Strong Externalized Universal Composability*
Generalized UC Revisited

Jesper Buus Nielsen and Mario Strefler

Aarhus University

Abstract. In this paper we revisit the notion of *generalized universal composability* (GUC) introduced by Canetti, Dodis, Pass and Walfish in 2007[3]. The GUC model was intended to model a practical setting where setup parameters, like a PKI or a CRS, are made public once and for all and then used by many different protocols. We show that there exist protocols which can be proven secure in the GUC model, but which are obviously insecure in practice, in the setting that the GUC model was intended to capture. We then proceed to revise the GUC model to a version that better models the intended practical setting. We call the new notion *strong generalized UC*. We finally prove that the GUC protocols proposed by [3] are also strong GUC secure, i.e., whereas there is a problem with the model in [3], the protocols seem to be secure in the intended setting.

Key words: Generalized Universal Composability

1 Introduction

In this paper we revisit the notion of generalized universal composability (GUC) introduced by Canetti, Dodis, Pass and Walfish at TCC 2007[3]. The GUC model was intended to model a practical setting where setup parameters, like a CRS, are published once and for all and then used by many different protocols. We show that there exist protocols which can be proven secure in the GUC model, but which are obviously insecure in practice, in the setting that the GUC model was intended to capture. In [3], the authors present a simplified version of GUC called EUC (externalized UC) and prove its equivalence to GUC. We revise the EUC model to a version that better models the intended practical setting. We call the new notion Strong Externalized UC (SEUC). We finally prove that the EUC protocol proposed by [3] is also SEUC secure, i.e., whereas there is a problem with the model in [3], the protocols seem to be secure in the intended setting.

1.1 A Brief History of UC and Externalized UC

The UC model[2] was developed to guarantee that a protocol proven secure is also secure in practice, even if run concurrently with many copies of itself and many concurrent copies of other secure and insecure protocols. This is a strong and desirable property. Unfortunately it soon turned out that essentially all interesting tasks cannot be securely realized in the UC model[4,5] unless some kind of setup assumption is provided, like a common reference string (CRS)[6] or a key registration authority (KRA) where the parties give proofs that they know their secret key[1]. The reason essentially is that the UC model is a simulation based model and the simulator has no advantage over the real world adversary, like for instance rewinding, as is the case in other, weaker models.

In the UC model, setup is traditionally modeled as a network resource, i.e., the protocol is proven secure in a hybrid world where the parties have access to an ideal functionality for the given setup. Since such ideal functionalities used as a network resource are initialized together

* Supported by European Research Council Starting Grant 279447. Supported by Danish Council for Independent Research via DFF Starting Grant 10-081612.

with the protocol in the UC model, this means that each protocol will be given a fresh instance of the ideal functionality. I.e., if the ideal functionality publishes a CRS, each (instance of a) protocol will receive a fresh random string, generated after the protocol starts running and unknown to all parties prior to the execution of the protocol. Similarly, if the ideal functionality models a KRA, each (instance of a) protocol will have access to a fresh independent PKI which was completely unused prior to the execution of the protocol, and which cannot be accessed by other protocols. This very poorly models practice, where we envision that several or all protocols share the same PKI or use the same common reference string, e.g., randomness obtained by observing a random naturally noisy phenomenon. It is very impractical to assume that together with each (instance of) a protocol a new PKI will occur or a fresh random string will appear in the sky that cannot be observed by other protocols.

Given that some setup provably *is* needed for UC security, it is problematic that the model does not properly model real world setup assumptions. The purpose of the EUC model is to fix this, by proposing a model where the setup can be shared by several protocols and can be observed by all protocols. This potentially allows to use a single PKI or rely on a single CRS, generated for instance by observing sun spots[7]. Before we can describe how the EUC model deviates from the UC model and what the problems with the EUC model are, we need to dive a little deeper into the UC model.

Technically the UC model works by comparing a real world and an ideal world. In the real world, a protocol π is present, consisting of the parties of the protocol, each modeled as a poly-time interactive Turing machine (ITM).[1] Furthermore, an adversary $\mathcal{A}$ is present. It is the adversary which gives inputs to the corrupted parties and receives outputs from the corrupted parties. Furthermore, $\mathcal{A}$ can see all messages sent and can schedule their delivery. Additionally, $\mathcal{A}$ can fully control the actively corrupted (malicious) parties, i.e., send and receive messages on their behalf. A protocol might also use an ideal functionality $\mathcal{G}$ as an additional resource. Here $\mathcal{G}$ is just another poly-time ITM which can talk to all the parties. This can be used to model the presence of for instance a PKI or a CRS. It is the adversary which gives inputs to $\mathcal{G}$ on behalf of the malicious parties and which receives outputs from $\mathcal{G}$ intended for the corrupted parties. However, $\mathcal{A}$ does not see the inputs and outputs of the honest parties. Finally there is an environment $\mathcal{Z}$. It is $\mathcal{Z}$ which gives inputs to the honest parties and receives outputs from the honest parties. It can furthermore communicate with $\mathcal{A}$ at any time, in particular $\mathcal{A}$ might send all collected information to the environment. The execution ends by $\mathcal{Z}$ outputting a bit, which we think of as its guess at whether it interacted with the protocol or the ideal world. We denote the distribution of this guess after an execution of protocol π using the ideal functionality $\mathcal{G}$ under attack by $\mathcal{A}$ in the environment $\mathcal{Z}$ by $\mathsf{exec}_{\pi,\mathcal{G},\mathcal{A},\mathcal{Z}}$.

The ideal world is specified by an ideal functionality $\mathcal{F}$, which models the behavior that the protocol is intended to have. There is also an ideal world adversary, or simulator, $\mathcal{S}$. It is $\mathcal{S}$ which gives inputs to $\mathcal{F}$ on behalf of the malicious parties and which receives outputs from $\mathcal{F}$ intended for the corrupted parties. It is $\mathcal{Z}$ which gives inputs to $\mathcal{F}$ for the honest parties and receives outputs from $\mathcal{F}$ to the honest parties. It can furthermore communication with $\mathcal{S}$ at any time. The execution ends by $\mathcal{Z}$ outputting a bit, which we think of as its guess at whether it interacted with the protocol or the ideal world. We denote the distribution of this guess by $\mathsf{exec}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

The job of the simulator is to use its ability to give inputs on behalf of the corrupted parties to make $\mathcal{F}$ give the same outputs as in the real world (showing correctness of the protocol) and besides this the simulator should try to simulate the conversation that $\mathcal{Z}$ expects to have with $\mathcal{A}$ (showing privacy of the protocol).

[1] In the following all active entities are poly-time ITM's, so we shall not mention this explicitly anymore.

Formally we require that for all protocol adversaries $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\mathsf{exec}_{\pi,\mathcal{G},\mathcal{A},\mathcal{Z}} \approx_c \mathsf{exec}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, where $\approx_c$ denotes that the guesses are computationally indistinguishable.

Notice that the simulator also simulates the ideal functionality $\mathcal{G}$ used as resource by π. This means that if $\mathcal{G}$ for instance is a CRS, then it is $\mathcal{S}$ which picks the CRS. This allows $\mathcal{S}$ to pick the CRS, crs, in such a way that it learns some trapdoor information about crs. It could, e.g., pick it as $crs = P(r)$ for a uniformly random string r and a one-way permutation P. Now $\mathcal{S}$ knows r and no protocol adversary $\mathcal{A}$ can learn r as crs is picked by $\mathcal{G}$ in the real world, so learning r would involve inverting P on a uniformly random string. This now gives the simulator an advantage over the adversary, and then essentially any task can be realized in a UC secure way.

Note, however, that different simulators for different protocols might of course want to learn different trapdoors, so if two protocol use the same CRS, then the simulation strategies might not work together, as it might not be possible to learn both trapdoors – assuming, e.g., claw-free permutations it is indeed easy to cook up such an example. This is the technical reason why each protocol must be supplied with it own, fresh CRS.

The EUC model then proposes to model a global setup by running the ideal functionality $\bar{\mathcal{G}}$ used for setup out of control of the simulator. I.e., there will also be a copy of $\bar{\mathcal{G}}$ in the ideal world and $\mathcal{Z}$ can interact with this copy of $\bar{\mathcal{G}}$ as in the protocol world. We denote this type of ideal execution by $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$. In short hand we then define the predicate

$$\mathsf{EUC}(\pi,\mathcal{F}) \equiv \forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}(\mathsf{exec}_{\pi,\bar{\mathcal{G}},\mathcal{A},\mathcal{Z}} \approx_c \mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}) \ . \tag{1}$$

Notice that now, if $\bar{\mathcal{G}}$ models a CRS, then it is the copy of $\bar{\mathcal{G}}$ in $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$ which samples crs, and the environment will be able to observe the value of crs. Hence the simulator cannot pick the CRS, it is forced to simulate for the specific value crs chosen by $\bar{\mathcal{G}}$. Unfortunately, this means that now the simulator cannot learn a trapdoor anymore, and hence again has no advantage over the adversary. And, indeed, in [3] it is shown that a global CRS is not sufficient to securely realize even the ideal functionality for commitment, $\mathcal{F}_{com}$. However, on the positive side, the authors in [3] define a functionality called an *augmented CRS* (ACRS). This is an ideal functionality which publishes a "normal" CRS crs, but in addition it allows a corrupted party to ask for some trapdoor information associated with crs and the identity id of the corrupted party. We do not allow honest parties to retrieve their associated trapdoors. An example could be an identity based encryption scheme, where we might make the master public key the CRS crs, and then we could allow a corrupted party id to get the secret key for identity id. We would not allow honest parties to learn their secret key.

If we think of the corrupted parties as being on the team of the adversary and the honest parties on the team of the simulator, then the advantage of the simulator now is that it knows the trapdoor of all the players of the other team, whereas its opponent does not know any trapdoor of the players on the simulators team. This allows to use the well known trick of giving proofs of the form "either I'm running the protocol correctly, or I know the trapdoor information associated with crs and your identity." The authors of [3] use this trick plus a larger number of other technical contributions to construct a protocol UAIBC (for UC Adaptive Identity-Based Commitment) and show that it securely implements $\mathcal{F}_{com}$ in the ACRS model. Via [1] one can then get general multiparty computation from $\mathcal{F}_{com}$ and standard computational assumptions.

The ACRS might look as a strange setup assumption, but notice that in practice we might envision that such a CRS is set up by several highly trusted parties, like governments, standardization bodies and the IACR once a year running a highly secure protocol or physical procedure generating a random string crs, which is then published in an authenticated manner. There is no need to actually provide a service which allows corrupted parties to learn their associated

trapdoor information, all that is needed is to publish a uniformly random string. The reason why there is no reason to provide a service to learn trapdoor information is that only the corrupted parties are allowed to learn this information anyway, and we have proven that *even* if we allowed all corrupted parties to get their associated trapdoor information would the protocol be secure. So, it clearly remains secure if we deny them this information.

1.2 The Problem: Targeted Protocols

The intention of the EUC framework is to model setup which is initialised in advance of the protocol being executed and which may be used by arbitrary protocols.[2] In considering arbitrary protocols we should in particular consider protocols chosen by the adversary, maybe in some subversion-of-security attack. As an illustration, consider the following scenario: a number of highly trusted entities go together and once and for all generate a truly random string crs and make crs public. Now a nicely specious adversary (NSA) mounts the following covert attack. With crs in mind, it designs a protocol π which is insecure when run with exactly crs as setup, but which is EUC secure when run with a random CRS. It then uses a full-fledged, well-written proof that π is EUC secure to have π broadly accepted as being secure and eventually standardized. Now honest parties will start running π, using the already set crs as the global setup. In the following we call protocols which are provably secure in a given model M but which are chosen by the adversary as to be insecure in a given practical setting a *targeted protocol* (of M). This notion is related to the notion of security in the context of adaptive protocols, where the adversary picks a devious protocol after seeing the supposedly secure protocol of the honest protocol designer, and picks it to interact with the "honest" protocol in an insecure manner. The difference is that with targeted protocols, the adversary first gets to see some internal state of the honest protocol and even his own protocol before he has to design the targeted protocol. This allows more devastating attacks.

We show that the EUC model allows a targeted protocol. In the above attack first the setup is initialised, and then the protocol is chosen, and proved secure. In (1) we see that in EUC first π is fixed, as part of the "inputs" to the security definition, and then only in the execution of $\mathsf{exec}_{\pi,\bar{\mathcal{G}},\mathcal{A},\mathcal{Z}}$ or $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$ is the setup $\bar{\mathcal{G}}$ initialized. I.e., the PKI or the CRS is not known when the protocol is designed. This allows the above targeted protocol to slip through.

To be more specific, assume that $\bar{\mathcal{G}}$ is an ACRS functionality and that the ACRS once and for all is set to be $acrs_0$. Now take any secure protocol π for the EUC model, like the one in [3], and construct the following protocol π_{acrs_0}: it runs exactly as π, except that if the ACRS is exactly $acrs_0$, all parties will broadcast all their inputs and outputs.[3] It is still the case that $\mathsf{EUC}(\pi_{acrs_0}, \mathcal{F})$, as in $\mathsf{exec}_{\pi_{acrs_0},\bar{\mathcal{G}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$ the ideal functionality $\bar{\mathcal{G}}$ will sample a fresh ACRS $acrs$, and hence $acrs \neq acrs_0$ except with negligible probability.

One objection against this attack could be that eventually, after some years, the security parameter will be increased and a new ACRS will be published, and then π_{acrs_0} will run exactly as π. So, asymptotically the protocols are still the same. This of course does not help us in practice in those years where the system is insecure, and when the security parameter is increased, the NSA just mounts the same attack again. Notice that we will not spot this attack using current techniques for assessing the concrete security of protocols before deploying them with a concrete security level. A concrete security analysis of π in the EUC model would notice

[2] See for instance the quote "[T]he real-world phenomenon of setup that is set in advance and publicly known throughout the system. The UC with joint state theorem ('JUC Theorem') [...] allows several instances of specifically-designed protocols to use the same setup, but it too does not capture the case of public setup that can be used by arbitrary different protocols at the same time."[3, page 3].

[3] A note to the NSA: One can use, e.g., obfuscation to make a less overt version of this attack.

that $acrs = acrs_0$ happens with appropriately low probability and then reduce to a concrete security analysis of π.

1.3 Non-Uniform Protocols

Notice that our targeted protocol formally is non-uniform, as the adversary will pick a new protocol whenever the ACRS is replaced. This is actually formally possible in the UC model as adversaries are allowed to be non-uniform, via the auxiliary input of the environment.

However, the UC framework does not allow *protocols* to be non-uniform, so formally our protocol π_{acrs} is not an attack inside the model. Note, however, that this is not a way out of the problem that we pinpoint, it is rather *part* of the problem, as there are protocols which can be chosen by the class of adversaries that we want to consider but which are not considered by the model. Furthermore, being given such a protocol in practice, we cannot see that it does not fit the model, as we do not know how the NSA will pick the next level in the protocol and whether it will be done uniformly.

One could then advertise the notion that adversaries and protocols alike should be uniform. Then not even the adversary needed by the above targeted protocol attack would fit the model. However, this seems more a closing of the eyes than a real fix, as the above attack applies in practice whether or not the model recognizes it.

We therefore propose that adversaries and protocols alike should be modeled as non-uniform. In particular so because even some naturally occurring "protocols" are non-uniformly specified, like the US symmetric encryption standard: we had DES, now we have AES, and we do not yet know the protocol for the key levels of 100 years.

1.4 Our Fix

Before describing how we model that protocols might depend on the (A)CRS, let us consider two solutions that do not work.

An immediate idea for modeling that protocols might depend on the specific value, $acrs$, of ACRS would be to have a model where we give $acrs$ to the adversary, and then $\mathcal{A}$ gives us back a protocol π. However, the security would then depend crucially on how $\mathcal{A}$ lets π depend on $acrs$. If it outputs π_{acrs} it should not be considered secure. If it outputs just π, the protocol from [3] it should probably be considered secure, as the protocol indeed seems to be secure in practice. However, in practice, when we want to judge whether π is secure or not, we are only provided with π and $\mathcal{F}$, the code of the adversary is hidden from our eyes, and hence so is the dependence of π on the current instances of the ACRS(s) out in the real world. We could then try to require that the formal definition of security gets as input along with the protocol also a description of how the protocol depends on $acrs$. However, providing this description would then be the job of the protocol designer, *i.e.*, the adversary. The adversary would of course subvert this by designing a protocol like π_{acrs_0} and then just claim that there is no dependence on the current ACRS(s) – it can simply claim that it would have output this exact protocol no matter what the value of the ACRS(s) currently used in the real world is. We therefore need a formal definition of security, which takes just π and $\mathcal{F}$ as input.

However, if π is an input to the security definition it seems we are back at the problem of the EUC model: if now $\bar{\mathcal{G}}$ samples $acrs$ afresh, there will be no correlation between π and $acrs$ allowed. However, in picking π after seeing $acrs$, the temporal aspect is not that important, the attack is based on a correlation between $acrs$ and π. We can allow the adversary the same attack if instead we allow $\mathcal{A}$ to pick $acrs$ after seeing π. This is what we will do then: In our formal model the adversary will get to pick the specific value of $acrs$ to be used. Formally, we will let it be up to the environment $\mathcal{Z}$, as it is present in both $\mathsf{exec}_{\pi,\bar{\mathcal{G}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$. It is,

indeed easy to see that this is a strictly stronger attack than the one we tried to capture: To simulate the targeted protocol attack, the adversary/environment can sample $acrs$ as $\bar{\mathcal{G}}$ would, then design π from $acrs$, and then mount an attack where in $\mathsf{exec}_{\pi,\bar{\mathcal{G}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{exec}_{\mathcal{F},\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$ it asks $\bar{\mathcal{G}}$ to use $acrs$.

Unfortunately, allowing the adversary to pick $acrs$ is vastly too strong an ability to give away, as it might choose to learn the trapdoors of all honest parties, and then the simulator has no advantage anymore. We patch this hole using an idea by Rogaway. We will require from an adversary that it does not learn the trapdoors of the honest parties, by saying that a protocol is secure if it is secure against any adversary from which one cannot extract the trapdoor of any honest party. This is inspired by the human-ignorance definition of security of a fixed hash function. This restriction still allows the adversary to mount correlation attacks, but rules out the trivial attacks where it uses the trapdoor of the honest parties.

1.5 Added Value: Weakly Random Setup

We set out to model setup which is set once and for all and can be shared by arbitrary protocols. However, the notion of SEUC seems to provide more than just the ability to share setup. Note, namely, that in our security game, the security of a protocol is proven against *any* environment that is ignorant of the trapdoors of the honest parties. Hence, there is no reason that an ACRS is uniformly random. All that is required is that it is random enough that the adversary cannot learn the trapdoor.

1.6 Targeted Protocols, a Blind Spot?

We have shown that the EUC model is not sound, because it samples the global setup at random *after* the protocol has been designed. This does not closely model the real-world phenomenon that the setup "is set in advance". However, the real problem, in our view, is a more general one, not associated specifically to the EUC model. It seems that targeted protocols is a blind spot of many of our security models.

The general goal of modern cryptography is to make no assumptions on the behavior of the adversary, as it has proven hard to enumerate all possible attacks that an adversary might mount. And, we go to great lengths to allow the adversary any possible behavior in our models, sometimes imposing the only assumption that the adversary is efficient. However, most models assume the attack begins along with the execution of the protocol, at which point the protocol has already been fixed. In particular, few models pay explicit attention to targeted protocols. It would be interesting to investigate whether more existing models allow targeted protocols.

It would also be interesting to formulate a general model of targeted protocols, where the adversary is allowed to choose provably secure (sub-)protocols on the fly as part of the attack, maybe even after having seen internal values of an execution of the protocol that the targeted protocol should run along with or as a sub-protocol of. The techniques for modeling global setup that we introduce in this paper do not seem to generalise to this broader setting in any straight-forward manner.

1.7 Road Map

In section 2, we give some standard, basic definitions. In section 3, we introduce the SEUC model and formulate and prove its composition theorem. In section 4, we separate the EUC model and the SEUC model, by formalising our targeted protocol from above. In section 5, we prove that the UAIBC protocol from [3] securely implements $\mathcal{F}_{com}$ in the SEUC model, under a slightly stronger assumption on one of the primitives than used by the original proof.

2 Definitions

We let $\mathbb{N}$ be the set of positive integers. A function $\nu : \mathbb{N} \to [0,1]$ is *negligible*, if for every $c > 0$ there is $n_c \in \mathbb{N}$ such that $\nu(k) < k^{-c}$ for all $k > n_c$. We denote by *Negl* the set of negligible functions. A function $f : \mathbb{N} \to \mathbb{N}$ is *superadditive*, if $\forall n, m \in \mathbb{N} : f(n+m) \geq f(n) + f(m)$.

In this paper, $\mathcal{PPT}$ (for probabilistic polynomial time) is the set of all algorithms whose running time is polynomial in the total length of their input. We also define the set $\mathcal{IPT}$ (for interactive probabilistic polynomial time) of ITMs whose running time is bounded by a superadditive polynomial p in the following way: Let i_j be the total number of bits written on all the machine's input tapes until step j, o_j the total number of bits written by the machine on all its output tapes until step j. Then at step j we have $j \leq p(i_j - o_j)$. Note that this implies $\forall j \in \mathbb{N} : i_j > o_j$, since $p(0) \leq 0$ by its superadditivity.

For a probabilistic algorithm A taking t inputs, $y = A(x_1, \ldots, x_t; r)$ is the output of A when run on input $x_1, \ldots, x_t$ and coins r. Usually we assume that the coins r are drawn uniformly at random and write only $y \leftarrow A(x_1, \ldots, x_t)$. $[A(x_1, \ldots, x_t)]$ is the set of y that have positive probability of being output by A.

Let X, Y be two probabilistic algorithms $\mathbb{N} \times \{0,1\}^* \to \{0,1\}$ that use a polynomial (in the first parameter) number of random coins. For fixed inputs $k \in \mathbb{N}, z \in \{0,1\}^*$, we can see $X(k,z)$ and $Y(k,z)$ as probability distributions. We define the (computational) *distance* between the two probability distribution ensembles $X = \{X(k,z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$ and $Y = \{Y(k,z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$ as the function

$$\Delta(X,Y)(k) = \max_{z \in \{0,1\}^*} |\Pr[X(k,z) \to 1] - \Pr[Y(k,z) \to 1]|,$$

where the probabilities are over the random coins.

We say that two probability distribution ensembles X, Y are (computationally) indistinguishable ($X \approx_c Y$), if there is a negligible function ν such that

$$\Delta(X,Y)(k) < \nu(k).$$

3 Strong Externalized UC

In this section we specify our model of strong externalized UC (SEUC, pronounced see-U-see). We assume basic knowledge of the UC model. We reuse the basic model of Externalized UC (EUC) from [3] and make a few changes.

A Crash Course on the EUC Model The EUC models starts from the UC model and adds the notion of shared functionalities, which are allowed to interact with more than one protocol session, and we denote them with a bar, as in $\bar{\mathcal{G}}$. We still assume that all protocols are sub-routine respecting, except that they are allowed shared access to the shared functionalities as $\bar{\mathcal{G}}$. We say that they are $\bar{\mathcal{G}}$-sub-routine respecting. In [3, sec. 2.2.3] the notion is stated as follows: "none of the sub-parties of an instance of π provides output to or receives input from any ITI that is not also party/sub-party of that instance of π, *except* for communicating with a *single instance* of the shared ITI $\bar{\mathcal{G}}$".

Recall that in the UC model, the environment is only allowed to invoke a single copy of the protocol π being proven secure, we call this the *challenge protocol*. It is in particular not allowed to invoke or communicate with any protocol or ideal functionality used by π. This is to prevent that π unknowingly starts using an instance of a sub-protocol or ideal functionality that the environment has been playing with: only π gets to interact with its own sub-protocols.

The EUC model relaxes this requirement by requiring that the environment is only $\bar{\mathcal{G}}$-externally constrained, *i.e.*, it has the same restrictions as the environment of the UC model, but is allowed to invoke the instance of $\bar{\mathcal{G}}$ that is to be used by π. And, $\mathcal{Z}$ is allowed to arbitrarily interact with $\bar{\mathcal{G}}$, except that it is not allowed to makes calls to $\bar{\mathcal{G}}$ using the session identifier of the challenge protocol π or any of its sub-instance, to maintain the requirement that the environment is not allowed to directly observe or influence the network communications of the challenge protocol. This will allow $\mathcal{Z}$ to internally emulate any other protocol also using $\bar{\mathcal{G}}$, but will not allow it to interfere with the way π uses $\bar{\mathcal{G}}$.

Non-Uniform Protocols The first very minor change that we make is to allow protocols to be non-uniform. Formally, a non-uniform π is a series of protocols $\pi(k)$, where k is the security parameter. Here each $\pi(k)$ is a protocol in the usual sense. We measure the running time of a party in $\pi(k)$ as the size of the code of the ITM describing the party plus the usual running time as defined in the UC model. Protocols are run as usual, except that at security level k, we run the protocol $\pi(k)$. We already argued in the introduction why it is more natural to model protocols as non-uniform: the environment/adversary is non-uniform and we want to consider all protocols that might be chosen by the adversary, and in addition, some natural protocols are better modeled as being non-uniform.

Allowing Targeted Protocols Our next change is to allow protocols to be targeted, *i.e.*, to be chosen adversarially after playing with the shared functionality $\bar{\mathcal{G}}$ for a while. As explained in the introduction, in the real world, protocols can be constructed after global setup has been made public, so we need to incorporate this into our model.

You are Given Just a Protocol What we ideally want is for our model to allow that π is chosen after the setup functionality has been running for some period of time, and that π might be chosen adversarially by the environment. However, definitions of the form that we allow the adversary or environment to choose the protocol after seeing the setup does not lead to a workable definition.[4] Consider, *e.g.*, a protocol π_0 for a global CRS model which had been chosen to be secure for all values of the CRS, except for one hardcoded value crs_0. If the environment designed π_0 after seeing the global CRS, crs, and used $crs_0 = crs$, then this protocol is not secure. If, however, crs_0 was chosen uniformly at random, then the corresponding protocol, π_1 say, should probably be considered secure. However, when given a protocol in practice, we are given one fixed protocol, not the distribution from which it was drawn, so we cannot distinguish these two cases. In other words, if the definition of security depends on *how* the environment chose the protocol, then we cannot analyze the protocol without knowing if it was chosen adversarially or not and how the environment chose the protocol. It seems overly optimistic in practice that the attacker will tell us how the protocol was chosen. Consequently, we need a definition which takes a protocol as input and tells us whether it is secure or not.

Correlation is Sufficient Having the protocol π quantified/fixed first before we quantify over all environments makes it impossible to let the protocol be chosen by the environment after the environment sees the setup. We solve this apparent deadlock by instead letting the environment chose the *setup* after seeing the protocol. The problem of the protocol having exactly the global CRS hardcoded is a problem of correlation not a temporal problem *per se*: if we have a protocol with a hard-coded "trapdoor" CRS and the global CRS happens to be the same value, then the protocol is insecure, it does not matter in which order this correlation came about.

[4] In the following discussion we will for brevity use *environment* to mean *environment or adversary*.

Bear in mind, that letting the environment choose the setup is only a model choice made to facilitate correlation between setup and protocols, it is not our *goal* to model that the environment actually chooses the setup, it is a *tool*. Our goal is just to allow that protocols and global setups are adversarially correlated. It therefore makes sense to give the environment as restricted a power as possible, which still allows it to create correlation between protocols and the setup.

What we will do is say that the setup *functionality* will be the correct one, and during the actual protocol run, it behaves exactly as expected. All the environment gets to do is run the setup functionality for some period before the protocol gets to interact with it, and in this phase, called the *pre-execution phase*, the environment determines the random coins r_{pre} used by the functionality – think that each time the setup functionality needs a random bit, it will ask the environment to provide it. After the environment activates the challenge protocol the first time, the setup functionality will use fresh, internal randomness for all future random choices. This is exactly enough power that the environment can chose, *e.g.*, a global CRS to be the hardcoded "trapdoor" CRS of a protocol. Again, bear in mind, that we do not want to model that the environment can control the setup functionality like this in practice, it is still just a tool. And, as it happens, even this restricted tool that we used is too strong for the goal. This will cause other problems, which we address soon.

First notice, however, that with the definition we have now, both π_0 and π_1 are considered insecure. This consequence of the definition actually makes sense: In practice we cannot distinguish π_0 and π_1, and π_0 is insecure. So, in practice we must also consider π_1 insecure.

As said, we make a first attempt at defining security for a protocol π using a setup functionality $\bar{\mathcal{G}}$ by using an instance of $\bar{\mathcal{G}}$ whose random tape r_{pre} is chosen by the environment. We move the ideal functionality to the superscript to reflect this special access. As in the EUC model, we say that π securely realizes $\mathcal{F}$, if

$$\forall \mathcal{A} \in \mathcal{IPT}\ \exists \mathcal{S} \in \mathcal{IPT}\ \forall \mathcal{Z} \in \mathcal{IPT}:$$
$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}\ .$$

The main change lies in the definition of the experiment $\mathsf{exec}^{\bar{\mathcal{G}}}_{\cdot,\mathcal{Z}}$, which has two phases: In the first phase, $\mathcal{Z}$ is allowed to interact with $\bar{\mathcal{G}}$. During this phase, when $\bar{\mathcal{G}}$ reads its random tape, it will instead read a designated extra tape of $\mathcal{Z}$ containing a string which we will call r_{pre}. The environment gets to determine the value r_{pre}. The ideal functionality has left-to-right read-only access to r_{pre} and $\mathcal{Z}$ has left-to-right write-only access to r_{pre}. This first phase is the same in the real and ideal executions. The second phase is an execution of the experiment as usual, and we allow $\bar{\mathcal{G}}$ to flip new coins during the second phase. We model this by letting the random tape of $\bar{\mathcal{G}}$ be reset to a uniformly random string independent of r_{pre} at the beginning of the second phase, and still allowing only left-to-right, read-only access. The pre-execution phase ends when the environment activates the first entity which is not the copy of $\bar{\mathcal{G}}$ to be used in the protocol.

After the pre-execution phase $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$ will then be executed exactly as in the EUC model, except that π will use the instance of $\bar{\mathcal{G}}$ initialized by the environment in the pre-execution phase. Similarly, $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ will be executed exactly as in the EUC model, except that $\mathcal{S}$ will use the instance of $\bar{\mathcal{G}}$ initialized by the environment in the pre-execution phase.

Environmental Ignorance A rather obvious problem with this definition is that the environment is allowed to pick and hence see the randomness of the setup functionality. We know that any setup must contain a trapdoor for the simulator. In the formulation above, the environment might chose to learn all these trapdoors and misuse them. Since this does not correspond to

any of the attacks we want to model in the actual real world, but is a consequence of having used too strong a tool, we need to patch the definition to get rid of this problem.

A similar problem was treated by Rogaway [8]: For any collision-resistant hash-function (CRHF) there is an adversary that finds a collision, because it is hard-coded into its description. The solution Rogaway described was to give an explicit reduction from an adversary that breaks a construction using a CRHF to an adversary breaking the collision-resistance. In that case the scheme based on the hash-function is secure until at collision is found. We will follow a similar approach.

Note that if the protocol is secure when the environment does not misuse any trapdoor, then any noticeable difference in the distributions $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$ must come from a misuse of trapdoors. We can therefore still hope to prove security against environments that do not know the trapdoors of honest parties: we call such an environment *ignorant* and we call the principle that environments are not allowed to know/use trapdoors of honest parties *environmental ignorance*. We will only prove security against ignorant environments.

We make the definition constructive by demonstrating that an environment breaking the scheme has a trapdoor, by extracting the trapdoor from the environment. Essentially, there should exist a poly-time algorithm $\mathsf{Extract}$ such that if it is not the case that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}} \ ,$$

then

$$\Pr[\mathsf{td} \leftarrow \mathsf{Extract}(\mathcal{Z}) \text{ and } \mathsf{td} \text{ is a valid trapdoor}]$$

is non-negligible in k. Here $\mathsf{Extract}(\mathcal{Z})$ denotes that the extractor gets non-black-box access to the environment.

We could have made a stronger definition, requiring that the probability of extracting being equal to the distinguishing probability, but there is no need to do this. Note, namely that the job of the extractor is only to show that the environment behaved in a way it should not: it sometimes remembers a trapdoor. There is no reason that the extractor should be forced to extract with probability equal to the distance. An environment having a hard-coded trapdoor is considered outside the environment class, and we don't care about an attack carried out by a cheating environment from outside the environment class.

What is a Trapdoor? We are then left with deciding what it means for the extractor to produce a hardcoded trapdoor. Recall that what we want is that the extractor demonstrates that the environment "misused" the slightly too strong tool we gave it. We give it control over the randomness of the setup to allow it to create correlation, not to let it read out "trapdoors". It is, however, hard to give a generic definition of what a trapdoor is, save predicting what all future global setups will look like. We therefore delegate this definition to the global setup. We will require that all global setup functionalities take a special input from the adversary/simulator of the form $(\texttt{trapdoor?}, x)$. In response to such an input it will return a value $(\texttt{trapdoor}, x, t)$, to the adversary/simulator. Here t is a bit. The bit might depend on the state of the functionality and who is corrupted, but the functionality is not allowed to communicate with any other entities during the computation of t. We think of $t = 1$ as meaning that x was illegal trapdoor information, which the adversary ought not know at this point in the execution, like the secret key of an honest party. It should be hard to make the setup functionality output $t = 1$ when not controlling or seeing its random tape, but besides this we require nothing more from this trapdoor-identification mechanism. The goal of the extractor is then to make the setup functionality output $(\texttt{trapdoor}, x, 1)$.

Definition 1 (Well-formedness of setup). *For an ideal functionality $\bar{\mathcal{G}}$, let π be the dummy protocol with the session identifier being that of the challenge protocol and with access to $\bar{\mathcal{G}}$. We say that $\bar{\mathcal{G}}$ is well-formed for global setup if for all environments $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$ it holds that the probability that $\bar{\mathcal{G}}$ outputs a value of the form $(\texttt{trapdoor}, \cdot, 1)$ in $\mathsf{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k)$ is negligible in k. Furthermore, if a well-formed functionality is activated by the environment, then it will never give output to any other party but the environment in that activation.*

Recall that we defined that the pre-execution phase ends when the environment activates the first entity different from $\bar{\mathcal{G}}$. Combining this with $\bar{\mathcal{G}}$ returning the activation to $\mathcal{Z}$ when activated by $\mathcal{Z}$ we get that when the pre-execution phase ends, no entities except $\mathcal{Z}$ and $\bar{\mathcal{G}}$ were activated or received any inputs. It is to get this property that we require that $\bar{\mathcal{G}}$ does not pass the activation to any other instance but $\mathcal{Z}$. As a consequence, when $\bar{\mathcal{G}}$ is well-formed for global setup, then $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ does not depend on π or $\mathcal{A}$ during the pre-execution phase,[5] and we will therefore write $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}}(k,z)$.

As an example, it was shown in [3] that it is impossible to EUC-realize $\mathcal{F}_{com}$ in the standard CRS model. To circumvent this result, the authors propose a global setup, ACRS, that allows for per-user secrets. The environment/adversary/simulator are only allowed to ask for secrets for the corrupted parties. We can then define a bad trapdoor to be the secret of an honest party, i.e., we extend [3] to return $t = 1$ when presented with the secret of an honest party. We return to this example formally later, but for now, let us get the definition finished.

Extraction Success We first define a notion of extraction success, where an extractor is given an environment in a state at which the pre-execution phase has just ended. Then it must extract a trapdoor, *i.e.*, make $\bar{\mathcal{G}}$ output $t = 1$.

Definition 2 (Extraction success). *For an ITM $\bar{\mathcal{G}}$ well-formed for global setup, an unbounded time ITM $\mathsf{Extract}$, a $\bar{\mathcal{G}}$-externally constrained, IPT environment $\mathcal{Z}$ and a natural number c, define the following experiment:* $\mathrm{Exp}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}(k,z) \equiv$

1. *First run $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}}(k,z)$ until the pre-execution phase ends,* i.e., *$\mathcal{Z}$ writes a message intended for an ITM other than $\bar{\mathcal{G}}$. At this point, let $\sigma_{\bar{\mathcal{G}}}$ denote the current state of $\bar{\mathcal{G}}$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}}$ and similarly let $\sigma_{\mathcal{Z}}$ denote the state of $\mathcal{Z}$. We allow that $\mathcal{Z}$ deletes part of its state at the end of the pre-execution phase and we let $\sigma_{\mathcal{Z}}$ denote the pruned state.*
2. *Let $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ denote the ITM $\mathsf{Extract}$ with $\sigma_{\mathcal{Z}}$ placed on the input tape.*
3. *Let $\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})$ be a copy of $\bar{\mathcal{G}}$ in state $\sigma_{\bar{\mathcal{G}}}$,* i.e., *it is the ideal functionality which ignores the initial input containing k given to it by the UC framework, and then it runs from the state $\sigma_{\bar{\mathcal{G}}}$.*
4. *Run $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma_{\mathcal{Z}})}(k,z)$ for k^c steps. Note that k and z are given as input to $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ by the execution logic of the UC framework.*
5. *If during the execution in Step 4 the ideal functionality $\bar{\mathcal{G}}$ outputs a value of the form $(\texttt{trapdoor}, \cdot, 1)$, then output 1. Otherwise, output 0.*

We define the family of random variables $\mathsf{Succ}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}(k,z) = \Pr[1 \leftarrow \mathrm{Exp}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}(k,z)]$.

Remark 3. – The reason why we allow $\mathcal{Z}$ to delete part of its state between the pre-execution phase and the protocol-execution phase is that we made the model choice that $\bar{\mathcal{G}}$ reads its random choices from $\mathcal{Z}$ during the pre-execution phase, so even in the benign case where the environment would let these choices be random and not look at them, the extractor could

[5] Except that $\mathcal{Z}$ is restricted by the presence of π to not activate $\bar{\mathcal{G}}$ on the session identifier of the challenge session.

inspect $\mathcal{Z}$ to get the randomness used by $\bar{\mathcal{G}}$. This would allow the extractor to demonstrate misuse when there was none, giving a nonsensical definition. We therefore only consider trapdoor information a misuse on behalf of the environment if it is passed from the pre-execution phase to the protocol-execution phase by the environment.

– Recall that $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma_{\mathcal{Z}})}(k)$ denotes running the dummy protocol for $\bar{\mathcal{G}}$ and the dummy adversary, with $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ as environment. Hence inputs to the dummy protocol from $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ goes to $\bar{\mathcal{G}}$ with the session identifier of the challenge protocol, and the corresponding outputs go back to $\mathsf{Extract}(\sigma_{\mathcal{Z}})$. Hence $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ has the power to provide all inputs to and see all outputs from $\bar{\mathcal{G}}$. Furthermore, it has full access to special communication with $\bar{\mathcal{G}}$ via the dummy adversary. Hence, $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma_{\mathcal{Z}})}(k)$ is just a convenient way to say that we give $\mathsf{Extract}(\sigma_{\mathcal{Z}})$ full non-rewinding black-box access to $\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})$. We define the black-box access like this to not have to explicitly define what it means that the extractor has non-rewinding black-box access to the setup "as in the UC execution".

Definition 4 (Environmental ignorance, well-formedness for environments). *Let* $\mathsf{Succ}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}$ *denote the family of distributions* $\{\mathsf{Succ}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$. *Let* $\mathbf{0}$ *denote the family of distributions* $\{0\}_{k\in\mathbb{N},z\in\{0,1\}^*}$. *We say that* $\mathcal{Z}$ *is* ignorant of trapdoors for honest parties in $\bar{\mathcal{G}}$ *if for all* $\mathsf{Extract}\in ITM$ *and all* $c\in\mathbb{N}$ *it holds that* $\mathsf{Succ}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}\approx_c \mathbf{0}$. *We say that* $\mathcal{Z}$ is well-formed for global setup $\bar{\mathcal{G}}$ *if* $\mathcal{Z}$ *is a* $\bar{\mathcal{G}}$*-externally constrained environment and* $\mathcal{Z}$ *is ignorant of trapdoors for honest parties in* $\bar{\mathcal{G}}$.

Note that $\mathcal{Z}$ being $\bar{\mathcal{G}}$-externally constrained means that in the pre-execution phase it does not activate $\bar{\mathcal{G}}$ on the session identifier of the challenge protocol. Furthermore, when $\bar{\mathcal{G}}$ is activated by $\mathcal{Z}$ it always returns the activation to $\mathcal{Z}$, so it never sends messages to any other party but $\mathcal{Z}$. In particular, $\bar{\mathcal{G}}$ never sent messages with the session identifier of the challenge protocol, nor did it send messages to the adversary. Hence, right after the pre-execution phase 1) only $\mathcal{Z}$ and $\bar{\mathcal{G}}$ were ever activated, and 2) neither $\mathcal{Z}$ nor $\bar{\mathcal{G}}$ sent messages for the challenge protocol or the adversary. We say that the system consisting of the two ITMs $\mathcal{Z}$ and $\bar{\mathcal{G}}$ is open for the session identifier of the challenge protocol, meaning that any protocol with the session identifier of the challenge protocols can be plugged into the system $\{\mathcal{Z},\bar{\mathcal{G}}\}$ and run in the UC framework.

Lemma 5 (Pre-execution lemma). *Let* $\mathcal{Z}(\sigma_{\mathcal{Z}})$ *denote the environment which ignores the inputs* k *and* z *and then continues from state* $\sigma_{\mathcal{Z}}$. *Let* π *be a* $\bar{\mathcal{G}}$*-sub-routine respecting protocol. Let* $\mathcal{Z}$ *be* $\bar{\mathcal{G}}$*-externally constrained. Then first running* $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}}(k,z)$ *to get* $\sigma_{\bar{\mathcal{G}}}$ *and* $\sigma_{\mathcal{Z}}$ *and then running* $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\pi,\mathcal{A},\mathcal{Z}(\sigma_{\mathcal{Z}})}(k,z)$ *will give exactly that same output as running* $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$. *Similarly, first running* $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}}(k,z)$ *to get* $\sigma_{\bar{\mathcal{G}}}$ *and* $\sigma_{\mathcal{Z}}$ *and then running* $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathcal{F},\mathcal{S},\mathcal{Z}(\sigma_{\mathcal{Z}})}(k,z)$ *will give exactly that same output as running* $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$.

Defining Security In defining security we will only address the notion of securely realizing an ideal functionality. We will furthermore only address real-world and hybrid-world executions with the dummy adversary. When $\mathcal{A}$ is the dummy adversary we write $\mathsf{exec}_{\pi,\mathcal{A},\mathcal{Z}}$ as $\mathsf{exec}_{\pi,\mathcal{Z}}$ for brevity. The definition easily extend to the notion of protocol emulation and general adversaries.

Definition 6 (SEUC Security). *Let* $\bar{\mathcal{G}}$ *be any ideal functionality well-formed for global setup. Let* π *be any* $\bar{\mathcal{G}}$*-sub-routine respecting protocol. Let* $\mathcal{Z}$ *range over all IPT environments that are well-formed for global setup* $\bar{\mathcal{G}}$. *Let* $\mathcal{S}$ *range over all IPT simulators. Let* $\mathcal{F}$ *be an ideal functionality. We say that* π SEUC realizes $\mathcal{F}$ with global setup $\bar{\mathcal{G}}$ *if*

$$\exists\mathcal{S}\,\forall\mathcal{Z}(\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}\approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{Z}})\ .$$

We say that a SEUC environment is a *benign environment* if it provides $\bar{\mathcal{G}}$ with uniformly random bits in the pre-execution and then deletes these bits, *i.e.*, it passes on the state $\sigma_{\mathcal{Z}} = \perp$. We define *non-uniform EUC* as SEUC, but restricted to benign environments. This is essentially the notion of EUC, except that we allow protocols to be non-uniform.

One could expect most proofs relative to the above definition to be of the following form: First a simulator $\mathcal{S}$ is constructed. Then the simulator is proved to work by a proof of contradiction: Given an environment $\mathcal{Z}$ such that $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ is not indistinguishable from $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{Z}}$, construct an extractor $\mathsf{Extract}$ that demonstrates that $\mathcal{Z}$ is not ignorant according to Definition 4, *i.e.*, it uses the state of $\mathcal{Z}$ at the end of the execution phase to produce a trapdoor, *i.e.*, it makes $\bar{\mathcal{G}}$ output $t = 1$ while acting just as a normal IPT UC environment towards $\bar{\mathcal{G}}$. In doing this, it is important that $\mathsf{Extract}$ can run in IPT and still at least run $\mathcal{Z}$, so we will discuss this issue briefly now.

Recall that the notion of polynomial time of an ITM in [2] is constructed such that $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ and $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{Z}}(k,z)$ can be run in standard polynomial time in k for some polynomial. So, since we allow $\mathsf{Extract}$ a polynomial running time k^c which might depend on $\mathcal{Z}$ (as c is quantified after $\mathcal{Z}$), it can run $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ and $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi,\mathcal{Z}}(k,z)$ internally. If it needs to run them many times, say to allow rewinding, one can just choose c high enough to allow this too in Definition 4.

SEUC implies EUC implies UC Note that for any c, the experiment $\mathrm{Exp}^{\bar{\mathcal{G}}}_{\mathsf{Extract},\mathcal{Z},c}(k,z)$ can be run in poly-time. If furthermore $\mathcal{Z}$ is a benign environment, then $\sigma_{\mathcal{Z}}$ would be empty and independent of $\sigma_{\bar{\mathcal{G}}}$. Hence $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma_{\mathcal{Z}})}(k,z)$ would have exactly the same distribution as $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathsf{Extract}(\perp)}(k,z)$. Since $\mathsf{Extract}(\perp)$ is poly-time it then follows from Definition 1 that the probability that $\bar{\mathcal{G}}$ outputs $t = 1$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathsf{Extract}(\perp)}(k,z)$ is negligible. From this it then follows that the probability that $\bar{\mathcal{G}}$ outputs $t = 1$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathsf{Extract}(\sigma_{\mathcal{Z}})}(k,z)$ is negligible. Hence a benign environment $\mathcal{Z}$ is well-formed for global setup $\bar{\mathcal{G}}$. From this it follows that SEUC implies EUC, and since EUC implies UC, we also get that SEUC implies UC. Strictly speaking SEUC only implies non-uniform EUC, as EUC of a non-uniform π is not defined, but if π is uniform poly-time then non-uniform EUC and EUC are clearly equivalent.

3.1 Composability

We have to show that the composition theorem still holds. Since we have presented the security definition in terms of securely realizing ideal functionalities, we present also composition only in these terms.

Theorem 7 (composition). *Let $\bar{\mathcal{G}}$ be any ideal functionality well-formed for global setup. Let π_1 and π_2 be any $\bar{\mathcal{G}}$-sub-routine respecting protocols. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be ideal functionalities. Assume that π_1 is for the $\mathcal{F}_2$-hybrid model. Let $\pi_1^{\pi_2/\mathcal{F}_2}$ denote the usual EUC composition,* i.e., *it is the protocol which runs as π_1 except that calls to $\mathcal{F}_2$ are replaced by calls to π_2, and when π_1 and π_2 make calls to $\bar{\mathcal{G}}$, they both use the shared functionality $\bar{\mathcal{G}}$.*

If π_1 SEUC realizes $\mathcal{F}_1$ with global setup $\bar{\mathcal{G}}$ and π_2 SEUC realizes $\mathcal{F}_2$ with global setup $\bar{\mathcal{G}}$, then $\pi_2^{\pi_1/\mathcal{F}_1}$ SEUC realizes $\mathcal{F}_1$ with global setup $\bar{\mathcal{G}}$.

Proof. The proof proceeds exactly along the lines of the proofs of the UC and EUC composition. We therefore only sketch the main structure of the proof and refer to [2] for the details. Then we will zoom in on the parts that need modification.

By the premise of the theorem we know that there exist simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that for all $\mathcal{Z}_1$ and $\mathcal{Z}_2$ well-formed for global setup $\bar{\mathcal{G}}$ it holds that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1,\mathcal{Z}_1} \tag{2}$$

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}_2} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}_2} \ . \tag{3}$$

We have to prove that there exists a simulator $\mathcal{S}$ such that for all $\mathcal{Z}$ well-formed for global setup $\bar{\mathcal{G}}$ it holds that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S},\mathcal{Z}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1^{\pi_2/\mathcal{F}_2},\mathcal{Z}} \ . \tag{4}$$

So, consider any environment $\mathcal{Z}$ for $\pi_1^{\pi_2/\mathcal{F}_2}$. From $\mathcal{Z}$, we can construct a series of environments $\mathcal{Z}^{\pi_1,i}$ which are environments for a single instance of π_2 that internally run $\mathcal{Z}$ and π_1 together. Recall that π_1 makes calls to $\mathcal{F}_2$. The j^{th} time π_1 invokes $\mathcal{F}_2$, proceed as follows: If $i < j$, then create internally a new instance π_2^j of π_2 and let π_1 interact with this copy as in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1^{\pi_2/\mathcal{F}_2},\mathcal{Z}}$. This is possible, as each time π_2^j makes a call to $\bar{\mathcal{G}}$, $\mathcal{Z}^{\pi_1,i}$ is free to make the same call, as this instance of π_2 is not the challenge instance. If $i = j$, then make this instance the challenge instance of the game, *i.e.*, in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,i}}$ the environment $\mathcal{Z}^{\pi_1,i}$ will relay all communication to and from π_2^j to the copy π_2 in the game $\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,i}}$. We write $\pi_2^i = \pi_2$. Note that if we run the same environment in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,i}}$, then it is no longer the case that $\pi_2^i = \pi_2$. Instead π_2^i is simulated by $\mathcal{F}_2$ and $\mathcal{S}_2$. With this in mind, if $i > j$, then $\mathcal{Z}^{\pi_1,i}$ will create new copies $\mathcal{S}_2^j$ and $\mathcal{F}_2^j$ of $\mathcal{S}_2$ and $\mathcal{F}_2$ and then let π_1 interact with the simulation $\{\mathcal{F}_2^j, \mathcal{S}_2^j\}$ instead of a copy of π_2, exactly as is done for $i = j$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,i}}$.

It then follows by construction that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,i}} = \mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,i+1}}$$

for all i. It follows from (3) that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,i}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,i}}$$

for all i, as long as each $\mathcal{Z}^{\pi_1,i}$ is well-formed for global setup $\bar{\mathcal{G}}$. From this and transitivity of $\approx_c$ it follows that for any $c \in \mathbb{N}$.

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,k^c}} \ .$$

Using that, by construction and the environment being poly-time, there is a c such that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_2,\mathcal{Z}^{\pi_1,k^c}} = \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1^{\pi_2/\mathcal{F}_2},\mathcal{Z}}$$

we get that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1^{\pi_2/\mathcal{F}_2},\mathcal{Z}} \ .$$

What remains to prove (4) is then to show that there exist a simulator $\mathcal{S}$ such that for all $\mathcal{Z}$ it holds that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S},\mathcal{Z}} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}} \ . \tag{5}$$

Note that in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}}$, the environment never calls the ideal functionality or simulator of the game. It is internally running π_1 and simulating each call to π_2 by running the copies $\{\mathcal{F}_2^j, \mathcal{S}_2^j\}$. We will now refactor this into a protocol π_1 and an environment $\mathcal{Z}_2$ which runs only

the copies $\mathcal{S}_2^j$. Recall that the protocol π_1 makes calls to $\mathcal{F}_2$. *I.e.*, in $\mathsf{exec}_{\pi_1,\mathcal{Z}_1}$ the execution of π_1 will contain many copies of $\mathcal{F}_2$. Call them $\mathcal{F}_2^1, \mathcal{F}_2^2, \ldots$ in the order the instances are created. The environment $\mathcal{Z}_1$ will run $\mathcal{Z}$ internally. Furthermore, for each $\mathcal{F}_2^j$ it will create an instance $\mathcal{S}_2^j$ of the simulator $\mathcal{S}_2$ and let $\mathcal{Z}$ and $\mathcal{S}_2^j$ interact as inside $\mathcal{Z}^{\pi_1,0}$ and it will also let $\mathcal{F}_2^j$ and $\mathcal{S}_2^j$ interact as inside $\mathcal{Z}^{\pi_1,0}$. By construction we get that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\pi_1,\mathcal{Z}_1} = \mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}} \ .$$

If $\mathcal{Z}_1$ is well-formed for global setup $\bar{\mathcal{G}}$, then we get from from (2) that

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1} \approx_c \mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_2,\mathcal{S}_2,\mathcal{Z}^{\pi_1,0}} \ .$$

Recall then that inside the environment $\mathcal{Z}_1$ we are running $\mathcal{Z}$ and several instance $\mathcal{S}_2^j$ of $\mathcal{S}_2$. We can move these $\mathcal{S}_2^j$ to the simulator, as follows. Define $\mathcal{S}$ to be the simulator for the setting $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S},\mathcal{Z}}$ which internally runs $\mathcal{S}_1$ and lets it interact with $\mathcal{F}_1$ exactly as in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1}$. Furthermore, create the instances $\mathcal{S}_2^1$ exactly as $\mathcal{Z}_1$ would have done and let $\mathcal{S}_1$ and the instances $\mathcal{S}_2^j$ interact exactly as the instances inside $\mathcal{Z}_1$ interact with $\mathcal{S}_1$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1}$. And, let the $\mathcal{Z}$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S},\mathcal{Z}}$ and the instances $\mathcal{S}_2^j$ inside $\mathcal{S}$ interact exactly as the instances $\mathcal{S}_2^j$ inside $\mathcal{Z}_1$ interact with the instance of $\mathcal{Z}$ inside $\mathcal{Z}_1$ in $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1}$. By construction

$$\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S},\mathcal{Z}} = \mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{F}_1,\mathcal{S}_1,\mathcal{Z}_1} \ .$$

By transitivity this gives (5).

What remains is to show that if $\mathcal{Z}$ is well-formed for global setup $\bar{\mathcal{G}}$, then also each $\mathcal{Z}^{\pi_1,i}$ and $\mathcal{Z}_1$ as defined above are well-formed for global setup $\bar{\mathcal{G}}$. It is easy to see that they are $\bar{\mathcal{G}}$-externally constrained, as an $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}$ makes no calls to $\bar{\mathcal{G}}$ for the challenge instance *or any of its sub-protocols*.

To see that $\mathcal{Z}_1$ is ignorant, notice that if it is not ignorant, then neither is $\mathcal{Z}$. The reason for this is that the pre-execution phase of $\mathcal{Z}_1$ and $\mathcal{Z}$ ends at the same time, as the pre-execution phase of $\mathcal{Z}_1$ exactly is the pre-execution phase of $\mathcal{Z}$. Hence, given a state of $\mathcal{Z}$ at the end of the pre-execution phase it is also a state of $\mathcal{Z}_1$ at the end of its pre-execution phase, and then we can apply the extractor for $\mathcal{Z}_1$. The formal reduction is a special case of the harder one to come now, so we leave the details to the reader.

We show that $\mathcal{Z}^{\pi_1,i}$ is ignorant, by showing that if $\mathcal{Z}^{\pi_1,i}$ is not ignorant, then neither is $\mathcal{Z}$. Assume that we have an extractor $\mathsf{Extract}_2(\sigma_2)$ taking as input a pre-execution state σ_2 of $\mathcal{Z}^{\pi_1,i}$. We produce an extractor $\mathsf{Extract}(\sigma)$ taking as input a pre-execution state σ of $\mathcal{Z}$. Given σ, notice that trivially σ is a possible state of the copy of $\mathcal{Z}$ inside $\mathcal{Z}_2 = \mathcal{Z}^{\pi_1,i}$. So, $\mathsf{Extract}$ can create an instance of $\mathcal{Z}_2$ and can initialize the state of the $\mathcal{Z}$ inside $\mathcal{Z}_2$ to be σ, *i.e.*, run $\mathcal{Z}(\sigma)$ inside $\mathcal{Z}_2$. Denote the joint state of $\mathcal{Z}_2$ after initializing $\mathcal{Z}$ with σ by $\mathcal{Z}_2(\sigma)$. Note that since $\mathcal{Z}(\sigma)$ so far did not activate any other entity than $\bar{\mathcal{G}}$, neither would $\mathcal{Z}_2(\sigma)$ have done this, as $\mathcal{Z}_2$ only activates other entities than $\bar{\mathcal{G}}$ when the inner $\mathcal{Z}$ does so. Hence $\mathcal{Z}_2(\sigma)$ is a state of $\mathcal{Z}_2$ still in the pre-execution phase. Let $\sigma'_{\bar{\mathcal{G}}}$ denote the state of $\bar{\mathcal{G}}$ at the time where $\mathcal{Z}$ ended the pre-execution phase and produced σ, *i.e.*, the execution was in state $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})}_{\mathcal{Z}(\sigma)}$. Now $\mathsf{Extract}$ can use its black-box access to $\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})$ in $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma)}$ to simulate a run of $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})}_{\mathcal{Z}_2(\sigma)}$ until the pre-execution phase of $\mathcal{Z}_2$ ends. When this happens denote the current state of $\mathcal{Z}_2$ by σ_2 and denote the new state of $\bar{\mathcal{G}}$ by $\sigma_{\bar{\mathcal{G}}}$. We have that $\mathsf{Extract}$ can compute σ_2 simply by inspecting its instance of $\mathcal{Z}_2$. By the pre-execution lemma, we get that producing $\sigma_{\bar{\mathcal{G}}}$ and σ_2 like this gives

them the same distributions as producing them by running $\mathsf{exec}^{\bar{\mathcal{G}}}_{\mathcal{Z}_2}$ until the end of the pre-execution phase. So, from now on $\mathsf{Extract}(\sigma)$ will run exactly as $\mathsf{Extract}_2(\sigma_2)$, and it follows that $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma)}$ and $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}_2(\sigma_2)}$ have the same distribution. In particular, since by assumption $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma_{\bar{\mathcal{G}}})}_{\mathsf{Extract}_2(\sigma_2)}$ will make $\bar{\mathcal{G}}$ output $t = 1$, so will $\mathsf{exec}^{\bar{\mathcal{G}}(\sigma'_{\bar{\mathcal{G}}})}_{\mathsf{Extract}(\sigma)}$. Since the running time of $\mathsf{Extract}$ clearly is polynomial in the running time of $\mathsf{Extract}_2$, the theorem follows. □

4 Separating the EUC and the SEUC Framework

We have argued that SEUC security implies EUC security. In this section we will give an example of a protocol which is EUC secure, but which is not SEUC secure, showing that SEUC security is a strictly stronger security notion than EUC security.

Let $\mathcal{F}$ be the ideal functionality for commitment between S and R, with S being the committer, and with messages from $\{0,1\}^k$. By the results in [3] we can under standard computational assumptions construct a protocol π which EUC securely realizes $\mathcal{F}$ with global setup being an ACRS. Let $\mathcal{S}$ denote the simulator for the case where S is corrupted and R is honest. Note that $\mathcal{S}$ can extract the message from a commitment done by the possibly corrupted S.

Now let $C = \{(crs_k, r_k)\}_{k\in\mathbb{N}}$ be a family of ACRS's, where crs_k is a random ACRS set up by $\mathcal{S}$ for security level k and r_k is the randomness used to generate crs_k. Let π_C be the non-uniform protocol, where the sender S in $\pi(k)$ runs exactly as π, except that if the ACRS crs returned by the shared ideal functionality $\bar{\mathcal{G}}$ is exactly $crs = crs_k$, where k is the security parameter, the sender will at the end of the commitment phase send the committed message m to R in plaintext.

It is clear that π_C is not SEUC secure. The environment $\mathcal{Z}$ will at security level k let $\bar{\mathcal{G}}$ use randomness r_k when generating crs, leading to $crs = crs_k$. It can "guess" r_k via its auxiliary input z. Then it will ask S to commit to a random m and inspect the communication between S and R to see if m is sent at the end of the commitment phase. If so, it guesses that it is in the real world. Otherwise, it guesses that it is in the ideal world. Since the simulator in the ideal world gets no information on m, this strategy will clearly distinguish.

It is also clear that π_C is non-uniform EUC secure. Assume namely that it was not. Then it must clearly be because $crs = crs_k$ with non-negligible probability. However, if this was the case then $\mathcal{Z}$ could attack the honest sender in the underlying protocol π. Namely, we can without loss of generality assume that at security level k, the environment has crs_k and r_k hardcoded (it can guess them via z) and that it knows the code of $\mathcal{S}$. It can then corrupt R when it happens that $\bar{\mathcal{G}}$ sets up $crs = crs_k$. Then $\mathcal{Z}$ asks S to commit to a uniformly random m and it lets the honest S interact with the corrupted receiver $\mathcal{S}(r_k)$. In the real world this leads to $\mathcal{S}$ extracting m except with negligible probability. In the ideal world the simulator of S has no information on m, so the extraction will yield m with probability at most 2^{-k}.

Above we present a protocol which is secure in the EUC model but insecure in the SEUC model. As elaborated on in the introduction, we believe that π_C should be consider insecure. Protocols with hard coded bad ACRS's could have been designed with the current value of the ACRS(s) in mind and hence be insecure. A formal security notion cannot make references to such real world phenomena as *the current value of the ACRS(s)*. We therefore have to consider π_C insecure. We believe that this shows that the EUC model is unsound relative to practice.

Since we cannot make formal statements relating to the world, we cannot prove that the SEUC model is sound. The above example, however, shows that at least it seems to be more sound relative to practice than the EUC model.

5 Fixing the Protocol

The original GUC paper presented a GUC-secure commitment scheme named UAIBC (UC Adaptive Identity-Based Commitment) based on a global functionality providing an augmented CRS. We examine this scheme and find that the construction is sound and can be proven secure in our model, although we need to make slightly stronger assumptions on some of the primitives. These stronger assumptions are necessary, since the environment can choose the randomness which is the master secret key of the global functionality. This needs to be reflected in the properties of the building blocks.

We believe that proving this scheme secure in our model can be instructive and shed some light on the differences to GUC. We retrace the construction and proofs and highlight and motivate the differences.

5.1 Identity-based Trapdoor Commitments from Σ-Protocols

The main building block of UAIBC is an identity-based trapdoor commitment (IBTC), which is based on Σ-protocols for EUF-CMA signatures. We first define Σ-protocols and some properties, then define what we expect from the IBTC, and finally show that the IBTC construction from [3] is secure according to the stronger security notions we define, under the original security assumptions on the Σ-protocol. This immediately implies that the existence of OWFs is sufficient for the existence of IBTCs fulfilling our security notions. [3, Sec. 5.2.3]

Σ-Protocols

Definition 8 (Σ-Protocol). *Let X, W and $L \subset X$ be sets, R a relation for L such that $\forall x \in L \, \exists w \in W : (x, w) \in R$, and R is efficiently testable (in time polynomial in $|x|$). An* augmented Σ-protocol *for the language L is a tuple of deterministic poly-time algorithms (A, Z, B) run between a prover $P(x, w; r_a)$ and a verifier $V(x; c)$ (where $|c| \leq poly(|x|)$) in the following way.*

1. *P sends $a \leftarrow A(x, w; r_a)$*
2. *V sends c*
3. *P sends $z \leftarrow Z(x, w, c; r_a)$*
4. *V outputs $B(x, a, c, z)$.*

The completeness *condition must hold: If $(x, w) \in R$, then after an interaction with $P(x, w)$, $V(x)$ outputs 1 with overwhelming probability.*

[3] defines three security properties for Σ-protocols: Special HVZK (Honest Verifier Zero Knowledge), reverse state construction (RSC) and special soundness.

The first property models the expectation that even without the witness, one can produce a transcript indistinguishable from an actual protocol run. The indistinguishability even holds if the distinguisher chooses the witness; this is the reason why special HVZK is stronger than HVZK.

Definition 9 (Special HVZK). *A Σ-protocol $\Pi = (A, Z, B)$ is special HVZK, if there exists an algorithm $\mathsf{ZKSim} \in \mathcal{PPT}$ such that for any PPT distinguisher D, the advantage $\mathrm{Adv}^{\mathsf{hvzk}}_{\Pi,\mathsf{ZKSim},D}(k)$ of D in the experiment $\mathrm{Exp}^{\mathsf{hvzk}-b}_{\Pi,\mathsf{ZKSim},D}(k)$ defined below is negligible.*

1. $(x, w, c) \leftarrow D(1^k)$
2. $r \xleftarrow{\$} \{0,1\}^{poly(k)}; a_0 \leftarrow A(x, w; r); z_0 \leftarrow Z(x, w, c; r); (a_1, z_1) \leftarrow \mathsf{ZKSim}(x, c; r)$
3. $\hat{b} \leftarrow D(a_b, z_b)$

We define the advantage to be

$$\mathrm{Adv}^{\mathsf{hvzk}}_{\Pi,\mathsf{ZKSim},D}(k) = |\Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{hvzk}-1}_{\Pi,\mathsf{ZKSim},D}(k)] - \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{hvzk}-0}_{\Pi,\mathsf{ZKSim},D}(k)]|.$$

As in [3], we require ZKSim to either output an error symbol, or a pair (a, z) that causes B to output 1 (accept).

The next property ensures that it is possible to find random coins after a protocol run such that the coins and the transcript could also have come from a simulator ZKSim. This implies special HVZK. Again, the indistinguishability even holds if the distinguisher chooses the witness.

Definition 10 (RSC). *A Σ-protocol Π has the RSC property, if there exists an algorithm $\mathsf{RSC} \in \mathcal{PPT}$ such that for any PPT distinguisher D, the advantage $\mathrm{Adv}^{\mathsf{rsc}}_{\Pi,\mathsf{RSC},D}(k)$ of D in the experiment $\mathrm{Exp}^{\mathsf{rsc}-b}_{\Pi,\mathsf{ZKSim},D}(k)$ defined below is negligible.*

1. $(x, w, c) \leftarrow D(1^k)$
2. $r_a, r_1 \xleftarrow{\$} \{0,1\}^{poly(k)}; a_0 \leftarrow A(x, w; r_a); z_0 \leftarrow Z(x, w, c; r_a); r_0 \leftarrow \mathsf{RSC}(x, w, c; r_a); (a_1, z_1) \leftarrow \mathsf{ZKSim}(x, c; r_1)$
3. $\hat{b} \leftarrow D(a_b, z_b, r_b)$

We define the advantage to be

$$\mathrm{Adv}^{\mathsf{rsc}}_{\Pi,\mathsf{RSC},D}(k) = |\Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{rsc}-1}_{(\Pi,\mathsf{ZKSim},D}(k)] - \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{rsc}-0}_{\Pi,\mathsf{ZKSim},D}(k)]|.$$

The last property guarantees the existence of a successful rewinding extractor.

Definition 11 (Special soundness). *A Σ-protocol $\Pi = (A, Z, B)$ has special soundness, if there exists a rewinding extractor $E \in \mathcal{PPT}$ such that for any PPT distinguisher D, the success probability $\mathsf{Succ}^{\mathsf{spsound}}_{\Pi,E,D}$ of D in the experiment $\mathrm{Exp}^{\mathsf{spsound}}_{\Pi,E,D}(k)$ defined below is negligible.*

1. $(x, a, c, z, c', z') \leftarrow D(1^k)$
2. $w \leftarrow E(x, a, c, z, c', z')$
3. *if* $c \neq c' \wedge (B(x, a, c, z) = B(x, a, c', z') = 1) \wedge (x, w) \notin R$ *then return 1, else return 0.*

We define the success probability to be

$$\mathsf{Succ}^{\mathsf{spsound}}_{\Pi,E,D}(k) = \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{spsound}}_{\Pi,E,D}(k)].$$

IBTC An IBTC is a commitment scheme where a committer who knows the receiver's secret key can equivocate the commitment. We recall the definition of IBTC from [3], where a party decommits by sending the randomness used to commit.

Definition 12 (IBTC). *An identity-based trapdoor commitment is a five-tuple of PPT algorithms $\mathcal{IC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Com}, \mathsf{ECom}, \mathsf{Eqv})$. We additionally specify the set $\mathcal{D}$ from which the randomness used to commit and decommit is drawn.*

- Setup *takes as input $\mathit{MSK} \in \{0,1\}^k$ and outputs a public key PK.*
- KeyGen *on input $(\mathit{MSK}, \mathit{PK}, \mathit{id})$ outputs $\mathit{sk}_{\mathit{id}}$.*
- Com *on input $(\mathit{PK}, \mathit{id}, m; d)$ outputs a commitment κ to the message m for identity id using randomness $d \in \mathcal{D}$.*
- ECom *on input $(\mathit{sk}_{\mathit{id}}, \mathit{PK}, \mathit{id})$ outputs (κ, α) to be used with* Eqv.
- Eqv *on input $(\mathit{sk}_{\mathit{id}}, \mathit{PK}, \mathit{id}, \kappa, \alpha, m)$ produces $d \in \mathcal{D}$ such that $\kappa = \mathsf{Com}(\mathsf{PK}, \mathsf{id}, m; d)$.*

Since the commitment randomness is used to decommit, correctness is trivial. [3] defines two properties for IBTC: binding and equivocability. We need the binding property for the well-formedness of the global setup functionality, but need to define a different but related property for the security proof, which we call key extractability. Key extractability is incomparable to binding: On the one hand, the adversary can choose the master secret key, on the other hand, it needs to additionally beat any extractor.

Definition 13 (Binding). *We define the following experiment* $\mathrm{Exp}^{\mathsf{bind}}_{\mathcal{IC},\mathcal{A}}(k)$ *for an adversary* $\mathcal{A}$ *against an IBTC* $\mathcal{IC}$*.*

1. *The challenger draws* $\mathit{MSK} \xleftarrow{\$} \{0,1\}^k$, $\mathit{PK} = \mathsf{Setup}(\mathsf{MSK})$ *and sends* PK *to* $\mathcal{A}$*.*
2. $(\mathit{id}, d, m, d', m') \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{MSK},\mathsf{PK},\cdot)}(\mathit{PK})$*.*
3. *if* id *was not queried to the* KeyGen *oracle,* $m \neq m'$*, and* $\mathsf{Com}(\mathsf{PK},\mathsf{id},m;d) = \mathsf{Com}(\mathsf{PK},\mathsf{id},m';d')$ *then return 1, else return 0.*

$\mathcal{IC}$ *is binding if the success probability of any* $\mathcal{A} \in \mathcal{PPT}$ *is negligible in* k*.*

$$\mathsf{Succ}^{\mathsf{bind}}_{\mathcal{IC},\mathcal{A}}(k) = \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{bind}}_{\mathcal{IC},\mathcal{A}}(k)].$$

Definition 14 (Key extractability). *An IBTC* $\mathcal{IC}$ *is key extractable if there is a PPT extractor* E *that outputs the secret key* $\mathit{sk}_{\mathit{id}}$ *of user* id *on input a tuple* $(\mathit{PK}, \mathit{id}, d, m, d', m')$*,* $m \neq m'$*, for which* $\mathsf{Com}(\mathsf{PK},\mathsf{id},m;d) = \mathsf{Com}(\mathsf{PK},\mathsf{id},m';d')$*. This should hold even if the adversary chooses* MSK *and therefore* PK*. We define the following experiment* $\mathrm{Exp}^{\mathsf{key\text{-}extr}}_{\mathcal{IC},E,\mathcal{A}}(k)$*.*

1. $(\mathit{MSK}, \mathit{id}, d, m, d', m') \leftarrow \mathcal{A}(1^k)$
2. $\mathit{PK} = \mathsf{Setup}(\mathsf{MSK})$
3. *if* $m \neq m' \wedge \mathsf{Com}(\mathsf{PK},\mathsf{id},m;d) = \mathsf{Com}(\mathsf{PK},\mathsf{id},m';d') \wedge E(\mathsf{PK},\mathsf{id},d,m,d',m') \notin [\mathsf{KeyGen}(\mathsf{MSK},\mathsf{PK},\mathsf{id})]$ *then return 1, else return 0.*

$\mathcal{IC}$ *is key extractable, if the success probability of any* $\mathcal{A} \in \mathcal{PPT}$ *against our extractor is negligible in* k*.*

$$\mathsf{Succ}^{\mathsf{key\text{-}extr}}_{\mathcal{IC},E,\mathcal{A}}(k) = \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{key\text{-}extr}}_{\mathcal{IC},E,\mathcal{A}}(k)].$$

We strengthen the notion of equivocability by changing one detail: The adversary $\mathcal{A}$ can choose MSK himself instead of receiving it from the challenger.

Definition 15 (Strong equivocability). *We define the following experiment* $\mathrm{Exp}^{\mathsf{equivoc}-b}_{\mathcal{IC},\mathcal{A}}(k)$*.*

1. $(\mathit{MSK}, \mathit{id}, m) \leftarrow \mathcal{A}(1^k)$
2. $d_0 \xleftarrow{\$} \mathcal{D}$; $\mathit{PK} = \mathsf{Setup}(\mathsf{MSK})$; $\mathit{sk}_{\mathit{id}} \leftarrow \mathsf{KeyGen}(\mathsf{MSK},\mathsf{PK},\mathsf{id})$; $(\kappa,\alpha) \leftarrow \mathsf{ECom}(\mathsf{sk}_{\mathsf{id}},\mathsf{PK},\mathsf{id})$; $d_1 \leftarrow \mathsf{Eqv}(\mathsf{sk}_{\mathsf{id}},\mathsf{PK},\mathsf{id},\kappa,\alpha,m)$*.*
3. $\hat{b} \leftarrow \mathcal{A}(d_b)$*.*

$\mathcal{IC}$ *is strongly equivocable, if the advantage of any* $\mathcal{A} \in \mathcal{PPT}$ *is negligible in* k*.*

$$\mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IC},\mathcal{A}}(k) = |\Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{equivoc}-1}_{\mathcal{IC},\mathcal{A}}(k)] - \Pr[1 \leftarrow \mathrm{Exp}^{\mathsf{equivoc}-0}_{\mathcal{IC},\mathcal{A}}(k)]|.$$

Just to have everything in one place, we recall the construction of IBTC from [3]. The construction of the IBTC is as follows: The CRS is the verification key of a signature scheme, party secret keys are signatures on the party identifier. To open, the sender proves that either the revealed value is the committed value, or the sender knows a signature on the receiver's pid.

Definition 16. *Let $\Upsilon = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme.*[6] *We define the relation R^{Υ} such that $(x, w) \in R^{\Upsilon}$ if $x = (\mathit{vk}, m)$ and $w = \sigma$ such that $\Upsilon.\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$ and let Π_{Σ} be a Σ-protocol for R^{Υ}. We define the following identity-based trapdoor commitment $\mathcal{IBTC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Com}, \mathsf{ECom}, \mathsf{Eqv})$, where the set $\mathcal{D}$ is the randomness used by $\Pi_{\Sigma}.\mathsf{ZKSim}$.*

- Setup *on input $\mathit{MSK} \in \{0,1\}^k$ runs $(\mathit{vk}, \sigma k) \leftarrow \Upsilon.\mathsf{KeyGen}(1^k; \mathsf{MSK})$ and outputs the public key $\mathit{PK} = \mathit{vk}$.*
- KeyGen *on input $(\mathit{MSK}, \mathit{PK}, \mathit{id})$ runs $\Upsilon.\mathsf{KeyGen}(1^k; \mathsf{MSK})$ to obtain σk and outputs $\mathit{sk}_{\mathit{id}} = \Upsilon.\mathsf{Sign}(\sigma\mathsf{k}, \mathsf{id})$.*
- Com *on input $(\mathit{PK}, \mathit{id}, m; d)$ computes $(a, z) \leftarrow \Pi_{\Sigma}.\mathsf{ZKSim}(x = \langle \mathsf{PK}, \mathsf{id} \rangle, c = m; d)$ and outputs a commitment $\kappa = a$.*
- ECom *on input $(\mathit{sk}_{\mathit{id}}, \mathit{PK}, \mathit{id})$ draws $r_a \xleftarrow{\$} \{0,1\}^{poly(k)}$, computes $a = A(x = (\mathit{PK}, \mathit{id}), w = \mathit{sk}_{\mathit{id}}; r_a)$ and outputs $(\kappa, \alpha) = (a, r_a)$ to be used with Eqv.*
- Eqv *on input $(\mathit{sk}_{\mathit{id}}, \mathit{PK}, \mathit{id}, \kappa, \alpha, m)$ computes $r_s = \mathsf{RSC}(x = \langle \mathsf{PK}, \mathsf{id} \rangle, w = \mathsf{sk}_{\mathsf{id}}, c = m; r_a = \alpha)$ and returns $d = r_s$.*

We prove that $\mathcal{IBTC}$ fulfills our stronger notions under the original assumptions on the Σ-protocol. That is, we prove stronger properties for the same protocol using the same building blocks.

Theorem 17. *$\mathcal{IBTC}$ is binding, key extractable, and strongly equivocable, if the underlying signature scheme Υ is EUF-CMA, and Π_{Σ} has special soundness and the reverse state construction property.*

The theorem follows from the following three lemmata.

Lemma 18 (Th. 5.2 in [3]). *$\mathcal{IBTC}$ is binding if the underlying signature scheme Υ is EUF-CMA.*

$$\forall \mathcal{A} \in \mathcal{IPT} \exists \mathcal{B} \in \mathcal{IPT} \left(\mathrm{Adv}^{\mathsf{bind}}_{\mathcal{IBTC}, \mathcal{A}} \leq \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\Upsilon, \mathcal{B}} \right).$$

Lemma 19. *$\mathcal{IBTC}$ is key extractable if the Σ-protocol Π_{Σ} has special soundness.*

$$\forall \mathcal{A} \in \mathcal{IPT} \exists \mathcal{B} \in \mathcal{IPT} \left(\mathsf{Succ}^{\mathsf{key\text{-}extr}}_{\mathcal{IBTC}, \mathcal{A}} \leq \mathsf{Succ}^{\mathsf{spsound}}_{\Pi_{\Sigma}, \mathcal{B}} \right).$$

Proof. Let $\mathcal{A}$ be a successful adversary against the key extractability of $\mathcal{IBTC}$. We construct an adversary $\mathcal{B}$ against the special soundness of Π_{Σ}. Special soundness of Π_{Σ} means that there is an extractor E_{Π} such that for any two polynomially computable accepting transcripts with different c, z, E_{Π} forges a signature. We show that any good extractor for Π_{Σ} can be turned into a good key extractor for $\mathcal{IBTC}$.

On input 1^k, $\mathcal{A}$ returns a tuple $(\mathsf{PK}, \mathsf{id}, d, m, d', m')$ and with non-negligible probability $m \neq m'$, $\mathsf{Com}(\mathsf{PK}, \mathsf{id}, m; d) = \mathsf{Com}(\mathsf{PK}, \mathsf{id}, m'; d')$. $\mathcal{B}$ computes $(\kappa, z) \leftarrow \mathsf{ZKSim}(\langle \mathsf{PK}, \mathsf{id} \rangle, m, d)$ and $(\kappa', z') \leftarrow \mathsf{ZKSim}(\langle \mathsf{PK}, \mathsf{id} \rangle, m', d')$ and we have $\kappa = \kappa'$. $\mathcal{B}$ sets $x = \langle \mathsf{PK}, \mathsf{id} \rangle$ and sends $(x, \kappa, m, z, m', z')$ to the challenger.

The challenger computes $w \leftarrow E_{\Pi}(x, \kappa, m, z, m', z')$. If $\mathsf{Verify}(\mathsf{vk}, \mathsf{id}, w) = 1$, then $\sigma\mathsf{k}_{\mathsf{id}} = w$. Else, the challenger uses w to generate a key pair and signs id. Then he outputs the secret key of id. We have $B(x, a, c, z) = B(x, a, c', z') = 1$ by the correctness of ZKSim, and if $(x, w) \in R^{\Upsilon}$, then $\sigma\mathsf{k}_{\mathsf{id}} \in [\mathsf{KeyGen}(\mathsf{MSK}, \mathsf{PK}, \mathsf{id})]$ for a[7] MSK for which $\mathsf{PK} = \mathsf{Setup}(\mathsf{MSK})$.

If $(x, w) \in R^{\Upsilon}$, then given a successful E_{Π}, there is an extractor for $\mathcal{IBTC}$ with $w \in [\mathsf{KeyGen}(\mathsf{MSK}, \mathsf{PK}, \mathsf{id})]$. □

[6] We require that on input 1^k, KeyGen uses k bits of randomness. We can transform any scheme where KeyGen uses more randomness into a scheme fulfilling this definition by using a PRG to expand a k-bit seed.

[7] If Setup is not injective, there may be several such MSK

Lemma 20. *$\mathcal{IBTC}$ is strongly equivocable if the reverse state construction property of the Σ-protocol holds.*

$$\forall \mathcal{A} \in \mathcal{IPT} \exists \mathcal{B} \in \mathcal{IPT} \left(\mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IBTC},\mathcal{A}} \leq \mathrm{Adv}^{\mathsf{hvzk}}_{\Pi_\Sigma,\mathcal{B}} \right).$$

Proof. Let $\mathcal{A}$ be a successful adversary against the strong equivocability of $\mathcal{IBTC}$. We construct an adversary $\mathcal{B}$ against the reverse state construction of Π_Σ. On input 1^k, $\mathcal{A}$ returns a tuple $(\mathsf{MSK}, \mathsf{id}, m)$. $\mathcal{B}$ computes $\mathsf{PK} = \mathsf{Setup}(\mathsf{MSK})$ and $\mathsf{sk_{id}} = \mathsf{KeyGen}(\mathsf{MSK}, \mathsf{PK}, \mathsf{id})$, sets $x = (\mathsf{PK}, \mathsf{id})$, $w = \mathsf{sk_{id}}$, $c = m$, and outputs (x, w, c).

He receives the challenge r_b, which is either r_1, which is is drawn at random, or $r_0 \leftarrow \mathsf{RSC}(x, w, c) = \mathsf{Eqv}(\mathsf{sk_{id}}, \mathsf{PK}, \mathsf{id}, \kappa, \alpha, m)$ for $(\kappa, \alpha) = \mathsf{Com}(\mathsf{PK}, \mathsf{id}, m)$. This is exactly what $\mathcal{A}$ expects to see, so $\mathcal{B}$ sends r_b to $\mathcal{A}$ and forwards the guess bit $\hat{b}$.

5.2 The Augmented CRS Model

To match our definition, we first need to modify $\bar{\mathcal{G}}_{acrs}$ so that it answers queries of the form $(\texttt{trapdoor?}, x)$. $\bar{\mathcal{G}}^{\mathsf{Setup},\mathsf{KeyGen}}_{acrs}$ is parametrized by two functions Setup and KeyGen, which are part of an IBTC, and takes as input the security parameter k.

- Initialization phase: At the first activation, draw $\mathsf{MSK} \xleftarrow{\$} \{0,1\}^k$ and compute a CRS $\mathsf{PK} = \mathsf{Setup}(\mathsf{MSK})$, then record the pair $(\mathsf{PK}, \mathsf{MSK})$.
- Providing the public value: Upon receipt of a message $(\texttt{crs})$ by any party P (or the adversary), return PK to the requesting party (respectively the adversary).
- Dormant phase: Upon receipt of a message $(\texttt{retrieve}, \mathsf{sid}, \mathsf{pid})$ from a *corrupt* party P with PID pid, return the value $\mathsf{sk_{pid}} = \mathsf{KeyGen}(\mathsf{MSK}, \mathsf{PK}, \mathsf{pid})$ to P. (Receipt of this message from honest parties is ignored.)
- Trapdoor test: Upon receipt of a message $(\texttt{trapdoor?}, (x, \mathsf{pid}))$ from $\mathcal{Z}$, set $t = 1$ if $x = \mathsf{KeyGen}(\mathsf{MSK}, \mathsf{PK}, \mathsf{pid})$ and party pid has not been corrupted, $t = 0$ otherwise. Return $(\texttt{trapdoor}, (x, \mathsf{pid}), t)$ to $\mathcal{Z}$.

We have to be able to check whether x is a trapdoor. Since, as in [3], KeyGen is a function, it is deterministic.

Well-formedness of Setup We first have to show that the modified functionality $\bar{\mathcal{G}}_{acrs}$ is well formed according to Definition 1. We show that if the IBTC were not well-formed, it would not be binding.

Theorem 21.

$$\forall \mathcal{Z} \in \mathcal{IPT}\ \exists \mathcal{B} \in \mathcal{IPT} : \Pr[((\texttt{trapdoor}, \cdot, 1) \leftarrow \mathsf{exec}_{\bar{\mathcal{G}}^{\mathcal{IBTC}}_{acrs},\mathcal{Z}}(k)] \leq q(k) \mathrm{Adv}^{\mathsf{bind}}_{\mathcal{IBTC},\mathcal{B}}(k)$$

where $\mathcal{Z}$ asks at most $q(k)$ $\texttt{trapdoor?}$ queries.

Proof. Let $\mathcal{Z}$ be an IPT ITM which asks at most $q(k)$ $\texttt{trapdoor?}$ queries and makes $\bar{\mathcal{G}}_{acrs}$ output a message of the form $(\texttt{trapdoor}, \cdot, 1)$ with nonnegligible probability. Then we construct an adversary $\mathcal{B}$ which uses $\mathcal{Z}$ to break the binding property of the IBTC by simulating $\bar{\mathcal{G}}_{acrs}$. $\mathcal{B}$ receives PK from its challenger and uses it to answer all $\texttt{crs}$ queries. If $\mathcal{B}$ receives a $\texttt{retrieve}$ query for party pid, it queries pid to its KeyGen oracle.

$\mathcal{B}$ draws a value $i \xleftarrow{\$} [1, q(k)]$. If $\mathcal{B}$ receives a query of the form $(\texttt{trapdoor?}, (x, \mathsf{pid}))$ where pid is corrupted, he returns 0. On the i-th query, if pid is not corrupted, $\mathcal{B}$ computes $(\kappa, \alpha) = \mathsf{ECom}(\mathsf{PK}, \mathsf{pid}, x)$, chooses two different messages m_0, m_1, obtains $d_b = \mathsf{Eqv}(\mathsf{PK}, \mathsf{pid}, x, \kappa, \alpha, m_b)$ for $b = 0, 1$ and outputs $(\mathsf{pid}, d_0, m_0, d_1, m_1)$. Clearly, $\mathcal{B}$ wins if x was a valid secret key for party pid.

If $\mathcal{Z}$ wins the game, it has to output at least one valid secret key, which is used by $\mathcal{B}$ with probability at least $1/q(k)$.

5.3 UC Adaptive Identity-Based Commitments from IBTC and Dense PRC

We are now ready to present the construction of UC Adaptive Identity-Based Commitments (UAIBC) from IBTC in the augmented CRS model. UC commitments need to be equivocable and extractable. Extractability is added to the IBTC by first executing a coin flipping protocol to choose the public key for an encryption scheme. This necessitates the use of a special kind of PKE, which we present first.

PKE In the definition of dense PRC encryption, we follow a different approach by building on the definition of PKE in two steps. This makes it possible to point out more clearly which properties we will reduce to in the proof.

Definition 22 (PKE). *A public-key encryption scheme is a three-tuple of PPT algorithms* $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with a message space* $\mathcal{M}$*, which fulfills the correctness definition in Def. 23.*

- Gen *takes as input the security parameter* k *in unary and outputs a key pair* $(\mathsf{ek}, \mathsf{dk})$*. We suppose that* ek *and* dk *are of length at least* k*.*
- Enc *takes as input a public key* ek *and a message* $m \in \mathcal{M}$ *and outputs a ciphertext* c*.*
- Dec *is deterministic and takes as input a secret key* dk *and a ciphertext* c *and outputs a message* m *or an error symbol* $\perp$*.*

Definition 23 (Correctness). *We say that a PKE scheme* $\mathcal{PKE}$ *is* correct*, if the decryption error*

$$\varepsilon^{\mathsf{corr}}_{\mathcal{PKE}}(k) = \sup_{(\mathsf{pk},\mathsf{sk})\in[\mathsf{Gen}(1^k)],m\in\mathcal{M}} \Pr_r[\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m; r)) \neq m]$$

is negligible.

Definition 24 (Dense PKE). *For a PKE scheme* $\mathcal{PKE}$*, we denote the set of possible public keys for security parameter* k *with* Φ_k*, or just* Φ*, if* k *is clear from the context. We say that* $\mathcal{PKE}$ *is* dense*, if the following conditions hold.*

1. *Φ is an abelian group with efficiently computable inverse and group operations.*
2. *Membership in Φ is efficiently testable.*
3. *The uniform distribution on Φ is efficiently samplable.*
4. *The distribution of public keys produced by* Gen *is computationally indistinguishable from the uniform distribution on Φ:*

$$\mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE},\mathcal{A}}(k) = \Pr[(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^k); 1 \leftarrow \mathcal{A}(1^k, \mathsf{ek})] - \Pr[\mathsf{ek} \xleftarrow{\$} \Phi; 1 \leftarrow \mathcal{A}(1^k, \mathsf{ek})]$$

is negligible for all $\mathcal{A} \in \mathcal{PPT}$*.*

Definition 25 (Dense PRC). *A dense PKE scheme* $\mathcal{PKE}$ *is said to have pseudo-random ciphertexts (PRC), if*

$$\mathrm{Adv}^{\mathsf{prc}}_{\mathcal{PKE},\mathcal{A}}(k) = \Pr[1 \leftarrow \mathcal{A}^{\mathsf{LOR}(\cdot,0)}(1^k)] - \Pr[1 \leftarrow \mathcal{A}^{\mathsf{LOR}(\cdot,1)}(1^k)]$$

is negligible for all $\mathcal{A} \in \mathcal{PPT}$*, where* $\mathsf{LOR}(m, 0)$ *computes* $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^k)$*, and returns* $(\mathsf{ek}, \mathsf{Enc}_{\mathsf{ek}}(m))$ *and* $\mathsf{LOR}(m, 1)$ *draws* $\mathsf{ek} \xleftarrow{\$} \Phi$*,* $c \xleftarrow{\$} [\mathsf{Enc}_{\mathsf{ek}}(m)]$[8] *and returns* (ek, c)*.*

[8] We slightly relax the definition from [3], which allows us to capture ElGamal encryption.

The UAIBC protocol builds on identity-based trapdoor commitments (IBTC) to securely implement $\mathcal{F}_{com}$.

UAIBC adds extractability to realize $\mathcal{F}_{com}$ against adaptive adversaries. Both parties perform a coin toss to get a PKE encryption key using the IBTC. Then the sender commits to its bit using the IBTC and additionally encrypts the decommitment information if its bit is 0, else a random string.

Definition 26. *Let $\mathcal{IBTC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Com}, \mathsf{ECom}, \mathsf{Eqv})$ be an identity-based trapdoor commitment with randomness in $\mathcal{D}$, $\mathcal{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ a dense PKE scheme with public keys in Φ, ciphertexts in $\mathcal{C}$ and randomness in $\mathcal{R}$. The $\mathcal{UAIBC}$ protocol is run between a committer C, who wants to commit to a bit b, and a receiver R. Both have access to a functionality $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{KeyGen}}$. Then $\mathcal{UAIBC}[\mathcal{IBTC}, \mathcal{PKE}]$ is the protocol with the following three phases:*

Initialization Phase *Both R and C are initialized with the CRS PK produced by $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{KeyGen}}$, which is a public key for $\mathcal{IBTC}$.*

Commit Phase *The commit phase begins when C receives its input $(\mathtt{commit}, \mathit{sid}, \mathit{pid}_C, \mathit{pid}_R, b)$ from the environment. All messages are sent over a secure channel and prefixed with a flow identifier, $\mathit{sid}, \mathit{pid}_C, \mathit{pid}_R$.*

1. $C \to R : \mathtt{commit}, \mathit{sid}, \mathit{pid}_C, \mathit{pid}_R$
2. R
   (a) $\rho_1 \xleftarrow{\$} \Phi$
   (b) $d_1 \xleftarrow{\$} \mathcal{D}$
   (c) $\kappa_1 = \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_C, \rho_1; d_1)$
   (d) $R \to C : \kappa_1$
3. C
   (a) $\rho_2 \xleftarrow{\$} \Phi$
   (b) $C \to R : \rho_2$
4. R
   (a) $\rho = \rho_1 \cdot \rho_2$
   (b) $R \to C : d_1, \rho_1$
5. C
   (a) $\kappa_1 \stackrel{?}{=} \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_C, \rho_1; d_1)$
   (b) $\rho = \rho_1 \cdot \rho_2$
   (c) $d \xleftarrow{\$} \mathcal{D}$
   (d) $\kappa = \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_R, b; d)$
   (e) $\mathtt{if}\ \ b = 0\ \mathtt{then}\ \ r \xleftarrow{\$} \mathcal{R}; \varphi = \mathsf{Enc}_\rho(d; r)\ \mathtt{else}\ \ \varphi \xleftarrow{\$} \mathcal{C}$
   (f) $C \to R : \kappa, \varphi$

At the end of the commit phase, R outputs $(\mathtt{receipt}, \mathit{sid}, \mathit{pid}_C, \mathit{pid}_R)$.

Reveal Phase *The reveal phase begins when C receives its input $(\mathtt{reveal}, \mathit{sid})$ from the environment. All messages are sent over a secure channel and prefixed with a flow identifier, $\mathit{sid}, \mathit{pid}_C, \mathit{pid}_R$.*

1. $C \to R : d, b$ *and* r, *if* $b = 0$.
2. R *checks* $\kappa \stackrel{?}{=} \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_R, b; d)$ *and, if* $b = 0$, $\varphi \stackrel{?}{=} \mathsf{Enc}_\rho(d; r)$. *At the end of the reveal phase, R outputs $(\mathtt{reveal}, \mathit{sid}, \mathit{pid}_C, \mathit{pid}_R, b)$ if all checks pass.*

Inspection confirms that $\mathcal{IBTC}$ is $\bar{\mathcal{G}}_{acrs}$-subroutine respecting.

5.4 Security of the UAIBC Protocol

Theorem 27. *$\mathcal{UAIBC}[\mathcal{IBTC}, \mathcal{PKE}]$ GUC-realizes $\mathcal{F}_{com}$ given access to $\bar{\mathcal{G}}_{acrs}$, if $\mathcal{IBTC}$ is binding and equivocable and $\mathcal{PKE}$ is a dense PRC secure. Party corruptions can be adaptive as long as they are PID-wise. More exactly, if we define $\mathfrak{Z}$ to be the set of non-uniform IPT environments $\mathcal{Z}$ which are well-formed for $\bar{\mathcal{G}}_{acrs}$ according to definition 4, we have*

$$\forall \mathcal{A} \in \mathcal{IPT}\, \exists \mathcal{S} \in \mathcal{IPT}\, \forall \mathcal{Z} \in \mathfrak{Z}\, \exists \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6 \in \mathcal{IPT}$$
$$\Delta(\mathsf{exec}^{\bar{\mathcal{G}}_{acrs}}_{\mathcal{UAIBC}[\mathcal{IBTC},PKE],\mathcal{A},\mathcal{Z}}, \mathsf{exec}^{\bar{\mathcal{G}}_{acrs}}_{\mathcal{F}_{com},\mathcal{S},\mathcal{Z}}) \leq$$
$$4 \cdot \left(\mathrm{Adv}^{\mathsf{prc}}_{\mathcal{PKE},\mathcal{B}_1} + \mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE},\mathcal{B}_2}\right)^2 + \mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE},\mathcal{B}_3}$$
$$+ \mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IBTC},\mathcal{B}_4} + \mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IBTC},\mathcal{B}_5} + \mathsf{Succ}^{\mathsf{key\text{-}extr}}_{\mathcal{IBTC},\mathcal{B}_6} + \varepsilon^{\mathsf{corr}}_{\mathcal{PKE}}.$$

Proof. We first consider the pre-execution phase, in which $\mathcal{Z}$ determines the random coins of $\bar{\mathcal{G}}_{acrs}$. This allows the environment to fix $\mathsf{MSK}, \mathsf{PK}$, and the first q_1 keys of corrupted parties, *i.e.*, it can query to get the keys of corrupted parties, and it can then specify the randomness used by the ideal functionality when these keys are computed. We let q_1 denote the number of such keys. The rest of the proof proceeds as in the original paper.

We proceed by game-hopping. Let I_0 be the $\bar{\mathcal{G}}_{acrs}$-hybrid real-world setting.

We first introduce a conceptual change. In game I'_0, we magically provide C with sk_R if R is corrupt, and R with sk_C if C is corrupt. This does not change the output distribution of $\mathcal{Z}$, since honest users do not make use of this information, and corrupted users can be assumed to already know the keys of other corrupt users.

Game 1 In game I_1, if R is corrupt, C changes its behaviour as follows.

5. C
   (c) $(\hat{\kappa}, \alpha) \leftarrow \mathsf{ECom}(\mathsf{PK}, \mathsf{pid}_R, \mathsf{sk}_R)$
   (d) $d \leftarrow \mathsf{Eqv}(\mathsf{PK}, \mathsf{pid}_R, \mathsf{sk}_R, \hat{\kappa}, \alpha, b)$

Lemma 28. *The difference between I'_0 and I_1 is bounded by the advantage against the equivocability of $\mathcal{IBTC}$.*

$$\exists \mathcal{B}_4 \in \mathcal{IPT}\, \left(\Delta(I'_0, I_1) \leq \mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IBTC},\mathcal{B}_4}\right).$$

Proof. We want to show that if some $\mathcal{B}$ can distinguish between I'_0 and I_1, then we can use $\mathcal{B}$ to attack the equivocability of $\mathcal{IBTC}$.

We build an attacker $\mathcal{B}_4$ against the equivocability of $\mathcal{IBTC}$ as follows. $\mathcal{B}_4$ receives MSK as input and runs the parties $\bar{\mathcal{G}}_{acrs}$ and C of I'_0 in his head until step 5b, then he sends (pid_R, b) to the challenger. He receives d and continues at step 5d. At the end of the experiment, $\mathcal{B}$ outputs his guess bit b' to indicate his belief that he was involved in an execution of $I_{b'}$. $\mathcal{B}_4$ forwards this bit as his guess.

Game 2 In game I_2, we employ the full strategy of the committer.

5. C
   (d) $d^{\hat{b}} \leftarrow \mathsf{Eqv}(\mathsf{sk}_R, R, \hat{\kappa}, \alpha, \hat{b})$ for $\hat{b} = 0, 1$
   (e) $r \xleftarrow{\$} \mathcal{R}; \hat{\varphi} = \mathsf{Enc}_\rho(d^0; r)$

Lemma 29. *The difference between I_1 and I_2 is bounded by the advantage against the pseudo-randomness and the key-indistinguishability of $\mathcal{PKE}$.*

$$\exists \mathcal{B}_1, \mathcal{B}_2 \in \mathcal{IPT} \left(\Delta(I_1, I_2) \leq 4 \cdot \left(\mathrm{Adv}^{\mathsf{prc}}_{\mathcal{PKE},\mathcal{B}_1} + \mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE},\mathcal{B}_2} \right)^2 \right)$$

Proof. In order to reduce to the security of $\mathcal{PKE}$, the public key needs to be random. We use Shoup's coin-tossing lemma [3, Lemma 5.3 in the full version] to replace the coin-toss in the first three rounds with a random choice, then we apply the reduction.[9]

We first define a game $\hat{I}_1$, which is the same as I_1, except that C chooses ρ at random.

2. R do nothing
3. C
   (a) $\rho \xleftarrow{\$} \Phi$
   (b) $C \to R : \rho$
4. R do nothing
5. C
   (a)
   (b)
   (c) $d \xleftarrow{\$} \mathcal{D}$
   (d) $\kappa = \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_R, b; d)$
   (e) `if` $b = 0$ `then` $r \xleftarrow{\$} \mathcal{R}; \varphi = \mathsf{Enc}_\rho(d; r)$ `else` $\varphi \xleftarrow{\$} \mathcal{C}$
   (f) $C \to R : \kappa, \varphi$

We define another game $\hat{I}_2$, with the same changes as in I_2. By the coin-tossing lemma,

$$\Delta(I_1, I_2)(k) \leq 4 \left(\Delta(\hat{I}_1, \hat{I}_2)(k) \right)^2.$$

In order to put a bound on $\Delta(\hat{I}_1, \hat{I}_2)(k)$, we define some intermediate games. In game $\hat{I}'_1$

3. C
   (a) `if` $b = 0$ `then` $(\rho, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^k)$ `else` $\rho \xleftarrow{\$} \Phi$

Clearly there exists $\mathcal{B}_2 \in \mathcal{IPT}$ such that $\Delta(\hat{I}_1, \hat{I}'_1)(k) \leq \mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE},\mathcal{B}_2}(k)$. It is also easy to see that there exists $\mathcal{B}_1 \in \mathcal{IPT}$ such that $\Delta(\hat{I}'_1, \hat{I}_2)(k) \leq \mathrm{Adv}^{\mathsf{prc}}_{\mathcal{PKE},\mathcal{B}_1}(k)$, and the lemma follows.

Game 3 In game I_3, we change the commitment used by the receiver. Remember that R is magically provided with sk_C if C is corrupt. In this case, R changes its behaviour as follows.

2. R
   (b) $(\hat{\kappa_1}, \alpha) \leftarrow \mathsf{ECom}(\mathsf{PK}, \mathsf{sk}_C, C)$
   (c) $\hat{d_1} \leftarrow \mathsf{Eqv}(\mathsf{sk}_C, C, \rho_1)$

Lemma 30. *The difference between I_2 and I_3 is bounded by the advantage against the equivocability of $\mathcal{IBTC}$.*

$$\exists \mathcal{B}_5 \in \mathcal{IPT} \left(\Delta(I_2, I_3)(k) \leq \mathrm{Adv}^{\mathsf{equivoc}}_{\mathcal{IBTC},\mathcal{B}_5} \right)$$

[9] The wording of the coin-tossing lemma is complex, but it allows us to do exactly what we do: Replace a randomly chosen string with the result of a coin tossing protocol.

Proof. Let $\mathcal{B}$ be a distinguisher between I_2 and I_3. We build an attacker $\mathcal{B}_5$ against the $\mathsf{equivoc}$ of $\mathcal{IBTC}$ as follows. $\mathcal{B}_5$ gets as input MSK and runs the parties $\bar{\mathcal{G}}_{acrs}$ and R in his head until step 2a, then he sends pid_C, b to the challenger. He receives d and continues at step 2d, setting $\hat{d}_1 = d$. If d was chosen at random, we are in game I_2, if it was generated using Eqv, we are in game I_3. At the end of the experiment, $\mathcal{B}$ outputs his guess bit b', which $\mathcal{B}_5$ forwards.

In game I_3', we change the computation of ρ.

2. R
   (a) $\bar{\rho} \xleftarrow{\$} \Phi$
   (b) $(\hat{\kappa_1}, \alpha) \leftarrow \mathsf{ECom}(\mathsf{PK}, \mathsf{sk}_C, C)$
   (c)
   (d) $R \rightarrow C : \hat{\kappa_1}$

4. R
   (a) $\hat{\rho_1} = \bar{\rho} \cdot \rho_2^{-1}$
   (b) $\hat{d_1} \leftarrow \mathsf{Eqv}(\mathsf{sk}_C, C, \hat{\rho_1})$
   (c) $R \rightarrow C : \hat{d_1}, \hat{\rho_1}$

Lemma 31. *The games I_3 and I_3' are perfectly indistinguishable.*

$$\Delta(I_3, I_3')(k) = 0$$

Proof. Moving the computation of $\hat{d_1}$ to step 4 makes no difference, since $\hat{d_1}$ is only revealed afterwards. The distribution of $\hat{\rho_1}$ is identical to the distribution of ρ_1: both are uniform in Φ.

In I_3'', we substitute key generation for the random choice of $\bar{\rho}$.

2. R
   (a) $(\bar{\rho}, \mathsf{sk}_{\bar{\rho}}) \leftarrow \mathsf{Gen}(1^k)$

Lemma 32. *The difference between I_3' and I_3'' is bounded by the advantage against the key indistinguishability of $\mathcal{PKE}$.*

$$\exists \mathcal{B}_3 \in \mathcal{IPT} \left(\Delta(I_3', I_3'') \leq \mathrm{Adv}^{\mathsf{key\text{-}ind}}_{\mathcal{PKE}, \mathcal{B}_3} \right)$$

Proof. Let $\mathcal{B}$ be a distinguisher between I_3' and I_3''. We build an attacker $\mathcal{B}_3$ against the $\mathsf{key\text{-}ind}$ of $\mathcal{PKE}$ as follows. $\mathcal{B}_3$ takes as input an encryption key ek and simulates the game I_3' to $\mathcal{B}$, setting $\bar{\rho} = \mathsf{ek}$. If ek was drawn at random from Φ, this is game I_3', if it was generated using Gen, this is game I_3''.

Game 4 In game I_4, we delay computation of protocol flows until they are observable by $\mathcal{A}$.

Lemma 33. *The games I_3'' and I_4 are perfectly indistinguishable.*

$$\Delta(I_3'', I_4) = 0$$

Proof.

Game 5 Game I_5 is the simulation in the ideal world.

Lemma 34. *The difference between I_4 and I_5 is bounded by the decryption error probability of $\mathcal{PKE}$ and the advantage against the key extractability of $\mathcal{IBTC}$.*

$$\exists \mathcal{B}_6 \in \mathcal{IPT} \left(\Delta(I_4, I_5) \leq \mathsf{Succ}^{\mathsf{key\text{-}extr}}_{\mathcal{IBTC},\mathcal{B}_6} + \varepsilon^{\mathsf{corr}}_{\mathcal{PKE}} \right).$$

Proof. The simulator can equivocate using $\mathcal{IBTC}$ and the receiver's secret key. To extract the commitment from a corrupted sender at the end of step 5, the simulator decrypts φ using $\mathsf{sk}_{\bar{\rho}}$ to obtain d, then checks whether $\kappa = \mathsf{Com}(\mathsf{PK}, \mathsf{pid}_R, b; d)$. If this is the case, he commits to 0, else to 1.

There are two ways this can fail: Either φ was a valid encryption that the simulator could not decrypt, which contradicts the correctness of $\mathcal{PKE}$ and happens with at most negligible probability. Or the simulator decrypted φ to obtain a valid d, but the commitment is later opened to 1.

We first define an intermediate game I'_4, where decryption errors never happen. We have $\Delta(I_4, I'_4)(k) \leq \varepsilon^{\mathsf{corr}}_{\mathcal{PKE}}(k)$.

The difference between I'_4 and I_5 can only stem from errors of the second kind. Then we can use the key extraction of $\mathcal{IBTC}$ to extract the secret key sk_R of the honest party R.

Assume the simulator decrypts and obtains a valid d_0 opening the commitment to 0, but is later presented with a d_1 opening the commitment to 1. The simulator runs the extractor E on input $(\mathsf{PK}, \mathsf{id}, d_0, 0, d_1, 1)$. If the output is a valid secret key $\mathsf{sk}_{\mathsf{id}}$ for identity id, then $\mathsf{sk}_{\mathsf{id}}$ is a valid trapdoor, so this happens only with negligible probability by the well-formedness of the environment. If the output is not a valid secret key, this yields a successful adversary against the key extractability. □

References

1. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.
2. R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145. IEEE, October 2001. Full version at `http://eprint.iacr.org/2000/067/20011009:204202`.
3. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, 2007. Full version at `http://eprint.iacr.org/2006/432`.
4. R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.
5. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003.
6. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In J. H. Reif, editor, *STOC*, pages 494–503. ACM, 2002.
7. R. Canetti, R. Pass, and A. Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259. IEEE Computer Society, 2007.
8. P. Rogaway. Formalizing human ignorance. In *Vietcrypt 2006*, volume 4341 of *LNCS*, pages 211–228. Springer, 2006.