exception of [CSV16] which studied the above question in GUC for the case of a global PKI, no work has attempted to define such a notion of realization. This is perhaps due to the fact that up until recently, a global setup was a special type in the GUC framework, so that answering this question would mean defining a special type of global-setup protocols in GUC which was never defined before. However, this above limitation has been lifted by [BCH+20]: Since a global setup is not a new type, but rather a standard functionality which might be accessed by protocols from different sessions, a "global protocol" also does not need to be any new type (but is rather a property of how the protocol is being used). It is just a standard protocol which might be called by protocols with different sessions. Hence, the most natural notion of implementation is UC emulation.

Equipped with this realization notion, we can now focus on the more interesting part of the above question, i.e., whether replacing a global subroutine by its emulation preserves the security statements. One might quickly jump into the conclusion that since the UC framework allows subroutine replacement, the answer must be "yes". But as we discuss below the surprising answer is actually: "not necessarily." Unlike setups that are available to only one protocol session, global setups are standalone protocols that are typically accessible by many sessions – including the "local" adversaries of these sessions. The situation can be depicted as follows, with the real protocol execution with π accessing a global PKI on the left. The right-hand side depicts the ideal execution with functionality $\mathcal{F}$ that π UC-emulates, with a simulator $\mathcal{S}$ which makes the execution look like an execution of π with $\mathcal{A}$.



The above means that we understand the security of π by knowing its idealization $\mathcal{F}$ to which it is indistinguishable for all practical purposes - but where the context is equipped with the global PKI that the protocol is using, too. Therefore, the specification of $\mathcal{F}$ is tied with respect to this context. If the setup is replaced by a protocol that UC-emulates it, this means we switch the context, and this is where it gets problematic. We do not speak here of any technicality regarding the syntactic replacement of a subroutine, this will happen the same way as standard UC thanks to the embedding of globality in UC. The real issue that arises with this replacement is that the program $\mathcal{S}$ might not function properly if its interface at the global PKI becomes restricted – and that may happen since the replacement UC-emulates the global PKI, and hence might allow for less attacks[2]. In fact, in Appendix A we give example settings in which a global replacement by UC emulation is indeed impossible. This motivates a refined question which is at the core of our work, namely

> Under what conditions can a global setup be safely replaced by a protocol that UC emulates it, without this replacement affecting the security of protocols using the setup as a subroutine?

Having an answer to the above is essential for ensuring that statements involving global setups have an analogue in a realistic setting, where the setup is not offered by a trusted party but rather emulated by a protocol itself. In the upcoming section we discuss a number of situations in which not addressing the above question leads to counter-intuitive results.

Our results. We provide various conditions under which replacement of a global subroutine by a protocol realizing it does not affect the validity of the underlying security statement. Our results give a partial guide on how to navigate the very tricky question of what constitutes a "good" global setup. More concretely, we provide the following theorems for soundly replacing global setups by their emulation in existing security statements. We note that only the first replacement strategy is conditioned on the global setup and its emulation, and is oblivious of the underlying security statement. Contrary, the latter replacement strategy requires us to put conditions on the simulator of the underlying security statement.

Replacement with equivalent setup. A setup can be replaced with its implementation if the implementation is actually equivalent to the setup, including adversarial capabilities. The formulation of equivalence of adversarial capabilities is formalized using the simulation argument: after replacing, there must be an efficient way to emulate all queries that were available before.

[2]Note that this does not contradict the UC composition theorem. Indeed, if the PKI would be local to the instance of π, the simulator $\mathcal{S}$ consumes the PKI in the ideal world, and hence $\mathcal{S}$'s external interfaces do not change.

Replacements for agnostic simulations. As will be apparent from our discussion below on selected case-studies, the requirement on equivalence is impractical and unrealistic if we want to understand what a protocol ideally should achieve. Therefore, we need a more fine-grained condition. In a nutshell, we prove two things. The replacement of a global setup $\mathcal{G}$ in a protocol π UC-realizing ϕ is sound if the simulator witnessing this construction is either:

- agnostic of the adversarial capabilities of $\mathcal{G}$ and the only dependence is on exported capabilities that are available to honest parties, too.
- characterized by a set I of adversarial queries that are admissible, a concrete technical condition that generalizes the idea that adversarial capabilities and their actions will be preserved once $\mathcal{G}$ is going to be replaced.

The first condition on the simulator is appealing as it is simple to check. Furthermore, since $\mathcal{F}$ can communicate with $\mathcal{G}$ naturally via an ordinary party identifier (see [BCH+20]) or in its own "name", the simulator $\mathcal{S}$ can obtain information via $\mathcal{F}$ and hence use $\mathcal{G}$ just like an honest caller of the protocol.

The second condition brings more flexibility to protocol designers since $\mathcal{S}$ can use certain capabilities I at the adversarial interface. Intuitively speaking, Section 4 states the condition which sets I of adversarial queries are fine to use relative to an implementation (or a set of implementations) that is going to replace the setup.

Our theorems further follow by applications of the UCGS and UC composition theorems at a level of abstraction which seems to share a lot of similarities with other frameworks and their composition theorems. For example, the exact corruption model is irrelevant, as long as the behavior of and upon corruption can be formulated via an "adversarial" interface, where the above conditions can be evaluated on (such as the backdoor tape in UC). Therefore, we believe that our results are natural and of importance to other frameworks, too.

Why replacing the setup in both worlds? We point out that we also formally prove a related but weaker statement: we could just let π make subroutine calls to the replacement of $\mathcal{G}$, and leave the ideal world to be $\mathcal{F}$ in combination with context $\mathcal{G}$. This is a formally sound statement and we prove it in this work as a warm-up. The reason this cannot be the last word is the following: the different contexts allow to completely obscure the achieved level of security as formalized by $\mathcal{F}$. The high-level reason is that $\mathcal{F}$ is misleading in its role as idealization of π if we ignore the context. For example, $\mathcal{F}$ can offer much better security guarantees (for example, less powerful adversarial interface) *thanks to* the weak context offering more adversarial capabilities. In the sum, the real world is stronger and the ideal world is weaker (hence the statement must go through) but we still do not learn what the idealization of π would be. We provide a simple example for such an obscuring in Appendix A.2.

Related Work. To our knowledge, there is very limited work on the replaceability of a global UC setup. In fact, the only work that has looked at the question at an analogous generality as here is [CSV16]. As discussed above, the treatment there is in GUC which requires considerable effort to even define a "global" protocols, and even then, the treatment inherits the inconsistencies of the GUC model. Most importantly, although [CSV16] does identify necessary and sufficient conditions on the global setup and protocol replacing it to allow a generic preservation of security properties, these condition is too strict to be applied on more complicated primitives, such as blockchain ledgers, which have recently become a standard example of global subroutines. We remark that although our results are described using the recent UCGS modelling of [BCH+20], they can easily be adapted to

any framework which supports universal composition and global setups. Finally, an investigation of replaceability targeted to special variants of global random oracles was recently made in [CDG+18]; their approach is contrary to ours in the sense that their work investigates replacement by weaker global setups. Such a replacement can only be sound for contrived interpretations of "weaker" (ones that only take away leverage from the real-world adversary, but not the simulator) and hence does not work for realistic protocols such as Bitcoin or a protocol implementing a PKI.

Organization of the remainder of this paper. In Section 2 we discuss the pitfalls of global setup replacement with three concrete global setups that frequently appear in the literature. Section 3 gives an informal introduction to the UC framework and on how to formalize global setups in it using the language of UCGS [BCH+20]. We also recall all definitions relevant for this work in Appendix B. In Section 4 we deal with protocols accessing only a single global setup. Finally, we generalize our concepts to many global subroutines in Section 5.

2 (Ir)Replaceability of Common Global Setups

We now discuss more concretely the issues that arise when replacing global setups. Although originally introduced and broadly regarded as a concept of purely theoretical interest, several global setups are recently finding their way into modeling of practical applications. We pick three representatives for global setups, namely the (global) RO, PKI, and transaction ledger. With the RO, we demonstrate the impact of our observation on feasibility results. The PKI is exemplary for setups that are replaced by equivalence transformations. Finally, the ledger falls into the category of an idealization of a complex protocol where our main theorems must be used.

2.1 Global Random Oracles

Random oracles are often used to abstract a hash function used in cryptographic protocols. This abstraction makes a global random oracle less relevant for the scope of our work, as the replacement with a hash function does not satisfy any notion of secure realization (it is just a heuristic argument). Nonetheless, the random oracle model being one of the most commonly used abstractions in practice, it offers the basis for a smooth introduction to the pitfalls of global setup "replacement". For this, we consider different variants of global random oracles from the literature and investigate the effects of replacing one variant with another.

Introduction to Global Random Oracles. The standard instantiation of the (local) random oracle model in UC assumes that protocol parties have access to an ideal functionality which behaves as a random function. When we want to prove that a protocol π using such a random oracle realizes an ideal functionality $\mathcal{F}$, then in the security proof, the ideal world simulator is allowed to fully control/simulate the behavior of the random oracle. This allows it to arbitrary define the function it implements, a property typically referred to as *programmability*. In fact, this programability is what gives us the ability to prove a number of non-trivial feasibility results in the RO model. The reason is that it gives the simulator a comparative advantage over the real-world adversary who cannot program the random oracle.

Using our intuition discussed in the introduction, that a global setup is nothing more than a UC functionality which is used by protocols with different sessions, the natural way of capturing a "global" RO is to include it also in the ideal world. However, this will remove the ability of the simulator to program the RO, which is known to bring back most impossibility results that we circumvented in the local RO setting, rendering this global RO mostly useless.

Why global replacement intuitively fails. The above asymmetry of programmable vs. non-programable ROs already provides a first demonstration of the irreplaceability principle. Indeed, let π (resp. ϕ) denote the ideal (dummy) protocol in the non-programmable-RO-hybrid (resp. programable-RO-hybrid) model. It is straight-forward to verify that π UC-emulates ϕ. However, the known insufficiency of the non-programable-RO for certain tasks, such as non-committing encryption [Nie02], which can be realized from a programmable RO, indicates that we cannot use π to replace ϕ in arbitrary contexts.

Example I: A feasibility result. We now give a formal example where replacement of the global random oracle does not preserve the security statement. In [CJS14], a weakend version of the above global RO, denoted as gRO, was proposed. Informally, this version does not allow the simulator to program the RO, but does allow him to query it in a private manner, protecting him against an environment that might try to use its read access to the gRO to check the simulators claims about the answers to queries of simulated honest parties. This protection is done by informing the simulator when queries to the gRO relevant to the protocol he is trying to simulate are made by the environment. The concrete mechanics of how this works are not relevant for our argument, but what is important is that the above creates again an asymmetry between the power of real world adversary—who cannot know the queries to the RO made by honest parties in the protocol—and the simulator—who is the one making the queries for his simulation of the honest parties. As proved in [CJS14], by means of a smart trick, this asymmetry can be exploited to make the gRO as powerful, in terms of the feasibility statements it enables, as the local RO! For example it was proved that secure two party computation of any given function is possible assuming such a gRO.

However, we can go one step further with this and consider the following private-input/private-output n-party protocol Π_{RF} for UC emulating a random function: Upon receiving an input from the environment, a party p uses the n parties in order to evaluate a pseudo-random function (with a key which has been pre-distributed to the parties) in a private manner, e.g., by using the distributed PRF construction of Naor, Pinkas, and Reingold [NPR99]. It is straightforward to verify that if the invoked protocol securely realizes a distributed PRF, then Π_{RF} securely realizes the natural RO functionality. Hence, since gRO is weaker than the natural RO functionality (i.e., it behaves the same way but gives the simulator more power) it follows that Π_{RF} should also emulate the gRO functionality. Note however, that in Π_{RF}, the adversary is not informed about the value or the outcome of queries coming from honest parties. This means that, although Π_{RF} UC realized the gRO functionality, if we replace the gRO with Π_{RF}, the feasibility statements of [CJS14] no longer go through, as the simulator loses his advantage over the adversary.

We stress that the above is not a criticism of [CJS14], where it is clearly stated that the gRO is intended as a more functional instantiation of the random oracle heuristic[3], and it is not claimed that their results are preserved when the gRO is replaced by a protocol implementing it. Nonetheless, the above discussion does demonstrate a pitfall in what one might think is a straight-forward application of the UC subroutine replacement theorem: depending of how a global setup is defined, it is possible that feasibility statements are not preserved when the setup is replaced by a protocol which securely realizes it.

Example II: Relating assumptions and composition. The different strengths of random oracles model different assumptions. This leads to the problem of comparison and composition in a modular protocol design: assume two protocols are proven with respect to different global RO's as

[3]Indeed, since a hash function is not an implementation of the natural RO, it is unclear what one would choose to heuristically abstract it as the natural RO and not as the gRO.

above (with $\mathcal{G}_2$ exporting strictly more adversarial capabilities than $\mathcal{G}_1$): π_1 realizes $\mathcal{F}_1$ w.r.t. $\mathcal{G}_1$, and π_2, which makes (local) calls to $\mathcal{F}_1$ and realizes $\mathcal{F}_2$ w.r.t. $\mathcal{G}_2$. Therefore, it is hard to obtain a combined security claim: they assume different global setups and hence $\mathcal{F}_1$'s usage in the presence of $\mathcal{G}_2$ has never been formally realized. Therefore, applying the UCGS theorem is not possible and replacing $\mathcal{F}_1$ by π_1 must be deemed a heuristic under the global $\mathcal{G}_1$ assumption. Our main theorem implies the conditions under which π_2 can replace hybrid $\mathcal{F}_1$ by π_1 (in particular, this happens when we can replace the setup of π_2), and hence the UCGS theorem can be applied under assumption $\mathcal{G}_1$. We give more details on this for the random-oracle case in Section 4.4.

2.2 Global PKI

Is there any way we can safely replace a global setup generically by a protocol, so that the above issues, and any UC feasibility statements using the setup are not disturbed by the replacement? Canetti *et al.* studied this question in GUC framework in the context of analyzing global PKIs [CSV16]. In a nutshell, such a replacement is possible if the setup-ideal protocol ϕ (consisting of dummy parties that relay inputs and outputs to the global setup) and its realizing protocol ψ are *UC equivalent*, i.e., ψ UC emulates ϕ and ϕ emulates ψ.[4] As part of our treatment we confirm that UC equivalence is sufficient for such feasibility-preserving replacement of global setups in plain UC equipped with the UCGS theorem from [BCH$^+$20].

To gain intuition why UC equivalence is a sufficient condition, one can look at the issues with the above gRO example. As discussed, what enabled a richer feasibility landscape in the gRO model as opposed to the natural (non-programable) RO was the advantage that it gave the simulator. This advantage is lost when the gRO is replaced by Π_{RF}. If, however, instead of π_{RF} we used a protocol π which is UC emulated by the dummy gRO-hybrid protocol, denoted as ρ, then this would imply that any power that is given to the adversary/simulator when interacting with ρ would have to be also given to the simulator when interacting with π, which would, again, allow to obtain the strong feasibility statements from [CJS14].

In fact, for a natural version of a global PKI setup, [CSV16] provided a protocol using a (global) certification authority (CA) which is UC equivalent with the global PKI. However, as discussed below, for global setups which require complicated protocols to be realized, such as the ledger realized by Bitcoin, devising a simple and usuable functionality which is UC equivalent[5] with the protocol is in fact infeasible.

2.3 Global transaction ledger

The last common global setup discussed here is also the one that has recently attracted the most attention, and is a global transaction ledger. The works of Andrichowitz *et al.* [ADMM14] and Bentov and Kumaresan [BK14] demonstrated how the Bitcoin protocol can be used to achieve a compensation-based variant of fairness in multi-party computation—where the adversary forcing an unfair abort yields a penalty for the adversary which translates in compensation for the honest parties that lost their fairness. Following these results a plethora of works investigated how Bitcoin or alternative cryptocurrencies can be used by other cryptographic protocol either to circumvent limitation such as fairness, robustness, etc., or to improve their underlying properties.

[4] We note that the above is the intuition of the formal theorem of [CSV16]; as in order to even make such a statement, [CSV16] needed to define what a global protocol is in GUC.

[5] In slight abuse of terminology we will say that a functionality $\mathcal{F}$ is UC equivalent to a protocol π if the (dummy) $\mathcal{F}$-ideal protocol is UC equivalent to π.

Kiayias *et al.* [KZZ16] were the first to notice that any universally composable treatment of such constructions would yield a mismatch with reality unless the functionality that is implemented by Bitcoin—or alternative blockchain protocols—is abstracted as a *global* functionality. Indeed, not only Bitcoin should not be considered as local to any of these protocols, as in reality it pre-existed them and works independently of them, but also it is the prototypical example of what it means for all these protocols to share state, as technically they are proposed as building on top of the (same) Bitcoin network. In [KZZ16] a simple version of such a global ledger was proposed and a compiler that adds a compensation-based robustness—every honest party either terminates with his output or it gets compensated with funds that are derived by penalizing the adversasry—to any dishonest majority MPC was devised. This initiated a number of works that have modelled blockchain ledgers as global functionalities, making "globality" a central feature of the UC modeling of such ledgers.

The ledger functionality from [KZZ16] operates as follows: it receives transactions from its users or from its adversary and maintains two type of memory: the *state* which is an append only immutable memory—it corresponds to transactions that have settled in a deep enough Bitcoin block; and the *buffer* which includes recently received valid transactions that have not yet made it into the state. Every few rounds, the simulator needs to include (a subset of the transactions in) the buffer to a block which is then added to the state. Finally, upon request by any party, the ledger functionality outputs the current state to the requestor. As observed in [KZZ16], the above simplistic description could not be an abstraction of a real world execution of Bitcoin, where by using his network influence, the adversary is allowed at the very least to interfere with the order in which messages appear in any blocks added to the state. To capture such interference, the ledger from [KZZ16] allowed the simulator to reorder transactions before they are inserted to the block.

However, by the above modelling choice, the ledger of [KZZ16] introduced to the model an asymmetry: Unlike in their transaction ledger, the Bitcoin protocol does not allow the adversary to arbitrary reorder transactions that are inserted in a block—in particular for blocks created by honest parties, the adversary have very limited influence on this ordering. Similar to the RO case, by a simulator exploiting this asymmetry one might be able to prove feasibility statements that are not preserved when the ledger is replaced by the Bitcoin protocol.

We note in passing that Badertscher *et al.* [BMTZ17] observed that the ledger proposed in [KZZ16] is in fact not UC realized by (the backbone) of Bitcoin and provided an alternative, albeit more complicated version of a ledger that is (provably) realized, which give further influence to the simulator on the state update and the output of the ledger. However, as this ledger also gives the simulator (at least) the power to reorder transactions, a UC equivalence theorem between this and the Bitcoin protocol cannot hold. Despite its complex description, the ledger is far from such an equivalent functionality. This hints to the fact that any such UC equivalent to the Bitcoin ledger functionality would be as complex to use as the original protocol. This means that we cannot hope to have a usable version of a Bitcoin ledger which can be replaced by the Bitcoin protocol without destroying feasibility statements.

But perhaps more interesting and leaving Bitcoin aside, the feasibility statement of [KZZ16] does remain unaffected if we replace their proposed ledger by a ledger protocol which does not give this power to its adversary. Whether this is by chance or by design, it does raise the natural question: What is special about the compiler of [KZZ16] that makes the mismatch between the proposed ledger and its more restrictive implementation irrelevant? The short answer to this question is that the simulator used in the security proof of the compiler does not use the reordering power offered to it by the ledger, and therefore cannot fail when just this power is removed. This hints to the more relevant goal which is central in our results: to give formal conditions on global setups, protocols and security proofs that allow for replacement of the global setup by its implementation, without invalidating the security proof.

3 Preliminaries

While our ideas are formulated in a generality such that they can be applied to several composable frameworks that support global setup [KMT20, MR11], when it comes to proofs, we must fix a particular model which we choose to be UC [Can20] and its treatment of global subroutines as recently established in [BCH+20]. We provide an overview of all UC concepts in Appendix B and an overview of UCGS in Section 3.2. We give here a very brief introduction in order to follow the ideas of our proofs.

3.1 UC Basics

Formally, a protocol π is an algorithm for a distributed system and formalized as an interactive Turing machine. An ITM has several tapes, for example an identity tape (read-only), an activation tape, or input/output tapes to pass values to its program and return values back to the caller. A machine also has a backdoor tape where (especially in the case of ideal functionalities) interaction with an adversary is possible or corruption messages are handled. While an ITM is a static object, UC defines the notion of an ITM instance (denoted ITI), which is defined by the extended identity $\mathsf{eid} = (M, id)$, where M is the description of an ITM and $id = (\mathsf{sid}, \mathsf{pid})$ is a string consisting of a session identifier sid and a party identifier $\mathsf{pid} \in \mathcal{P}$. An instance, also called a session, of a protocol π (represented as an ITM M_π) with respect to a session number sid is defined as a set of ITIs $\{(M_\pi, id_{\mathsf{pid}})\}_{\mathsf{pid}\in\mathcal{P}}$ where $id_{\mathsf{pid}} = (\mathsf{sid}, \mathsf{pid})$.

The real process can now be defined by an environment $\mathcal{Z}$ (a special ITI) that spawns exactly one session of the protocol in the presence of an adversary $\mathcal{A}$ (also a special ITI), where $\mathcal{A}$ is allowed to interact with the ITIs via the *backdoor tape*, e.g., to corrupt them or to obtain information from the hybrid functionalities, e.g. authenticated channels, that the protocol is using. The adversary ITI can only communicate with the backdoor tapes of the protocol machines. An environment can be restricted by a so-called identity bound $\xi \in \Xi$ which formalizes which external parties the environment might claim when interacting as input provider to the protocol. The less restrictive the bound, the more general the composition guarantees are. The UC theorem is quantified by such a predicate.

The output of the execution is the bit output by $\mathcal{Z}$ and is denoted by $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, r)$ where k is the security parameter, $z \in \{0,1\}^*$ is the input to the environment, and randomness r for the entire experiment. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, r)$. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal-world process The ideal process is formulated with respect to an another protocol ϕ, which in its most familiar form is a protocol $\text{IDEAL}_{\mathcal{F}}$ for an ITM $\mathcal{F}$ which is called an ideal functionality for which we describe the situation. In the ideal process, the environment $\mathcal{Z}$ interacts with $\mathcal{F}$, an ideal-world adversary (often called the simulator) $\mathcal{S}$ and a set of trivial, i.e., dummy ITMs representing the protocol machines of $\text{IDEAL}_{\mathcal{F}}$ that forward to the functionality whatever is provided as inputs to them by the environment (and return back whatever received from the functionality). In the ideal world, the ideal-world adversary (aka the simulator) can decide to corrupt parties and can interact via the backdoor tape with the functionality. For example, via the backdoor tape, the functionality could for example leak certain values about the inputs, or allow certain influence on the system. We denote the output of this ideal-world process by $\text{EXEC}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k, z, r)$ where the inputs are as in the real-world process. Let $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ denote the random variable

obtained by choosing the randomness r uniformly at random and evaluating $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z,r)$. Let $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

Secure Realization and Composition In a nutshell, a protocol π ξ-UC-emulates (ideal) protocol ϕ if the "real-world" process (where π is executed) is indistinguishable from the ideal-world process (where ϕ is executed), i.e., if for any (efficient) adversary $\mathcal{A}$ there exists an (efficient) ideal-world adversary (the simulator) $\mathcal{S}$ such that for every (efficient) ξ-bounded environment $\mathcal{Z}$ it holds that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$.

The emulation notion is composable, i.e., if π UC-emulates ϕ, then in a larger context protocol ρ, the subroutine ϕ can be safely replaced by π, denoted by $\rho^{\phi\to\pi}$. For this replacement to be well-defined, a few technical preconditions must be satisfied. First, the protocols must be compliant, which ensures that in case π and ϕ might both be subroutines in ρ they do not share the same session (ensuring that the replacement operator works as intended). Furthermore, compliance also makes sure that the protocol is invoked properly, i.e., with the correct identities specified in ξ. The full definition is found in Appendix B. The second major precondition is that protocols should be subroutine respecting, meaning that each session of π can run in parallel with other sessions of protocols without interfering with them (in order for the UC-emulation notion which considers a single challenge session to be a reasonable precondition for the composition theorem). For details we refer to Appendix B.

Theorem 3.1 (UC Theorem)**.** *Let ρ, π, ϕ be protocols and let ξ be a predicate on extended identities, such that ρ is (π,ϕ,ξ)-compliant, both ϕ and π are subroutine exposing and subroutine respecting, and π UC-emulates ϕ with respect to ξ-identity-bounded environments. Then $\rho^{\phi\to\pi}$ UC-emulates protocol ρ.*

3.2 UC with Global Subroutines

A global subroutine can be imagined as a module that a protocol uses as a subroutine, but which might be available to more than this protocol only. While initial formalizations to capture when a module is available to everyone, i.e., to the environment, defined a UC-variant [CDPW07], it was recently shown that capturing this can be fully accommodated within UC [BCH+20]. In a nutshell, if π is proven to realize ϕ in the presence of a global subroutine γ, then the environment can access this subroutine in both, the ideal and the real world, which must be taken care of by the protocol. As a rule of thumb, proving that π realizes ϕ in the presence of global γ is harder than when γ is a local subroutine, i.e., not visible by the environment.

The framework presented in [BCH+20] defines a new UC-protocol $\mathsf{M}[\pi,\gamma]$ that is an execution enclave of π and γ. $\mathsf{M}[\pi,\gamma]$ provides the environment access to the main parties of π and γ in a way that does not change the behavior of the protocol or the set of machines. The clue is that $\mathsf{M}[\pi,\gamma]$ itself is a normal UC protocol and the emulation is perfect under certain conditions on π and γ. We first state the definition from [BCH+20].

Definition 3.2 (UC emulation with global subroutines)**.** Let π, ϕ and γ be protocols. We say that π *ξ-UC-emulates ϕ in the presence of γ* if protocol $\mathsf{M}[\pi,\gamma]$ ξ-UC-emulates protocol $\mathsf{M}[\phi,\gamma]$.

The first condition is the following and expresses the fact that γ might communicate with protocols outside of π's realm:

Definition 3.3 (γ-subroutine respecting)**.** A protocol π is called γ-subroutine respecting if the four conditions of Definition B.6 required from any (sub-)party of some instance of π are relaxed as follows:

- the conditions do not apply to those sub-parties of instance s that also belong to some extended session s' of protocol γ;
- (sub-)parties of s may pass input to machines that belong to some extended session s' of protocol γ, even if those machines are not yet part of the extended instance of s (cf. Definition B.6, item 4).

The second condition is a technical condition on the global subroutine which is called *regularity*. The condition says that (a) a shared subroutine does not spawn new ITIs by providing subroutine output to them, and (b) that the shared subroutine may not invoke the outside protocol as a subroutine. It is usually not a problem for global setups to satisfy this, since most of the time, we can model functionalities to be reactive and assume "signaling events" to happen out-of-band.

The formal definition is taken from [BCH+20].

Definition 3.4 (Regular setup)**.** Let ϕ, γ be protocols. We say that γ is a *ϕ-regular setup* if, in any execution, the main parties of an instance of γ do not invoke a new ITI of ϕ via a message destined for the subroutine output tape, and do not have an ITI with code ϕ as subsidiary.

In [BCH+20, Proposition 3.5], the authors show that if the protocol π is γ-subroutine respecting, where γ itself is π-regular and subroutine respecting, then the interaction between π and the global subroutine γ is very structured without unexpected artifacts. We state the proposition here for completeness. Here, α is an arbitrary protocol and $\widehat{\alpha}$ is a version of α that makes use of $\mathsf{M}[[]\pi, \gamma]$ instead of π and has an indistinguishable behavior. We refer to [BCH+20] and just state the proposition.

Proposition 3.5. *Let γ be subroutine respecting and π be γ-subroutine respecting. Then the protocol $\mathsf{M}[\pi, \gamma]$ is subroutine respecting. In addition, let γ be π-regular, and let α be a protocol that invokes at most one subroutine with code π. Denote by $\widehat{\alpha}$ the transformed protocol described above. Then the transcript established by the set of virtual ITIs in an execution of some environment with $\widehat{\alpha}$ is identical to the transcript established by the set of ITIs induced by the environment that has the same random tape but interacts with α.*

The UCGS theorem is then the composition theorem for protocols that are defined with respect to a global subroutine γ. Note that not γ is replaced, but ϕ by its implementation π.

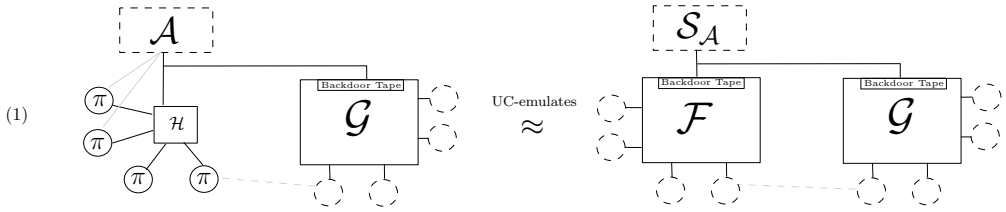
Theorem 3.6 (Universal Composition with Global Subroutines – UCGS Theorem)**.** *Let ρ, ϕ, π, γ be subroutine-exposing protocols, where γ is a ϕ-regular setup and subroutine respecting, ϕ, π are γ-subroutine respecting and ρ is (π, ϕ, ξ)-compliant and $(\pi, \mathsf{M}[\mathsf{code}, \gamma], \xi)$-compliant for $\mathsf{code} \in \{\phi, \pi\}$. Assume π ξ-UC-emulates ϕ in the presence of γ, then $\rho^{\phi \to \pi}$ UC-emulates ρ.*

4 Replacement Theorems for a Global Subroutine

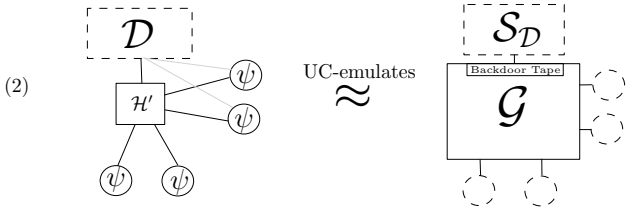
In this section, we consider a setting where protocols access only one global subroutine, e.g., a global CRS, or a global ledger, but not both of them. That is, we only consider protocols whose shared setup is formulated as a single protocol. For this simplest global setting, we start by exploring which replacement of the global subroutine follows already from application of the UC composition theorem. Then, we recover the replacement theorem of [CSV16], which preserves security statements if the global subroutine is replaced by an equivalent protocol. And finally, we give conditions for security-preserving replacement of non-equivalent global subroutines.

4.1 Common Preconditions of our Theorems

Throughout this section, we assume the following preconditions for our theorems. Recall that we are interested in replacing a global subroutine while preserving security statements made with respect to this subroutine. We assume the security statement to be the following: protocol π UC-emulates an ideal functionality $\mathcal{F}$ in the presence of global subroutine $\mathcal{G}$, with respect to dummy adversary $\mathcal{A}$. Simulator $\mathcal{S}_{\mathcal{A}}$ is a witness for this emulation. The statement is depicted below and referred to in the text as precondition (1).



Second, since our aim is to investigate how UC emulation of global subroutines can be useful for context protocols, we assume that the global subroutine is emulated as follows: ψ UC-emulates $\mathcal{G}$, with simulator $\mathcal{S}_{\mathcal{D}}$ for dummy adversary $\mathcal{D}$. We refer to this emulation as precondition (2).



Given this notation, the core question of our work can be stated as follows: given preconditions (1) and (2), under which additional conditions does it hold that

$$\pi \text{ UC-emulates } \mathcal{F} \text{ in the presence of global } \psi?$$

Simplifying notation. We note that, while our theorems hold for arbitrary UC protocols, to ease understanding, we formulate them with the special protocols $\text{IDEAL}_{\mathcal{F}}$ and $\text{IDEAL}_{\mathcal{G}}$. Intuitively, $\mathcal{F}$ is a "target" functionality that is to be realized and $\mathcal{G}$ a global ideal setup. To further simplify, we slightly abuse notation and write $\mathcal{G}$ instead of $\text{IDEAL}_{\mathcal{G}}$, e.g., we write "$\psi$ UC-emulates $\mathcal{G}$" instead of "ψ UC-emulates $\text{IDEAL}_{\mathcal{G}}$".

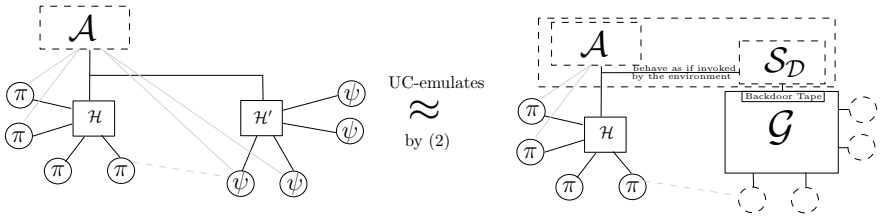
4.2 Warm-Up: Replacing Real-World Global Setups

Our first lemma states that under precondition (2) we can replace the shared subroutine by the construction that UC emulates it. Another way to view this is that "lifting" to global subroutines (w.r.t any application protocol π) preserves UC emulation. An important feature of this statement is that it follows from standard UC composition thanks to the embedding of global setups in standard UC. Throughout the section, we will maintain a running example to illustrate all our statements.

Running Example. Let $\mathcal{G} = \mathcal{G}_{\mathsf{ledger}}$ be an ideal ledger and π a lottery protocol requiring a ledger. Further, let $\psi =$ FunCoin be a cryptocurrency implementing the ledger $\mathcal{G}_{\mathsf{ledger}}$. By UC emulation, all manipulation and attacks allowed on FunCoin must also be allowed against $\mathcal{G}_{\mathsf{ledger}}$. In particular, this holds for any manipulation or attack carried out while running a lottery.

Lemma 4.1. *Assume a protocol π makes subroutine calls to global subroutine $\mathcal{G}$ and that ψ is a protocol that UC-emulates $\mathcal{G}$. Then π invoking ψ instead of $\mathcal{G}$ UC-emulates protocol π.*

Proof. On a high level, the argument is as follows: if an environment could tell a run of π with ψ from a run of π with $\mathcal{G}$, then running π internally would already let the environment distinguish a run of ψ from a run of $\mathcal{G}$, violating the precondition of the lemma.



Since for the technical argument, we have to stick to a particular model, we have use the UC language in a more precise way: hence, we assume that π, ψ, $\mathcal{G}$ be protocols and let $\xi \in \Xi$ be a predicate on extended identities, such that π is $(\psi, \mathcal{G}, \xi)$-compliant, π, $\mathcal{G}$, ψ are subroutine exposing, $\mathcal{G}$ and ψ are subroutine respecting, and π is subroutine respecting except via calls to $\mathcal{G}$. We note that these technical conditions are as they appear in UC in order to guarantee that the UC-operator is well defined. To formalize emulation in the presence of a shared setup, we use the terminology of UCGS [BCH+20] (see Section 3.2 for a short recap), where global access to $\mathcal{G}$ is granted by an overlay $\mathsf{M}[\cdot, \mathcal{G}]$. In order for this overlay to be opaque to the execution of π with $\mathcal{G}$, we need to assume $\mathcal{G}$ to be π-regular (see Definition 3.4 and Proposition 3.5).

With this terminology, it remains to show that if ψ UC-emulates $\mathcal{G}$ with respect to ξ-identity-bounded environments, then $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates protocol $\mathsf{M}[\pi, \mathcal{G}]$ (with respect to ξ-identity bounded environments). This follows from the UC composition theorem: First, observe that $\mathsf{M}[\pi, \mathcal{G}]$ is an ordinary UC-protocol, mimicking all effects that the global (and hence shared) subroutine might have with the environment. Similarly, $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ is an ordinary UC-protocol where subroutine $\mathcal{G}$ is replaced by ψ. Note that, similar to the role of the control function in UC, the embedding $\mathsf{M}[\cdot]$ does not reveal the code of the main instances when interacting with the environment, and therefore we have that $\mathsf{M}[\pi, \mathcal{G}]^{\mathcal{G}\to\psi}$ and $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ are equivalent protocols. Since ψ UC-emulates $\mathcal{G}$ w.r.t. all environments that are bounded by ξ, the UC composition theorem implies that $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates $\mathsf{M}[\pi, \mathcal{G}]$. □

Lemma 4.1 will serve mainly as a tool in proving the upcoming theorems. Next, we can apply the UC composition theorem to our two preconditions. This yields the following theorem. It says that, in any UC emulation statement w.r.t a global setup, we can safely strengthen the *real-world* setup, while leaving the setup in the ideal world unchanged. The intuition behind it is illustrated with the following example.

Running Example. Back to our lottery. The lottery's provider wants to create trust in his product. He therefore proves that, when run with the global ideal ledger, the lottery protocol UC-emulates some ideal functionality $\mathcal{F}_{\mathsf{lottery}}$ which enforces a fair lottery. In his proof, both the lottery protocol and $\mathcal{F}_{\mathsf{lottery}}$ may exploit weaknesses of $\mathcal{G}_{\mathsf{ledger}}$. Since FunCoin is at least as secure as $\mathcal{G}_{\mathsf{ledger}}$, the provider can safely advertise that running the lottery with FunCoin is as secure as $\mathcal{F}_{\mathsf{lottery}}$ with $\mathcal{G}_{\mathsf{ledger}}$, since this replacement can only decrease the number of possible attacks on the global setup while running the lottery.

Lemma 4.2. *Assume a protocol π UC-emulates $\mathcal{F}$ in the presence of global subroutine $\mathcal{G}$ and that $\mathcal{G}$ is UC-emulated by ψ, then replacing π's subroutine $\mathcal{G}$ by ψ UC-emulates $\mathcal{F}$ that has access to global subroutine $\mathcal{G}$.*

Proof. We again need some technical conditions from standard UC and UCGS: Let π, $\mathcal{F}$, ψ, $\mathcal{G}$ be protocols and let ξ, $\xi' \in \Xi$ be predicates on extended identities, such that π is $(\psi, \mathcal{G}, \xi)$-compliant, π, $\mathcal{F}$, $\mathcal{G}$, ψ are subroutine exposing, $\mathcal{G}$ and ψ are subroutine respecting, π and $\mathcal{F}$ are subroutine respecting except via calls to $\mathcal{G}$ and $\mathcal{G}$ is π-regular. If ψ UC-emulates $\mathcal{G}$ with respect to ξ-identity-bounded environments, and if π UC-emulates $\mathcal{F}$ in the presence of $\mathcal{G}$ w.r.t. ξ'-identity-bounded environments, then what we have to prove is that $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates protocol $\mathsf{M}[\mathcal{F}, \mathcal{G}]$ w.r.t. ξ'-identity-bounded environments. This however follows from standard composition: Recall that protocols $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ and $\mathsf{M}[\pi, \mathcal{G}]$ from Lemma 4.1 are embeddings of protocols with global setup as normal UC protocols. Therefore, we can apply the UC composition theorem: $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates $\mathsf{M}[\pi, \mathcal{G}]$, and by our assumption $\mathsf{M}[\pi, \mathcal{G}]$ UC-emulates $\mathsf{M}[\mathcal{F}, \mathcal{G}]$. □

The conclusion of this subsection is that under both conditions (1) and (2) it follows that both π and ψ running together are indistinguishable from the ideal world, where both components are idealized. This is often assurance enough that the protocol in combination with a particular implementation of the global setup achieves a good level of security. However, note that the security is stated in terms of abstractions of *both* real-world components. The overall guarantees are thus hard to tell, and false impressions of security might be created. Let us illustrate this issue with the following.

Running Example. Assume that the provider does not have a strong cryptographic background and that he actually struggled conducting the aforementioned proof. But suddenly, he realized that the proof is easy when he assumes that $\mathcal{G}_{\mathsf{ledger}}$, which is used by both the poker game and $\mathcal{F}_{\mathsf{lottery}}$, admits arbitrarily many adversarial ledger entries. He calls this new setup $\mathcal{G}_{\mathsf{weakLedger}}$ and is delighted when he finds out that it is still emulated by FunCoin (since UC emulation is transitive). He then happily applies Lemma 4.2 and rightfully advertises that his poker game run with FunCoin is as secure as $\mathcal{F}_{\mathsf{lottery}}$ together with $\mathcal{G}_{\mathsf{weakLedger}}$.

With this example we see that Lemma 4.2 falls short in examining the security of the challenge protocol when proven w.r.t. an (even slight) abstraction of the setup and not its implementation. In the above example, $\mathcal{F}_{\mathsf{lottery}}$ might provide very strong fairness guarantees, that however can only be achieved with a simulation that crucially exploits introduction of adversarial entries into $\mathcal{G}_{\mathsf{weakLedger}}$. Thus, when looking only at $\mathcal{F}_{\mathsf{lottery}}$, false impressions of security guarantees are created. In particular, with the stronger global $\mathcal{G}_{\mathsf{ledger}}$ or the actual protocol FunCoin, which do not have this weakness, $\mathcal{F}_{\mathsf{lottery}}$ might not even be realizable by the lottery – to say the least, the existing simulation is likely to fall short in witnessing such an emulation statement.

To remedy the situation (and to blow our provider's cover), we need to understand the implications of replacing the global setup in the ideal world. In particular, preventing a security proof from exploiting weaknesses in the abstraction of the setup seems to be crucial to arrive at a plausible and realistic level of security. In the remainder of this section, we ask under which conditions a security proof might be preserved when replacing the global setup in both worlds.

4.3 Full Replacement of the Global Subroutine

We now turn our attention to "full" replacement strategies, where the global subroutine is replaced by a protocol UC-emulating it in both the real and the ideal world. Of course, this is to be understood w.r.t an existing security statement, that is, our precondition (1). Let us emphasize again that we are only interested in replacement strategies that preserve the underlying security statement.

4.3.1 Equivalence Transformations of the Global Subroutine.

Canetti et al. demonstrated, using the terminology of GUC, that replacing the global subroutine by an equivalent procedure preserves protocol emulation w.r.t the subroutine. The replacement theorem is proven in [CSV16], and we recover it here for completeness. Thanks to the embedding within plain UC that UCGS achieves, our proof is able to capture the arguments at a more abstract level, essentially reducing all steps to standard UC-emulation. Let us first illustrate how and why equivalence replacement works with the lottery.

Running Example. The provider keeps receiving calls from cryptographers who find it suspicious that his simulation exploits the weaknesses of $\mathcal{G}_{\mathsf{weakLedger}}$. Since FunCoin does not offer introduction of arbitrary adversarial blocks, the provider however cannot carry out his simulation with FunCoin. Searching the internet, the provider learns about a shady cryptocurrency called DarkCoin. Further investigating, the provider can prove that DarkCoin admits the exact same attacks as $\mathcal{G}_{\mathsf{weakLedger}}$, i.e., is UC-equivalent to $\mathcal{G}_{\mathsf{weakLedger}}$[6]. Thus, the provider can run his simulation with DarkCoin instead of $\mathcal{G}_{\mathsf{weakLedger}}$, since DarkCoin allows for all adversarial queries that are possible with $\mathcal{G}_{\mathsf{weakLedger}}$. Moreover, the provider can be assured that his simulation is still good for the now modified real world, since DarkCoin does not admit more attacks than $\mathcal{G}_{\mathsf{weakLedger}}$. Relieved, he announces that, when using the globally available DarkCoin, his lottery protocol emulates $\mathcal{F}_{\mathsf{lottery}}$.

Theorem 4.3 (Full Replacement via Equivalence Transformations)**.** *Assume π UC-emulates $\mathcal{F}$ in the presence of a global subroutine $\mathcal{G}$. If ψ UC-emulates $\mathcal{G}$ and vice-versa, i.e., their adversarial interfaces are equivalent, then π, invoking ψ instead of $\mathcal{G}$, UC-emulates $\mathcal{F}$, invoking ψ instead of $\mathcal{G}$, and where ψ is the global subroutine.*

Proof. We again have to phrase our theorem in the language of UCGS: Let π, $\mathcal{F}$, ψ, $\mathcal{G}$ be protocols and let $\xi, \xi' \in \Xi$ be predicates on extended identities, such that π is $(\psi, \mathcal{G}, \xi)$-compliant, π, $\mathcal{F}$, $\mathcal{G}$, ψ are subroutine exposing, $\mathcal{G}$ and ψ are subroutine respecting, π and $\mathcal{F}$ are subroutine respecting except via calls to $\mathcal{G}$ and $\mathcal{G}$ is π-regular. If ψ UC-emulates $\mathcal{G}$ with respect to ξ-identity-bounded environments — and vice-versa— and if π UC-emulates $\mathcal{F}$ in the presence of $\mathcal{G}$ w.r.t. ξ'-identity-bounded environments, then $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates protocol $\mathsf{M}[\mathcal{F}^{\mathcal{G}\to\psi}, \psi]$ w.r.t. ξ'-identity-bounded environments.

The sequence of steps needed in this proof are the following hybrid protocols.

- The real protocol $H_0 := \mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$.
- The first intermediate step $H_1 := \mathsf{M}[\pi, \mathcal{G}]$.
- The second intermediate step $H_2 := \mathsf{M}[\mathcal{F}, \mathcal{G}]$.
- The destination protocol $H_3 := \mathsf{M}[\mathcal{F}^{\mathcal{G}\to\psi}, \psi]$.

As in the proof of Lemma 4.1 , H_0 is equivalent to $\mathsf{M}[\pi, \mathcal{G}]^{\mathcal{G}\to\psi}$ and hence $H_1 = H_0^{\psi\to\mathcal{G}}$. By standard composition, H_0 UC-emulates H_1 since the embedding is an normal UC-protocol and subroutine ψ UC-emulates $\mathcal{G}$. Next, the transition from H_1 to H_2 is trivial: H_1 UC-emulates H_2 by the theorem assumption. Finally, we go the "reverse" direction as in the argument of the first step thanks to the fact that we know that $\mathcal{G}$ UC-emulates ψ. More formally, we have $H_3 = \mathsf{M}[\mathcal{F}, \mathcal{G}]^{\mathcal{G}\to\psi}$ and again, H_3 is obtained by normal subroutine replacement within protocol H_2. Therefore, H_2 UC-emulates H_3 by the theorem assumption and we have that H_0 UC-emulates H_3 which concludes the proof. □

[6]Formally, ψ and ψ' are UC-equivalent if ψ UC-emulates ψ' and ψ' UC-emulates ψ.

To the best of our knowledge, Theorem 4.3 is the only composition theorem allowing for replacement of global subroutines with their UC emulation that already exists in the literature [CSV16]. It can be applied to soundly replace, e.g., a globally available ideal PKI with its implementation at a trusted PKI provider. However, it falls short in replacing global setups with protocols, which are likely to be stronger than their abstraction as a UC functionality. In the remainder of this section we discuss solutions for such replacements.

4.3.2 Global-agnostic Simulations of the Challenge Protocol.

The condition discussed in this section is useful for protocols designers to check whether their proof remains valid when a global subroutine is replaced, by means of checking the structure of the simulator. Intuitively, a sufficient condition is if the simulator can simulate without accessing the adversarial interface of the global setup. More generally speaking, for all UC-adversaries $\mathcal{A}$ the corresponding simulation strategy $\mathcal{S}_{\mathcal{A}}$ should only externally-write onto the backdoor tape of the global subroutine session(s) if the real-world adversary did so. An easy way to achieve this is to have the ideal functionality $\mathcal{F}$ communicate with the global setup $\mathcal{G}$ directly and if needed, provide the simulator (simulating the actions of π when having access to the backdoor tape of $\mathcal{F}$) with the necessary information. Intuitively, the reason this is sound is that the only way $\mathcal{F}$ can interact with the global setup just like an honest party would do (and in particular, not via the backdoor tape). Since replacing $\mathcal{F}$ by a protocol that implements it can never change the behavior for honest parties in a noticeable way (otherwise, it is obviously distinguishable) the replacement is unproblematic. We first formally capture what it means for a simulator to not use the adversarial interface of the global subroutine.

Definition 4.4 ($\mathcal{G}$-agnostic)**.** An adversary $\mathcal{S}$ interacting with subroutine $\mathcal{G}$ is *$\mathcal{G}$-agnostic* if the only external write requests (made by $\mathcal{S}$'s shell) destined for (the backdoor tape) of parties and subparties of any session of $\mathcal{G}$ are those instructed by the environment directly and any messages via the backdoor tapes of (sub-)parties of any session of $\mathcal{G}$ are delivered directly to the environment without activating the body of $\mathcal{S}$.

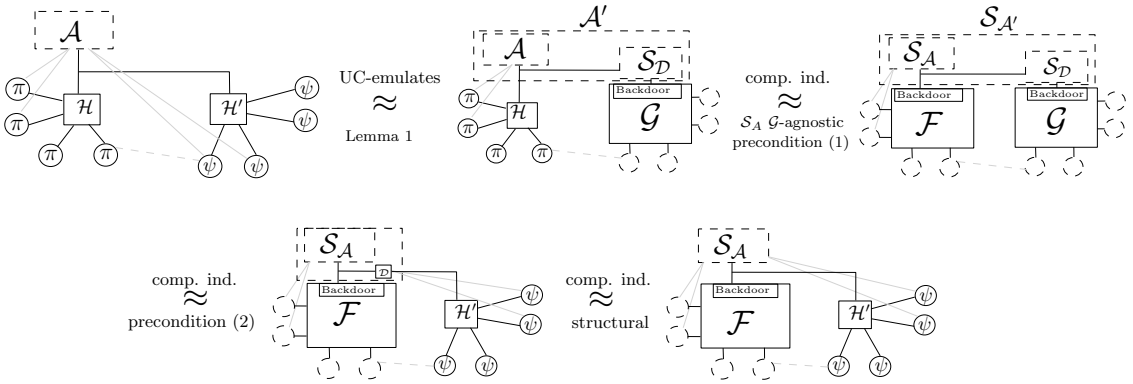
Running Example. Recently, numbers of users participating in the provider's lottery dropped significantly. Being sure that this is because of his recent recommendation to use DarkCoin, the provider desperately hires a team of cryptographers. Examining the provider's simulation carried out with respect to $\mathcal{G}_{\mathsf{weakLedger}}$, the team finds a better simulation strategy that only requires legitimate use of the ledger by sending transaction requests to it. The new simulator thus acts like an honest party using the ledger. In particular it does not exploit any of the adversarial interfaces of $\mathcal{G}_{\mathsf{weakLedger}}$. Since FunCoin allows to submit transactions, replacing $\mathcal{G}_{\mathsf{weakLedger}}$ by FunCoin in the proof does not hinder the new simulation. With FunCoin back in the picture, user statistics begin to slowly recover and the provider is delighted.

Theorem 4.5 (Full Replacement due to Agnostic Simulations I)**.** *Assume π UC-emulates $\mathcal{F}$ in the presence of a global subroutine $\mathcal{G}$ such that the simulator $\mathcal{S}$ for this construction is $\mathcal{G}$-agnostic. Let further ψ UC-emulate $\mathcal{G}$. Then π, invoking ψ instead of $\mathcal{G}$, UC-emulates $\mathcal{F}$, invoking ψ instead of $\mathcal{G}$, and where ψ is the global subroutine.*

Proof. We first state the theorem in the language of UCGS as before. Let π, $\mathcal{F}$, ψ, $\mathcal{G}$ be protocols and let ξ, $\xi' \in \Xi$ be predicates on extended identities, such that π is $(\psi, \mathcal{G}, \xi)$-compliant, π, $\mathcal{F}$, $\mathcal{G}$, ψ are subroutine exposing, $\mathcal{G}$ and ψ are subroutine respecting, π and $\mathcal{F}$ are subroutine respecting except via calls to $\mathcal{G}$ and $\mathcal{G}$ is π-regular. Let ψ UC-emulate $\mathcal{G}$ with respect to ξ-identity-bounded

environments and let π UC-emulate $\mathcal{F}$ in the presence of $\mathcal{G}$ w.r.t. ξ'-identity-bounded environments. Let $\mathcal{S}_\mathcal{A}$ denote a simulator for the latter emulation that satisfies Definition 4.4. Then $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates protocol $\mathsf{M}[\mathcal{F}^{\mathcal{G}\to\psi}, \psi]$ w.r.t. ξ'-identity-bounded environments.

The proof strategy is as follows:



More formally, we make the following transitions, going from top left to bottom right in the picture.

$\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ **UC-emulates** $\mathsf{M}[\pi, \mathcal{G}]$**.** This directly follows from Lemma 4.1 and the precondition of ψ UC-emulating $\mathcal{G}$. Let $\mathcal{A}'$ denote the simulator of this emulation.

$\text{EXEC}_{\pi,\mathcal{A}',\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S}_{\mathcal{A}'},\mathcal{Z}}$**.** We show how to simulate for the specific adversary $\mathcal{A}'$. $\mathcal{S}_{\mathcal{A}'}$ works as $\mathcal{S}_\mathcal{A}$, but lets the internally simulated $\mathcal{A}$ on π issue its external write requests to the global subroutine directly to $\mathcal{S}_\mathcal{D}$, which overall has the effect as if $\mathcal{A}$ and $\mathcal{S}_\mathcal{D}$ were combined when talking to the global subroutine. The simulator $\mathcal{S}_\mathcal{A}$ (simulating π while interacting with $\mathcal{F}$) performs a good simulation even against this combined attacker, because $\mathcal{S}_\mathcal{A}$ does not care about this interaction due to the agnostic property: $\mathcal{S}_\mathcal{A}$ does not issue any queries to $\mathcal{G}$ itself (that might get blocked or modified by $\mathcal{S}_\mathcal{D}$) and acts as a relay between $\mathcal{G}$ and $\mathcal{Z}$. Assume $\mathcal{Z}$ distinguishes both distributions. Then, $\mathcal{Z}$ running $\mathcal{S}_\mathcal{D}$ internally instead of sending requests to $\mathcal{S}_\mathcal{D}$ to the adversary is a successful distinguisher of $\pi, \mathcal{A}$ and $\mathcal{F}, \mathcal{S}_\mathcal{A}$, since due to $\mathcal{S}_\mathcal{A}$ being $\mathcal{G}$-agnostic, $\mathcal{Z}$ is oblivious of the order of $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_\mathcal{D}$ (and, naturally, of the order of $\mathcal{A}$ and $\mathcal{S}_\mathcal{D}$). Since such a $\mathcal{Z}$ would violate precondition (1), we conclude that both distributions are indistinguishable.

$\text{EXEC}_{\mathcal{F},\mathcal{S}_{\mathcal{A}'},\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}}$**,** where $\mathcal{S}'$ denotes the simulator $\mathcal{S}_\mathcal{A}$ sending requests to ψ via dummy adversary $\mathcal{D}$. Recall that $\mathcal{S}_{\mathcal{A}'}$ combines $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_\mathcal{D}$. If both executions are distinguishable, an environment running $\mathcal{S}_\mathcal{A}$ and $\mathcal{F}$ could distinguish an execution of ψ and $\mathcal{D}$ from an execution of $\mathcal{S}_\mathcal{D}$ with $\mathcal{G}$, violating the precondition that ψ UC-emulates $\mathcal{G}$, i.e., precondition (2).

$\text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S}_\mathcal{A},\mathcal{Z}}$**.** Since the dummy adversary $\mathcal{D}$ is just a relay, we can safely remove it from the execution.

□

4.3.3 General Condition for Global-Functionality Replacement.

With the previous theorem, we showed that a global subroutine can be safely replaced by its emulation in all security statements which are proven via a simulator who does not access the global subroutine. This however not only means that the simulator cannot manipulate the state of the

global setup, but is also completely oblivious of it. This is often too strong of a condition. For example, consider a simulator witnessing a protocol's security in the presence of a global CRS. Such a simulator should at least be allowed to *read* out the CRS, since, intuitively, the CRS is publicly available information. Similarly, a simulator in a global ledger world should at least be allowed to read the state of the ledger. And indeed, our next replacement theorem admits global replacements that do not interfere with such simulators, as long as the power of the simulator is reflected in the real world even with respect to the stronger emulation of the global subroutine.

To ease the technical presentation of the condition on the simulator, for the next theorem we restrict ourselves to the special case of functionalities as global subroutines. The treatment could be generalized to arbitrary global subroutines. Let us start with introducing some technical tools which help us formalize interaction between adversaries and global functionalities.

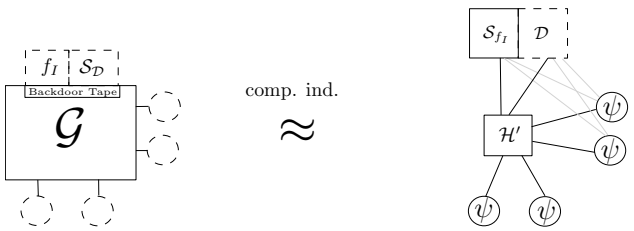
Definition 4.6 (Ordered interaction)**.** Let I be a set of queries. An ideal functionality $\mathcal{G}$ is called *I-ordered* if G answers to inputs $x \in I$ on the backdoor tape with (x, y), and uses format $(\perp, \cdot)$ otherwise.

The definition simply demands that ITM $\mathcal{G}$, in his answers to the adversary, repeats what query it responds to if the query belongs to some set I. Note that quite often in the literature, such an association is necessary but left implicit in the description, since it is obvious which query will result in which answer (by repeating the input and maintaining a clear order when answering adversarial requests). Next, we define some useful notation when running two programs in one machine. Essentially, we define a wrapper that routes incoming queries to the program which they are intended for.

Definition 4.7 (Parallel composition of adversaries)**.** Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two ITMs. Then $[\mathcal{S}_1, \mathcal{S}_2]$ denotes the adversary with the following shell: whenever activated with value (x, y) on the backdoor tape, it activates $\mathcal{S}_i$ if x was issued by $\mathcal{S}_i$ and in any other case activates $\mathcal{S}_2$ by default. Conversely, if activated with input (i, x) on the input tape (for any x), the shell activates $\mathcal{S}_i$ on input x.

Definition 4.8 (Admissible backdoor-tape filter)**.** Let $\mathcal{S}_\mathcal{D}$ be the simulator of condition (2), i.e., the construction of $\mathcal{G}$ from ψ. Let I be a subset of adversarial queries allowed by $\mathcal{G}$, and let $\mathcal{G}$ be I-ordered. Let further f_I denote a simple program which takes inputs $x \in I$ and writes them on the backdoor tape of $\mathcal{G}$, and if provided with input (x, y) on the backdoor tape, returns y to the caller that provided the corresponding input x (other values on the subroutine output tape are ignored by f). We say that f_I is an *admissible backdoor-tape filter for* $(\mathcal{S}_\mathcal{D}, \psi, \mathcal{G})$ if there exists a simulator $[\mathcal{S}_{f_I}, \mathcal{D}]$ such that $\text{EXEC}_{\mathcal{G},[f_I,\mathcal{S}_\mathcal{D}],\mathcal{Z}} \approx \text{EXEC}_{\psi,[\mathcal{S}_{f_I},\mathcal{D}],\mathcal{Z}}$. We omit $(\mathcal{S}_\mathcal{D}, \psi, \mathcal{G})$ if it is clear from the context.

Pictorially, f_I is an admissible filter if there is a simulator $\mathcal{S}_{f_I}$ such that:



Note that a filter is nothing else than a program making the adversarial interface of $\mathcal{G}$ less powerful while not interfering with the assumed simulator.

Running Example. Let us assume that the ledger $\mathcal{G}_{\mathsf{weakLedger}}$ has adversarial interfaces $J := \{\texttt{readState}, \texttt{permute}, \texttt{putEntry}\}$. DarkCoin UC-emulates $\mathcal{G}_{\mathsf{weakLedger}}$ with simulator $\mathcal{S}_{\mathcal{D}}$ that, say, only uses interface `putEntry`. Thus, $f_{\{\texttt{readState}\}}$ is admissible for $(\mathcal{S}_{\mathcal{D}}, \text{DarkCoin}, \mathcal{G}_{\mathsf{weakLedger}})$ since $\mathcal{S}_{\mathcal{D}}$ does not depend on how often $\mathcal{G}_{\mathsf{weakLedger}}$ outputs the state. The simulator $\mathcal{S}_{f_{\{\texttt{readState}\}}}$ simply collects the state of the DarkCoin ledger from publicly available information. On the other hand, $f_{\{\texttt{permute}\}}$ (and $f_{\{\texttt{permute},\texttt{readState}\}}$) can only be admissible if $\mathcal{S}_{\mathcal{D}}$ performs a good simulation regardless of the order in which entries (including adversarial ones) appear on the ledger, and if there exists an attacker $\mathcal{S}_{f_{\{\texttt{permute}\}}}$ that can carry out a permuting attack against DarkCoin.

The next definition restricts the simulator's usage of the global functionality. Essentially, the simulator is not allowed to query the global $\mathcal{G}$ except for queries in some set I.

Definition 4.9 ($\mathcal{G}\backslash I$-agnostic)**.** Let $\mathcal{S}$ denote an adversary interacting with global subroutine $\mathcal{G}$ and let I denote a subset of the adversarial queries allowed by $\mathcal{G}$, and let $\mathcal{G}$ be I-ordered. $\mathcal{S}$ is called *$\mathcal{G}\backslash I$-agnostic* if the only external write requests (made by the simulator's shell) destined for $\mathcal{G}$ are either requests $x \in I$ or those instructed by the environment directly, and any messages via the backdoor tapes from the (sub-)parties of $\mathcal{G}$ are delivered directly to the environment without activating the body of $\mathcal{S}$, except when they are of the form $(x, \cdot)$ where the query $x \in I$ has been issued by the body of $\mathcal{S}$.

We are now ready to state our most general replacement theorem for global subroutines for simulators that are global-agnostic except for queries in some set I that pass the backdoor-tape filter of the shared subroutine. Those queries can be asked by the simulator any time. The intuition is that, due to the admissible property, we know how to "attack" an instantiation of $\mathcal{G}$ to extract information from it that is indistinguishable from what the filtered adversarial interface of $\mathcal{G}$ offers.

Running Example. Bitcoin is known to UC-emulate a ledger functionality $\mathcal{G}_{\mathsf{ledger}}$ [BMTZ17], which we assume to offer an adversarial interface `readState`[7]. Let $\mathcal{S}_{\mathcal{D}}$ denote the simulator of this emulation statement. Since any permissionless blockchain, and in particular Bitcoin, publicly encodes the ledger state, it holds that $f_{\{\texttt{readState}\}}$ is admissible for $(\mathcal{S}_{\mathcal{D}}, \text{Bitcoin}, \mathcal{G}_{\mathsf{ledger}})$ (the simulator $\mathcal{S}_{f_{\{\texttt{readState}\}}}$ that witnesses admissibility is interacting with Bitcoin and obtains the state the same way an honest miner would do). Now if some blockchain application π proven w.r.t $\mathcal{G}_{\mathsf{ledger}}$ comes with a simulation that only queries $\mathcal{G}_{\mathsf{ledger}}$ with `readState`, the security statement remains valid when $\mathcal{G}_{\mathsf{ledger}}$ is replaced with Bitcoin. That is, π is guaranteed to realize the *same* functionality, regardless of whether $\mathcal{G}_{\mathsf{ledger}}$ or Bitcoin is used as global ledger.

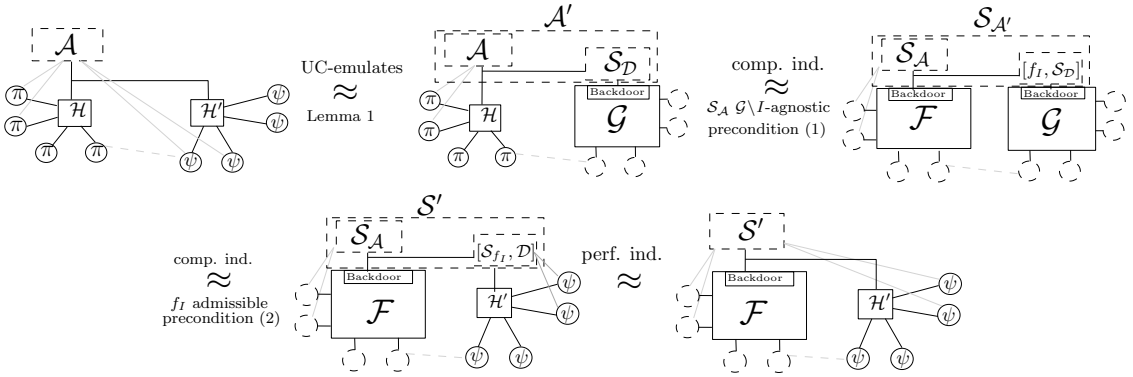
Theorem 4.10 (Full Replacement due to Agnostic Simulations II)**.** *Assume* $\text{EXEC}_{\psi,\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\mathcal{G},\mathcal{S}_{\mathcal{D}},\mathcal{Z}}$ *and let I be a subset of adversarial queries allowed by $\mathcal{G}$ such that f_I is an admissible backdoor-tape filter for $(\mathcal{S}_{\mathcal{D}}, \psi, \mathcal{G})$. Let further π UC-emulate $\mathcal{F}$ in the presence of the global subroutine $\mathcal{G}$ such that the simulator $\mathcal{S}_{\mathcal{A}}$ for this precondition is $\mathcal{G}\backslash I$-agnostic. Then, π, invoking ψ instead of $\mathcal{G}$, UC-emulates $\mathcal{F}$, invoking ψ instead of $\mathcal{G}$, and where ψ is the global subroutine.*

Proof. We first state the theorem in the language of UCGS as before. Let π, $\mathcal{F}$, ψ, $\mathcal{G}$ be protocols and let ξ, $\xi' \in \Xi$ be predicates on extended identities, such that π is $(\psi, \mathcal{G}, \xi)$-compliant, π, $\mathcal{F}$, $\mathcal{G}$, ψ are subroutine exposing, $\mathcal{G}$ and ψ are subroutine respecting, π and $\mathcal{F}$ are subroutine respecting except via calls to $\mathcal{G}$ and $\mathcal{G}$ is π-regular. Let ψ UC-emulate $\mathcal{G}$ with respect to ξ-identity-bounded environments. Let $\mathcal{S}_{\mathcal{D}}$ denote the simulator of this condition, and be I a subset of adversarial queries allowed by $\mathcal{G}$ such that f_I is admissible for $(\mathcal{S}_{\mathcal{D}}, \psi, \mathcal{G})$. Let further π UC-emulate $\mathcal{F}$ in the presence of $\mathcal{G}$ w.r.t. ξ'-identity-bounded environments. Let $\mathcal{S}_{\mathcal{A}}$ denote a simulator for this emulation, and let

[7] In [BMTZ17], any party, including the adversary, can obtain the ledger state by sending $(\texttt{READ}, \mathsf{sid})$ to $\mathcal{G}_{\mathsf{ledger}}$.

$\mathcal{S}_\mathcal{A}$ be $\mathcal{G}\backslash I$-agnostic. Then $\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ UC-emulates protocol $\mathsf{M}[\mathcal{F}^{\mathcal{G}\to\psi}, \psi]$ w.r.t. ξ'-identity-bounded environments.

The sequence of steps needed in this proof are the following.



More formally, we make the following transitions, going from top left to bottom right in the picture.

$\mathsf{M}[\pi^{\mathcal{G}\to\psi}, \psi]$ **UC-emulates** $\mathsf{M}[\pi, \mathcal{G}]$**.** This directly follows from Lemma 4.1 and the precondition of ψ UC-emulating $\mathcal{G}$. Let $\mathcal{A}'$ denote the simulator of this emulation.

$\text{EXEC}_{\pi,\mathcal{A}',\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S}_{\mathcal{A}'},\mathcal{Z}}$**.** We show how to simulate for the specific adversary $\mathcal{A}'$. $\mathcal{S}_{\mathcal{A}'}$ works as $\mathcal{S}_\mathcal{A}$, but lets the internally simulated $\mathcal{A}$ on π issue its external write requests to the global subroutine directly to $[f_I, \mathcal{S}_\mathcal{D}]$ (using the adressing mechanism described in Definition 4.7), which overall has the effect as if $\mathcal{A}$ and $[f_I, \mathcal{S}_\mathcal{D}]$ were combined when talking to the global subroutine. We need to argue that the simulator $\mathcal{S}_\mathcal{A}$ (simulating π while interacting with $\mathcal{F}$) still performs a good simulation even against this combined attacker. Due to $\mathcal{S}_\mathcal{A}$ being $\mathcal{G}\setminus I$-agnostic, $\mathcal{S}_\mathcal{A}$'s requests reach $\mathcal{G}$ unmodified since they pass f_I. Definition 4.9 further ensures that $\mathcal{S}_\mathcal{A}$ acts as a dummy adversary regarding all requests between $\mathcal{Z}$ and $[f_I, \mathcal{S}_\mathcal{D}]$. A distinguisher $\mathcal{Z}$ between both distributions can thus be turned into a distinguisher between executions $\pi, \mathcal{A}$ and $\mathcal{F}, \mathcal{S}_\mathcal{A}$ which runs program $[f_i, \mathcal{S}_\mathcal{D}]$ internally, violating precondition (1).

$\text{EXEC}_{\mathcal{F},\mathcal{S}_{\mathcal{A}'},\mathcal{Z}} \approx \text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}}$**,** where $\mathcal{S}'$ denotes the simulator $\mathcal{S}_\mathcal{A}$ sending requests to ψ via adversary $[\mathcal{S}_{f_I}, \mathcal{D}]$. Recall that $\mathcal{S}_{\mathcal{A}'}$ combines $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_\mathcal{D}$. If both executions are distinguishable, an environment running $\mathcal{S}_\mathcal{A}$ and $\mathcal{F}$ could distinguish an execution of ψ and $[\mathcal{S}_{f_I}, \mathcal{D}]$ from an execution of $[f_I, \mathcal{S}_\mathcal{D}]$ with $\mathcal{G}$, violating the precondition that f_I is an admissible backdoor-tape filter for $(\mathcal{S}_\mathcal{D}, \psi, \mathcal{G})$.

□

4.4 Case study: Comparable Constructions and Random Oracles

The goal of composable frameworks is to obtain a library of constructions that are identified by three elements: assumed functionality, the realized functionality, and the protocol that achieves the construction. When one tries to achieve a new idealization, then one can safely take the known constructions from the library, rely on those realized functionalities as setup, being assured that they can be replaced by their already known implementations at any time. As we showed in this paper, this idea generally fails for global (hybrid) setups, but is partly restored by the above theorems by giving conditions on when such a replacement of a global setup is possible.

Still, the following mismatch might occur in such a modular protocol design which motivates another important aspect of Theorem 4.10. Assume two protocols are proven with respect to

different global setups, π_1 realizes $\mathcal{F}_1$ in the GRO setting, and π_2, which makes (local) calls to $\mathcal{F}_1$ and realizes $\mathcal{F}_2$ w.r.t. a GRO that allows the adversary upon request to program random points of the function table and otherwise is identical to GRO. Therefore, it is hard to obtain a combined security claim: they assume different global setups and hence $\mathcal{F}_1$'s usage in the presence of the observable RO has never been formally realized. Therefore, applying the UCGS theorem is not possible and replacing it by π_1 must be deemed a heuristic.

However, Theorem 4.10 gives us a tool to figure out whether π_2 actually achieves $\mathcal{F}_2$ in the presence of the plain GRO (which in turn would allow us to apply the UCGS composition theorem): For protocol π_2 UC-emulating $\mathcal{F}_2$ in the presence of a global RO that supports, say, adversarial queries I (e.g, including random-points programmability), it is therefore enough to specify the set $I' \subseteq I$ of filter request for which the preconditions of Theorem 4.10 is satisfied. In this case, it follows that the very same construction can be proven with respect to any stronger version of the assumed GRO that blocks inputs from any subset of $I \setminus I'$ and hence preserving the queries that are necessary for this simulator. The reason is that the simulator in the UC-emulation proof of the construction π_2 is agnostic to what happens aside of its filter requests, and this includes the possibility that no request aside of its filter requests of queries in I' are made (and on the other hand, the protocol in the real world is not disturbed by the exact set of queries since it is proven w.r.t. the rpGRO).

The final conclusion is that incomparable constructions can become comparable by security preservation results, such as the one in Theorem 4.10: if I' does not contain the programmability request, then the two protocol π_1 and π_2 work for the same GRO as established by Theorem 4.10. Hence, for those two constructions, π_2 can replace hybrid $\mathcal{F}_1$ by π_1, which is then not a heuristic argument, but a sound composition step backed by the UCGS theorem.

5 Generalization to many Global Subroutines

We now consider protocols that use more than one global setup. Such a situation often appears in the literature, e.g., when a protocol makes use of a global ledger and a global clock, or a global PKI and a global random oracle. Formally, such a protocol is subroutine respecting except via calls to subroutines γ_i, $i \in [n]$. In this section, we show how to leverage the results from the previous section to replace one, or several, or all of the global subroutines γ_i. A bit more formally, we now assume precondition (1) be as follows:

$$(1) \quad \pi \text{ UC-emulates } \mathcal{F} \text{ in the presence of global } \gamma_1, \ldots, \gamma_n$$

Looking ahead, we will have to make some assumptions on the global subroutines $\gamma_1, \ldots, \gamma_n$ and the corresponding protocols $\psi_1, \ldots, \psi_n$ to realize them. Roughly speaking, ψ_n will not depend on any other global subroutine to realize γ_n, while ψ_{n-1} (and hence also γ_{n-1}) is allowed to depend γ_n but on no other global subroutine. We will be more formal about how to define "depend" in this context.

Before formalizing our results, let us describe the idea behind them. Essentially, we will interpret the setups $\gamma_1, \ldots, \gamma_n$ as a single global setup $\widehat{\gamma}$. $\widehat{\gamma}$ simply runs all γ_i internally and dispatches messages correspondingly. For this single global setup $\widehat{\gamma}$, we can interpret precondition (1) above as precondition (1) from the previous section with single setup $\widehat{\gamma}$, and apply the replacement theorems from the previous section. The only open question is: which protocol realizes the single global setup $\widehat{\gamma}$? Note that this emulation is needed to replace precondition (2) in Section 4.1. So let $\psi_1, \ldots, \psi_n$ denote the protocols we want to replace the global subroutines with, i.e., ψ_i UC-emulates γ_i for all

i. We show that, under the condition that all setups form a hierarchy regarding who gives input to whom, $\widehat{\psi}$ UC-emulates $\widehat{\gamma}$.

We first state a general program structure:

Definition 5.1 (Merging subroutines.)**.** Let $\widehat{\rho}_{\rho_1,\ldots,\rho_n} := [\rho_1, \ldots, \rho_n]$ be a program that accepts inputs of the form $(\mathtt{query}, \mathsf{sid}, i, x)$ and invokes subroutine ρ_i with input x, all with respect to the same session sid.

In UC, we must ensure that this simple program structure can be made a compliant protocol (and subroutine exposing) as we are going to replace its subroutines later. For two protocols γ_i, ψ_i, the above program becomes (ψ_i, γ_i, ξ) compliant if it never relays inputs not satisfying the bound ξ by its caller. The remaining, more technical conditions for compliance can be trivially satisfied. In order not to overload notation, we assume such a predicate is known and enforced by $\widehat{\rho}_{\rho_1,\ldots,\rho_n}$. [8] We identify UC-realization with multiple setups with the single global subroutine case as follows:

Definition 5.2 (UC emulation with multiple global setups)**.** Let π, ϕ and $\gamma_1, \ldots, \gamma_n$ be protocols. We say that π *ξ-UC-emulates ϕ in the presence of global subroutines* $\gamma_1, \ldots, \gamma_n$ if protocols π and ϕ are formulated with respect to a global subroutine $\widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}$ and $\mathsf{M}[\pi, \widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}]$ ξ-UC-emulates protocol $\mathsf{M}[\phi, \widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}]$.

Note that the overlay we define is just a dispatching service. Hence, a protocol designer might still define π in the way of having π directly access each γ_i. This transition is straightforward.[9]

We hence obtained a reduction between the single global-subroutine world and the multiple global-subroutine world.

Remark. The following theorem makes the hierarchy idea formal that we discussed at the onset of this section. In order to express that γ_i does not depend on other subroutines γ_j, $j < i$ we use the concept of regularity to ensure that γ_i does only invoke global subroutines that presumably already have been replaced (by condition 1. below, only the γ_i's and no other protocol can be seen as global). This facilitates that for any subroutine γ_i we can make use of precondition 3. that ψ_i realizes γ_i in the presence of global subroutines γ_j, $j < i$, and be sure this is independent of what is yet to be replaced later. This gives a sound order of replacements.

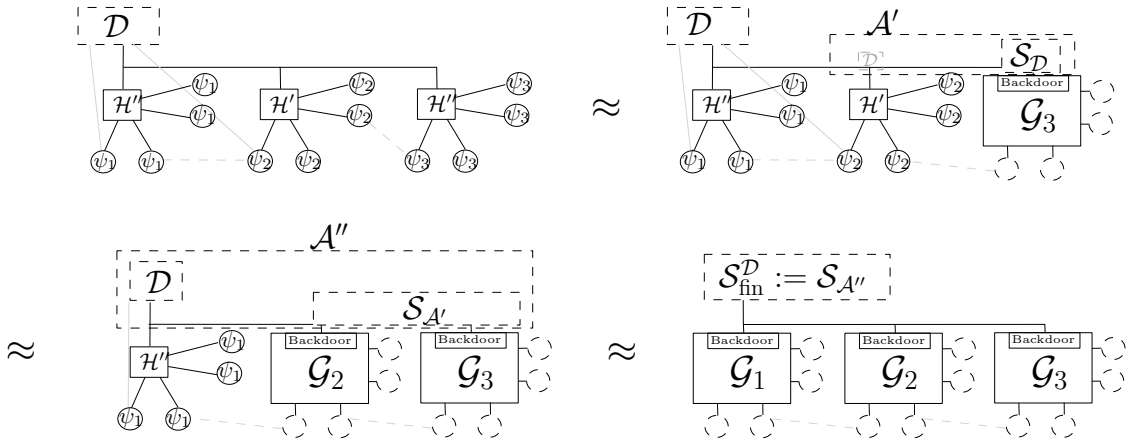
Theorem 5.3 (Reduction Theorem)**.** *Let $\gamma_1, \ldots, \gamma_n$ and $\psi_1, \ldots, \psi_n$ be protocols. $\widehat{\psi}_{\psi_1,\ldots,\psi_n}$ UC-emulates $\widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}$ if for each protocol $\rho_i \in \{\gamma_i, \psi_i\}$ the following conditions hold:*

1. *ρ_i, when $i < n$, is subroutine respecting except for calls to $\gamma_{i+1}, \ldots, \gamma_n$. ρ_n is subroutine respecting. All ρ_i are subroutine exposing.*
2. *ρ_i, when $i > 1$, is γ_j-regular and ψ_j-regular for all $j \in \{1, \ldots, i-1\}$.*
3. *ψ_i ξ-UC-emulates γ_i, for $i < n$, in the presence of global subroutines $\gamma_{i+1}, \ldots, \gamma_n$. And ψ_n UC-emulates γ_n.*

Proof. We again use the transitivity of indistinguishability of ensembles. The sequence of hybrid worlds that are needed to conclude are depicted below for the case of three global subroutines.

[8] The remaining conditions are technicalities such as setting the forced-write flag and not calling ψ_i and γ_i with the same session sid which obviously can be satisfied. For the UCGS theorem, this protocol is compliant if it additionally never invokes a model element, which is obvious.

[9] Whether the transition is also trivial is a different question. In frameworks that have a complex runtime structure, introducing such an intermediate dispatching machine might be costly and would require π to request more runtime-resources. In UC, this would cost k import more for π, where k denotes a security parameter.



Each step is characterized by two elements: a single context protocol μ_i, and the number i which protocol is to be replaced. Let $\mu_i := [\psi_1, \ldots, \psi_i, \gamma_{i+1}, \ldots, \gamma_n]$, $i = 1, \ldots, n$ and $\mu_0 := \widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}$. We start with $\mu_n := \widehat{\psi}_{\psi_1,\ldots,\psi_n}$.

Step 1: In the context protocol μ_{n-1} we perform the replacement $\mu_{n-1}^{\gamma_n \to \psi_n}$, resulting in μ_n. By the Theorem's precondition, we can invoke the UC composition theorem, since γ_n and ψ_n are subroutine respecting and subroutine exposing and μ_n is compliant. Therefore, the UC composition theorem implies $\text{EXEC}_{\mu_n,\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\mu_{n-1},\mathcal{S}_n,\mathcal{Z}}$.

Step $2 \leq i \leq n$: starting with context protocol μ_{n-i} we replace $\mu_{n-i}^{\gamma_n \to \psi_n}$ which results in μ_{n-i+1}. For this step, we can invoke the UCGS theorem since the preconditions of the UCGS theorem are satisfied: γ_i resp. ψ_i can be treated as protocols that are subroutine respecting except with calls to $\gamma_{i+1}, \ldots, \gamma_n$ and hence Definition 5.2 applies. Furthermore, all protocols are subroutine exposing, and formally, the "global setup" of this construction, i.e., the subsystem consisting of $\gamma_{i+1}, \ldots, \gamma_n$, is γ_i- and ψ_i-regular as demanded by the precondition, i.e., they never send input to any of the subroutine prior to i that have not yet been replaced. Hence, the UCGS theorem yields that μ_{n-i} UC-emulates μ_{n-i+1} and in other words, $\text{EXEC}_{\mu_{n-i+1},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mu_{n-i},\mathcal{S}_{n-i+1},\mathcal{Z}}$.

The final step follows by applying transitivity to obtain the final simulator $\mathcal{S}_{\text{fin}}$ for the overall construction. Since we started with the dummy real-world adversary for $\widehat{\psi}_{\psi_1,\ldots,\psi_n}$ this formally yields a simulator for the dummy adversary that proves $\text{EXEC}_{\widehat{\psi},\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\widehat{\gamma},\mathcal{S}^{\mathcal{D}}_{\text{fin}},\mathcal{Z}}$. □

We now set $\psi := \widehat{\psi}_{\psi_1,\ldots,\psi_n}$, $\mathcal{G} := \widehat{\gamma}_{\gamma_1,\ldots,\gamma_n}$ and $\mathcal{S}_{\mathcal{D}} := \mathcal{S}^{\mathcal{D}}_{\text{fin}}$ in precondition (2) in Section 4.1. This yields a precondition that lets us replace all global subroutines using the various replacement theorems from the previous section.

Remark. In some situations, we might want to replace only one global subroutine but not all of them. As an example, consider a protocol accessing a global PKI functionality γ_1, which in turn uses a global RO γ_2. In an instantiation, the global PKI is likely replaced by an interactive protocol ψ_1 (potentially involving a certificate authority, but still using the global RO). To ensure that the protocol's security proof remains valid under this replacement, we need to replace only γ_1 but not γ_2. However, due to the fact that every protocol trivially UC-emulates itself, we can apply Theorem 5.3 with $\psi_2 := \gamma_2$, which will leave the global RO as a proof element.

References

[ACKZ20] Aydin Abadi, Michele Ciampi, Aggelos Kiayias, and Vassilis Zikas. Timed signatures and zero-knowledge proofs - timestamping in the blockchain era. In *ACNS*, 2020.

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In *Financial Cryptography and Data Security - FC*, 2014.

[BCH+20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain uc. Cryptology ePrint Archive, report 2020/1209, 2020.

[BGK+18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 913–930. ACM, 2018.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, 2014.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO*, volume 10401 of *LNCS*, pages 324–356. Springer, 2017.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–145, 2001.

[Can20] Ran Canetti. Universally composable security. Journal of the ACM, Vol. 67, No. 5, 2020.

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *EUROCRYPT*, volume 10820 of *LNCS*, pages 280–312. Springer, 2018.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography*, pages 61–85, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CGJ19] Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. Founding secure computation on blockchains. In *EUROCRYPT 2019*, 2019.

[CGL+17] Alessandro Chiesa, Matthew Green, Jingcheng Liu, Peihan Miao, Ian Miers, and Pratyush Mishra. Decentralized anonymous micropayments. In *EUROCRYPT*, 2017.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2014.

[CKWZ13] Seung Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. Efficient, adaptively secure, and composable oblivious transfer with a single, global CRS. In *PKC 2013*, 2013.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, 2003.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In *Public-Key Cryptography – PKC 2016*, pages 265–296, Berlin, Heidelberg, 2016.

[DEF+19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *EUROCRYPT*, 2019.

[DEFM19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy, SP*, 2019.

[DFH18] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2018.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security, FC*, 2019.

[EMM19] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2019.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *ACM symposium on Theory of computing*, 1985.

[KL20] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. In *33rd IEEE Computer Security Foundations Symposium, CSF*, 2020.

[KMT20] Ralf Küsters and Daniel Rausch Max Tuengerthal. The IITM model: a simple and expressive model for universal composability. Journal of Cryptology, 2020.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT*, 2016.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science*, 2011.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, 2002.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In *EUROCRYPT*, 1999.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT*, 2018.

Appendix

A Examples Illustrating Impossibilities

We sketch two artificial but technically easy examples that show that security statements can indeed fail completely, once a global setup is replaced by another one that UC-emulates it. Here, we replace the global setup by a stronger variant to achieve the contradiction. In the first example, we show that the commitment problem [Nie02] can occur all of a sudden, and in the second one, we illustrate how security guarantees can be blurred when exploiting adversarial capabilities of the global setup.

A.1 Global Repository and PKE

Consider the following communication protocol π with sender S that assumes access to a global repository. S first generates a key pair (pk, sk) and sends pk to the repository. It then takes any input message m and pushes an encryption $c := \mathrm{Enc}_{pk}(m)$ to the global repository and additionally sends c on the network to, say, a list of receiving parties. It also internally stores m and returns the activation to the caller. We point out that we assume here an ordinary encryption scheme. Assume that the global repository records $(S,\ x)$ if sender S provides input x and allows the adversary to read out any recorded pair. We point out that such a functionality is essentially realized by authenticated broadcast.

The ideal functionality $\mathcal{F}_S$ this protocol realizes is clearly the following: $\mathcal{F}_S$ takes input m from S and asks the simulator for a public key pk (and never leaks m). Upon receiving pk, $\mathcal{F}_S$ encrypts m and provides the ciphertext as input to the repository in the name of S[10]. To prove that the protocol realizes $\mathcal{F}_S$, we have to come up with a proper simulator. This is easy: the simulator simulates a public-private key pair, provide $\mathcal{F}_S$ with the public key, and simply take the ciphertext that the functionality pushed to the repository to simulate the ciphertext on the network.

Now, assume we replace this repository by a stronger one that works identically except that the adversary only receives the length $|x|$ when reading any of the sender's records. This corresponds to encrypted broadcast to a list of honest receivers. One would now assume that working with a stronger repository, i.e., using encrypted broadcast rather than authenticated, is better for everyone. However, this is not the case. This change does not only make the above simulation strategy impossible, but in fact, *no simulator* exists to prove that π realizes $\mathcal{F}_S$ unless we change the protocol (e.g, by forcing it to use a non-committing encryption scheme): no simulator has access to the ciphertext and hence a good simulator must simulate a ciphertext without knowing the underlying message m and the simulation is trapped in the commitment problem [Nie02]. The environment can now perform the standard trick: after seeing the ciphertext on the network (either real or simulated), the distinguisher can afterwards instruct the (dummy) adversary to corrupt the sender S and check that the ciphertext contains the right message. Ordinary encryption schemes do with substantial probability not allow to simulate this situation correctly and therefore no simulator exists if π is run with the stronger repository (or when using encrypted broadcast).

A.2 A Global Ledger with Adversarial Reordering

Assume a simple protocol ϕ for some party P that works as follows: it expects as input transactions of a certain type. Before submitting them to a global transaction ledger, ϕ orders the transactions according to size and submits this list to the ledger. Assume that the ledger is a transaction ledger

[10] We refer to [BCH+20] for a definition of this

similar to the one in [BMTZ17] that allows the adversary to re-order transactions before a block is formed and added to the immutable ledger state.

The ideal functionality $\mathcal{F}_\phi$ that this protocol realizes can be the following: it takes as input the list of transactions provided by P, and orders them *differently*, say according to lexicographic order, and submits this list to the global transaction ledger. This is of course weird, but possible to simulate: to prove this construction, we observe that the simulator has the freedom to reorder freely and chooses the ordering that equals the one induced by the actions of the real-world adversary, which even yields a perfect simulation!

Now assume we use a stronger transaction ledger that does not allow to reorder the transaction list in the ledger and hence makes the adversarial capabilities less powerful. However, since no simulator can now change the order, the order of transactions in the transaction ledger directly signals to the environment, whether it interacts with the ideal world (lexicographic order) or the real world (size). Therefore, using a stronger ledger (which UC-realizes the weaker one) renders the construction invalid: no simulator will exist. The point here is that every simulator must crucially carry out a reordering attack and that there is no other strategy to rectify the ideal world if re-ordering is impossible. This shows how usage of a global ideal ledger can create false impressions of security, since $\mathcal{F}_\phi$ is impossible to realize w.r.t any real transaction ledger protocol which disallows arbitrary reordering.

B Overview of the UC Framework

This section gives a summary of the main concepts of the UC framework by Canetti [Can01]. Since its introduction in 2001, the UC framework has undergone a sequence of versions, and this work is based on the UC version of 2020 as specified in [Can20], which for compactness, we refer to as *UC 2020* in this work. While we assume some familiarity with the general concepts of universal composition, we introduce the definitions used in this work for the sake of self-containment and, along the way, point out some of the key differences of UC 2020 with the previous versions of the framework.

To give the reader an overview which notions she should be familiar with, and to provide a glossary of terms for conveniently accessing them, we list the notions and results from UC 2020 that are restated in this section:

Def. B.1: ITI configuration, extended identity
Def. B.2: External-write request
Def. B.3: Identity-bound environment
Def. B.5: Compliant protocol
Def. B.6: Subroutine-respecting protocol
Def. B.7: Structured protocol
Def. B.8: Subroutine-exposing protocol
Def. B.10: UC Emulation
Def. B.11: UC Operator
Thm. B.12: UC Composition Theorem

B.1 Basics

The UC framework aims to capture what it means for a protocol to securely carry out a task and to guarantee this statement in any context in which the protocol is used. To achieve this, UC defines the *real process* of executing a protocol in some environment and in the presence of an adversary. Along the same lines, an *ideal process* is defined to capture what the protocol should achieve. A security proof consists of showing that no (efficient) environment can distinguish the real process and the ideal process. The core defining element of the ideal process is the ideal functionality

that specifies what guarantee each party obtains from the protocol. If a protocol execution is indistinguishable from its ideal process, then we say that the protocol UC-emulates the ideal process.

Protocol and protocol instances. A protocol π is an algorithm for a distributed system and is formalized as an interactive Turing machine (ITM). An ITM has several tapes; the basic tapes are the identity tape (read-only), an activation tape (to encode whether this ITM is active) and the outgoing message tape that is used to store external-write requests (these are instructions to allow a program to give input to another program).

An ITM has three externally writable tapes for holding information that is considered as input from other programs. The input tape intuitively holds the information from the calling program and the subroutine-output tape holds return values from called programs. Last but not least, there is the backdoor tape which formalizes the interaction of an ITM, e.g., a functionality, with the adversary. The backdoor tape is also used by the adversary to send corruption messages to parties to take control over them. (In earlier versions of the UC framework the backdoor tapes were called the *communication tapes;* however they functioned in the same way.)

While an ITM is a static object, UC defines the notion of an ITM instance (denoted ITI), which is defined as the pair $M = (\mu, \mathsf{id})$, where μ is the description of an ITM and $\mathsf{id} = (\mathsf{sid}||\mathsf{pid})$ is its identity, consisting of a session identifier sid and a party identifier pid. Each instance is associated with a Turing machine configuration, which is as usual the contents of all of its tapes and positions of its heads, and the state of the Turing machine. The identity tape of ITI M contains an encoding of $M = (\mu, \mathsf{id})$. An *instance of a protocol* μ, also called a *session* (of the protocol), with respect to a session identifier sid, is defined as a set of ITIs $(\mu, \mathsf{id}_1), (\mu, \mathsf{id}_2), \ldots$ with $\mathsf{id}_i = \mathsf{sid}||\mathsf{pid}_i$. Each such ITI in a given protocol instance is called a *party* or *main party* of that instance. The *extended instance* s of protocol μ is then the transitive closure of machines spawned as a consequence of running μ, i.e., each newly invoked ITI M becomes part of the extended instance if (a) M is a main party of this instance, or (b) M is a subroutine of an ITI that is already in the extended instance, or (c) M was invoked by an ITI that is already a sub-party of this extended instance. An ITI is a *sub-party* of an extended instance if it is in the extended instance but is not a main party of the instance.[11]

Executions. An execution of a system of ITMs is formally a pair (I, C), where I is the initial ITM and C is the control function. The initial configuration of I (which has identity 0) is the first ITI that gets activated by definition. Using the *external-write* mechanism, any ITI can pass input to another ITI. (If the target ITI does not yet exist, a new instance is spawned using the code and identity specified in the request.) An external-write request specifies, amongst other elements, the message as well as source and destination ITI, including whether this is a subroutine output or an input to the destination ITI. The control function is responsible for writing the message (and possibly the identity of the source ITI) on the respective tape of the destination ITI. The control function also defines which ITI is activated next; typically, the destination ITI of the last processed external write request is activated next. An execution consists of a sequence of activations. An activated ITI can change its configuration according to the rules of its code. An activation ends by issuing one external-write request (in case an ITI halts without issuing an external-write request, the initial ITI is activated). The control function guarantees that all invoked ITIs have unique values on their identity tapes, i.e., there are never two ITIs with identical pairs (μ, id) with code μ and identity id.

[11] The extended session includes the transitive closure of the invocation relation when viewed as rooted at the main ITIs of the instance and disregarding invocations made by the main ITIs via their subroutine output tapes.

Execution of a protocol, adversary, and corruption models. In the context of executing a protocol, say π, the above general idea of an execution is instantiated by having the initial ITI be called the environment and defining a specific control function [Can20] to capture a meaningful notion of execution of a protocol: the environment is allowed to only spawn one session of π, i.e., only issue external-write requests that specify a destination ITI with code π and all having the same session identifier. In UC 2020, one can further specify which identities an environment can set as source identities in an input to the protocol. This mechanism can be used to model flexible context restrictions. Note that prior to UC 2020, all source identities except for those that share the session identifier with the test session π were allowed.

Additionally, the environment is allowed to invoke an adversary. Within this execution, the adversary, typically denoted by $\mathcal{A}$, is simply another ITI just with the special identity $(\perp, \perp)$ on its identity tape. The adversary can communicate with other ITIs by writing (only) on their respective backdoor tapes. This tape can therefore be used to model security properties provided by functionalities (e.g., a secure channel could leak the length of the message via the backdoor tape). The backdoor tapes are also used to model party corruption: The adversary can, at any time, issue special corruption messages in order to corrupt ITIs. The exact model of corruption—passive/active, static/adaptive—is specified by how ITIs react to these messages on the backdoor tape. The plain UC model does not prescribe or require any specific corruption model. It is instructive to keep in mind the standard interpretation: when an ITI gets corrupted, it tells the adversary the contents of all tapes, inform the adversary upon any input, and allow the adversary to decide on the next output (in the name of this ITI). This corruption dynamics corresponds to active and adaptive. By default, ideal functionalities cannot be corrupted.

Compared to its previous versions, the UC model does not include a built-in form of communication. If one wants to model potentially insecure communication within a protocol, such as messages sent between different protocol parties (in the sense that an adversary could interrupt, read, and modify the message), then one should specify a channel functionality that the parties use for this communication.

UC emulation. The concept of emulation induces a relation among protocols. Intuitively, a protocol π UC-emulates another protocol ϕ if for any adversary $\mathcal{A}$ against π there is another adversary $\mathcal{S}$ against ϕ such that no (efficient) environment can tell, from the observed input-output behavior, whether it is running with π and $\mathcal{A}$ or with ϕ and $\mathcal{S}$. This indicates that any attack on π can be translated to attacks on ϕ, and thus, π does not admit more attacks than ϕ.

The typical incarnation of this notion is when ϕ is an ideal protocol: More specifically, the ideal protocol is formulated with respect to an ITM $\mathcal{F}$ which is called an ideal functionality and captures "a trusted third party" implementing a protocol task. In the ideal process, the environment $\mathcal{Z}$ interacts with an ideal-world adversary (simulator) $\mathcal{S}$ and a set of trivial, i.e., dummy ITIs representing the protocol machines that only relay inputs to the functionality and forward the outputs. These dummy ITIs are the "access points" of a calling program; they give the environment the impression of interacting with structured protocol ITIs of a party and not an ideal functionality. (The dummy protocol for ideal functionality $\mathcal{F}$ is denoted as $\text{IDEAL}_{\mathcal{F}}$.) $\mathcal{F}$ has to specify all outputs generated for each party, and the amount of information the ideal-world adversary learns (via the backdoor tape) and what its active influence is via its interaction with $\mathcal{F}$. Functionalities directly handle the corruption requests by an adversary via the backdoor tape in UC 2020 (and can adjust their behavior based on this information). Note that one always assumes that a corruption mechanism exposes towards the environment enough information about who is corrupted to enforce that the real and ideal world adversaries corrupt, for example, the same parties—identified by the

party identifier. If a protocol π UC emulates the ideal process with $\mathcal{F}$, then one says that π securely realizes $\mathcal{F}$.

B.2 Technical Definitions

We now state the formal definitions from [Can20] that are used in this work.

B.2.1 Executions

We state the definition of ITI configuration and the extended identity of an ITI.

Definition B.1 (ITI configuration, extended identity)**.** Let M denote an instance of an ITM μ, i.e., the pair $M = (\mu, \mathsf{id})$, where $\mathsf{id} = \mathsf{sid}||\mathsf{pid}$ is an identity string consisting of two parts. A Turing machine configuration of μ is a *configuration of an ITI* M if the contents of the identity tape contains M—which is henceforth referred to as the *extended identity* $\mathsf{eid} = M$. Let further an *activation of an ITI* M refer to a sequence of configurations of M, i.e., state transitions that follow the rules described by μ starting from an active configuration of M until an inactive configuration is reached in which case the ITI waits for the next activation.

ITIs can write to each other's writable tapes via external-write requests that are interpreted by the control function that "delivers" the message.

Definition B.2 (External-write request)**.** An *external-write request* is a message written by an ITI onto its outgoing message tape. It must have the format $(f, \mathsf{eid}_{dest}, t, r, \mathsf{eid}_{src}, m)$. In this vector, $\mathsf{eid}_{dest} = (\pi_{dest}, \mathsf{sid}_{dest}||\mathsf{pid}_{dest})$ is the *destination ITI* with *target code* π and *target session identifier* sid_{dest}, and $t \in \{\mathsf{input}, \mathsf{subroutine\text{-}output}, \mathsf{backdoor}\}$ is the tape that message m shall be written to. If the *reveal-sender flag* r is set, then eid_{src} is written on t as well. Finally, the *forced-write mode* $f = 1$ indicates that if ITI eid_{dest} does not exist yet, then one is created in its initial configuration.

The UC control function requires that the field eid_{src} in the external write request match the contents of the identity tape of the ITI creating the request, unless the initial ITI with identity 0 is creating the request.

When the environment $\mathcal{Z}$ provides input to an ITI in the system, it can choose an arbitrary value for the source identifier eid_{src} it uses in the external-write request. We refer to the source identifiers chosen by the environment in a particular execution as *external identities*. The latest revision of the UC paper [Can20] specifies a method to restrict the external identities that the environment is allowed to choose in an execution, based on a predicate ξ. Predicate ξ is evaluated on the complete view of the environment, namely all inputs and outputs the environment provides to or obtains from other ITIs in the system. One natural predicate is the one that disallows $\mathcal{Z}$ to use as external identity the extended identity of any ITI that it provides input to in the system. Other choices of predicates may be helpful in various scenarios.

Of course, the more relaxed the predicate ξ, the more general the security statement. More restrictive predicates in turn lead to more restrictions on the contexts in which the protocols proved secure with respect to those predicates can be executed.

Definition B.3 (Identity-bounded environment)**.** Let ξ be a predicate on the view of an environment. A *ξ-identity-bounded environment* is an environment $\mathcal{Z}$ that only claims external identities eid (as the source of an input to an ITI eid') such that the predicate ξ on the view of each execution of $\mathcal{Z}$ evaluates to 1. ξ is then also called the *identity bound*.

It is often handy and realistic to assume that the universe of identity bounds has the property that if one proves an UC-emulation statement with no restriction, then this implies the same UC-emulation statement with respect to any more restrictive predicate ξ in the universe of predicates, more precisely:

Definition B.4 (Special class of identity-bound predicates.)**.** The class Ξ of ξ-identity-bounded environments for predicates ξ are defined on the sequence of values appearing on the *outgoing message tape* and the *subroutine output tape* of the initial ITI $\mathcal{Z}$ (instead of the entire view of each execution of $\mathcal{Z}$).

The definition of run-time changed significantly in recent versions of the UC framework (see [Can20], Section 3.2, for the most recent definition). Each message sent between ITIs contains an explicit field called *import*, which encodes a natural number. The number of computation steps performed by an ITI must be, at any point in time, bounded by a polynomial in the accumulated import received by the ITI minus the accumulated import sent by this ITI. In a system of ITMs that is *parameterized* by $k \in \mathbb{N}$, each ITI only starts executing after it received import at least k.

B.2.2 Protocol Properties

The composition theorem makes certain preconditions on the protocol it applies to. We start by introducing some nomenclature from [Can20]. In a given execution that includes two ITIs M and M', ITI M' is a *subroutine* of M if M has passed input to M' or M' passed subroutine output to M. ITI M' is a *subsidiary* of M if it is a subroutine of M or a subroutine of another subsidiary of M.

For the composition theorem to work, the parent protocol (often called ρ) must adhere to certain restrictions. For instance, ρ should not call both π and ϕ with the same session identifier: this would clearly make ρ and $\rho^{\phi \to \pi}$ distinguishable, since the latter protocol would then only invoke one instance of π (this follows from the uniqueness requirement on extended identities in an execution of ITIs). Also, if there exists an identity-bound ξ on the environment when proving a protocol, say π secure, then any protocol using π must use it in the way allowed by ξ.

The following condition, which in this form was introduced in UC 2020, formalizes a restriction to prevent such cases together with restrictions that ensure that the input-output communication between different ITIs is trustworthy:

Definition B.5 (Compliant protocols [Can20])**.** Let ρ, π, and ϕ be protocols, and let ξ be a predicate on extended identities. Protocol ρ is called $(\pi, \phi, \xi)-$*compliant* if the following holds in any execution with a (potentially ξ-identity-bounded) environment:

1. All external-writes made by parties and sub-parties of ρ, where the target tape is the input tape, use the forced-write mode as in Definition B.2. Similarly, all messages received on the subroutine-output tapes of these ITIs are expected to have reveal-sender-id flag on; other subroutine-outputs are rejected.

2. No two external-write instructions, out of all external-write instructions made by the members of an extended instance of ρ, where one instruction has target code π, and the other instruction has target code ϕ, have the same target session identifier.

3. The extended identities of all the ITIs in any extended instance of ρ (in any execution) that pass inputs to ITIs with code either π or ϕ satisfy the predicate ξ (based on the view of the interaction with those subroutines with code either π or ϕ).

Composition has a further precondition that is referred to as *subroutine respecting*. On a high level, this condition means that all sub-protocols of a protocol π receive all their inputs and provide all their outputs through π. This condition is a natural requirement for composition: if sub-protocols of π interacted with protocols outside of π, then the ideal protocol ϕ that is to be realized by π would have to resemble the same structure. Subroutine respecting is the requirement that these inner workings of the protocols remain hidden from the outside.

Definition B.6 (Subroutine respecting [Can20])**.** Protocol π is subroutine respecting if each session s of π, occurring within an execution of any protocol with any environment satisfies the following four requirements, in any execution of any protocol ρ with any adversary and environment (as per the definition of protocol execution; it is stressed that these requirements must be satisfied even when session s of π is a subroutine of ρ, and in particular when the execution involves ITIs which are not members of that extended session s):

1. The sub-parties of session s reject all inputs passed from an ITI which is not already a main party or subsidiary of session s (note that rejecting a message means that the recipient ITI returns to its state prior to receiving the message and ends the activation without sending any message; see [Can20, Section 3.1.2]).

2. The main parties and sub-parties of session s reject all incoming subroutine outputs passed from an ITI which is not already a main party or subsidiary of session s.

3. No sub-party of session s passes subroutine output to an existing ITI that is not already a main party or sub-party of session s.

4. No main party or sub-party of session s passes input to an existing ITI that is not already a main party or sub-party of session s.

A protocol ρ making calls to a subroutine π can be subroutine respecting even if π is not. Consider a case where ρ provides input to subroutines of π, which means that π is not subroutine respecting. At the same time, protocol π and all of its subroutines ignore all inputs from protocols outside of the session of ρ and also do not provide subroutine output to any protocol outside of ρ.

It is convenient to consider protocols that consist of two parts: a *shell* part that takes care of model functionality such as corruption or subroutine replacement, and a *body* part that encodes the actual cryptographic protocol. One advantage of this modeling is that the body is not cluttered with model formalism such as addressing and communication mechanisms. Furthermore, different corruption models can be formalized by using different shells, while leaving the body with the core protocol untouched. It is important to note that the shell mechanism lies at the core of many important definitions in [Can20] and that a protocol body can again consist of a protocol (consisting of shell and body) which yields a sequence of shells (with a clear distinction into inner shells and outer shells). This hierarchy of shells is quite vital to many definitions in UC (for example, the shell introduced by the UC operator is outer to the corruption shell and hence the corruption model does not interfere with the subroutine replacement mechanism). We refer the reader to [Can20, Section 5.1] for further details.

A protocol that follows this structure is called a *structured protocol*.

Definition B.7 (Structured protocol [Can20])**.** A *structured protocol*[12] consists of a shell and a body. The shell can read and modify the state of the body, but the body does not have access to the state of the shell. An activation of a structured protocol starts by running the shell, which may

[12]This property was called *compliant* in previous versions of UC.

(or may not) execute the body sub-process. In case the body executes, it keeps executing until it reaches a special "end of activation state", at which point the shell resumes executing. The body may prepare all the information necessary for executing an external-write operation, but it may not execute this operation. Only the shell executes external-write instructions.

In the model of execution described above, parties of a protocol can generate subroutines with arbitrary codes and identities. Upon the first external-write request to an extended identity, the ITI with that identity is created and will start following its instructions. One effect of this modeling is that the adversary is not necessarily aware of all subroutines that exist in the system as, for instance, session identities may be chosen at random. This is usually undesired, as it renders those subroutines effectively incorruptible. This undesired effect is countered in UC 2020 by the definition of *subroutine exposing*, in which specific ITIs that are readable by the adversary hold a directory of all existing members of an extended instance. We give here a direct constructive definition of this property as it is sufficient to follow this work and refer to [Can20] for a deeper discussion on the subroutine-exposure property.

Definition B.8 (Exposure mechanism of subroutines)**.** A protocol π implements the *subroutine exposing* mechanism if for each instance s of π there exists a special directory ITI with identity tape $(\pi, s||\top)$ that contains the list of the extended identities of all parties and sub-parties of this (extended) instance of π, and returns this list to the adversary upon request. More precisely, this list is a sequence of eid's ordered according to invocation. Each ITI that is a main- or sub-party of this instance notifies the directory ITI of its extended identity immediately upon its invocation, and also of each newly invoked ITI before invoking it. When notified by an ITI M that it has been invoked, the directory ITI adds M to its database if M is a main party of session s, or if some ITI already in the database invoked it.

B.2.3 Emulation and Composition

We are now ready to state the UC security definition—which is protocol emulation—the composition operation, and finally the composition theorem. The security notion targets computational security and is based on the computational indistinguishability of random variables.

The output of the execution of protocol π in presence of adversary $\mathcal{A}$ and in environment $\mathcal{Z}$ is the output of $\mathcal{Z}$, which we assume to be a binary value $v \in \{0,1\}$. We denote this output by $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ where k is the security parameter, $z \in \{0,1\}^*$ is the input to the environment, and randomness r for the entire experiment. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

Definition B.9. Denote by $\mathcal{X} = \{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and $\mathcal{Y} = \{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ two distribution ensembles over $\{0,1\}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are *(computationally) indistinguishable* if for any $c,d \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that $|\Pr[X(k,z)=1] - \Pr[Y(k,z)=1]| < k^{-c}$ for all $k > k_0$ and all $z \in \bigcup_{\kappa\le k^d}\{0,1\}^\kappa$. We use the shorthand notation $\mathcal{X} \approx \mathcal{Y}$ to denote two indistinguishable ensembles.

UC-realization. The UC security definition is stated in terms of emulation of protocols. Intuitively, a protocol π emulates another protocol ϕ if π is at least as secure as ϕ. This is formalized following the simulation paradigm [GMR85], by showing that for every adversary $\mathcal{A}$ against π there exists an ideal adversary (or simulator) $\mathcal{S}$ against ϕ that emulates $\mathcal{A}$'s attack. The additional strength of UC comes from the requirement that the statement hold true in presence of every environment $\mathcal{Z}$.

Definition B.10. Let $n \in \mathbb{N}$, Let π and ϕ be subroutine respecting protocols. We say that π *UC-emulates* ϕ if for any (efficient) adversary $\mathcal{A}$ there exists an (efficient) ideal-world adversary (the simulator) $\mathcal{S}$ such that for every (efficient and balanced[13]) environment $\mathcal{Z}$ it holds that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$. If $\mathcal{Z}$ is ξ-identity-bounded then we say π ξ-UC-emulates ϕ.

Composition. Let π be a protocol that presumably UC-emulates another protocol ϕ. The composition operation in UC is then defined as replacing, in a given protocol ρ, all subroutine calls to protocols with code ϕ by subroutine calls to protocols with code π. The underlying idea here is that one designs protocol ρ with an idealized, simple and abstract protocol ϕ in mind, and later realizes ϕ with the concrete protocol π.

We first state the definition of the UC operator that formalizes the operation on protocols described above [Can20].

Definition B.11 (UC Operator)**.** The UC-operator is a transformation denoted by

$$\rho^{\phi \to \pi} := \text{UC}(\rho, \pi, \phi)$$

and maps a (context) protocol ρ, which presumably makes subroutine calls to ϕ, to a (context) protocol $\rho^{\phi \to \pi}$ that makes subroutine calls (i.e., provides input) to π whenever ρ (or a sub-party of an extended instance of ρ) makes a call to ϕ (more precisely, gives input to any top-level instance of ϕ in an extended instance of ρ). For the technical definition, we refer to [Can20, Section 6.1].

It is important to note that the UC-operator replaces ϕ by π in a code-oblivious fashion, meaning that the transformed protocol has only a different input-output behavior due to the fact that π and ϕ might have a different behavior, and not by the fact that the source eid on the subroutine-output tape denotes π instead of ϕ. The technical definition in [Can20] basically rewrites the shell instructions of the context protocol to ensure a proper replacement (and is overall similar to the approach taken in this paper).

We are finally ready to state the composition theorem. It basically states that if protocol π emulates protocol ϕ, then protocol $\rho^{\phi \to \pi}$ emulates protocol ρ, for any protocol ρ fulfilling certain preconditions.

Theorem B.12 (UC Theorem)**.** *Let ρ, π, ϕ be protocols and let ξ be a predicate on extended identities, such that ρ is (π, ϕ, ξ)-compliant, both ϕ and π are subroutine exposing and subroutine respecting, and π UC-emulates ϕ with respect to ξ-identity-bounded environments. Then $\rho^{\phi \to \pi}$ UC-emulates protocol ρ.*

[13] An environment is *balanced* if it, intuitively speaking, allocates to the adversary at least as much runtime as it allocates to the honest parties. As this notion is not crucial for this work, we refer to [Can20] for details.