# Universally Composable Authentication and Key-Exchange with Global PKI

Ran Canetti[1,2⋆], Daniel Shahaf[1⋆⋆], and Margarita Vald[1⋆⋆⋆]

[1]Tel-Aviv University [2]Boston University

**Abstract.** Message authentication and key exchange are two of the most basic tasks of cryptography and are often basic components in complex and security-sensitive protocols. Thus composable security analysis of these primitives is highly motivated. Solutions based are prevalent. Still, the state of the art in composable security analysis of these primitives is somewhat unsatisfactory in the prevalent case where solutions are based on public-key infrastructure (PKI). Specifically, existing treatments either (a) make the unrealistic assumption that the PKI is accessible only within the confines of the protocol itself, thus failing to capture real-world PKI-based authentication, or (b) impose often-unnecessary requirements—such as strong on-line non-transferability—on candidate protocols, thus ruling out natural candidates.

We give a modular and universally composable analytical framework for PKI-based message authentication and key exchange protocols. This framework guarantees security even when the PKI is pre-existing and globally available, without being unnecessarily restrictive. Specifically, we model PKI as a global set-up functionality within the *Global UC* security model [Canetti *et al.*, TCC 2007] and relax the ideal authentication and key exchange functionalities accordingly. We then demonstrate the security of basic signature-based authentication and key exchange protocols. Our modeling makes minimal security assumptions on the PKI in use; in particular, "knowledge of the secret key" is not needed. Furthermore, there is no requirement of uniquenss in this binding: an identity may be represented by multiple strings of public keys.

**Keywords:** public-key infrastructure, message authentication, digital signatures, key exchange, deniability, non-transferability, universal composability

⋆ Email: canetti@bu.edu. Supported by the Check Point Institute for Information Security, an NSF EAGER grant, and NSF Algorithmic Foundations grant no. 1218461.

⋆⋆ Email: daniel.shahaf@cs.tau.ac.il. Supported by the Check Point Institute for Information Security.

⋆⋆⋆ Email: margarita.vald@cs.tau.ac.il. Supported by the Check Point Institute for Information Security.

# 1 Introduction

**Public-key-based authentication.** Authentication may be done in many different ways, such as biometric human identification, or via some pre-shared longer-term secret (such as a pre-shared key or a password). In this work we concentrate on public-key authentication, as put forth in the groundbreaking work of Diffie and Hellman [DH76]: The parties have no *à priori* shared secret information or other physical means for authentication. The only mechanism available for authenticating messages is a globally-accessible public database that allows actors to record arbitrary information; each record is made publicly available and linked to the public identity of the actor who created it. We call this setting the *global public-key infrastructure* (PKI) setting.

A simple and frequently-used message authentication protocol in this setting proceeds as follows. For Alice to send an authenticated message to Bob, Alice signs (using her private key) the message, together with her and Bob's identities and a session identifier that's unique to that message, and sends the message and the signature to Bob over an unauthenticated channel. Bob authenticates the message by obtaining Alice's public key from the PKI and verifying the signature.

An almost equally simple authenticated key exchange protocol is the following: Alice sends to Bob her Diffie-Hellman message $g^a$, bob responds by sending his Diffie-Hellman message $g^b$, together with $g^a$ and a signature $s_B = \mathsf{Sig}_{\text{Bob}}(g^a, g^b, \texttt{`Alice'})$. Alice responds by $s_A = \mathsf{Sig}_{\text{Alice}}(g^a, g^b, \texttt{`Bob'})$. Both parties are assumed to have each other's verification key in advance, and verify the signatures to authenticate. (This is essentially the ISO 9798-3 key exchange standard.) For sake of illustration, we keep these two simple protocols, respectively denoted $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$, as running examples throughout this paper. Practical protocols that use $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$ (or close variants thereof) to establish trust in the identity of an interlocutor or in data payloads are ubiquitous. For instance, they include the TLS standard, chip-and-pin debit cards [EMV11], end-to-end authentication of email contents [RFC 1847], and many others.

Since these protocols use signatures against a globally-available PKI, and send them in the clear over world-readable channels, anyone in the system can verify Alice's and Bob's signatures, even though they were intended only for each other. While we recognize this as an inherent property of signatures (namely, they provide *transferable* verifiability), in the context of authentication this is merely a side-effect which may or may not be desirable.

We know that faithfully analyzing the security of public-key based authentication and key exchange protocols turns out to be a difficult problem, mainly due to the intricate interactions among the various components of the actual protocols, the public-key infrastructure, and the systems they run in. So a natural question arises: Is $\phi_{\text{auth}}$ a good authentication protocol? Is $\phi_{\text{ke}}$ a good key exchange protocol? Should we keep using them? Should we treat them as broken and use more sophisticated protocols instead?

**Modular analysis.** In light of the complexity and ubiquity of authentication protocols, it would be desirable to be able to analyze them in a modular fashion: to abstract out an ideal authentication service for higher-level protocols to use, such that the security of the higher-level protocols would be independent of the details of its implementation. This approach allows consumers of authentication to dynamically replace their authentication

implementations—for example, to base authentication on a different setup service or on a different hard problem—without affecting the security of the higher-level protocol. Conversely, modularity also encourages reuse of an authentication module by multiple higher-level protocols, discouraging local, *ad hoc* implementations.

Several efforts to model public-key based authentication within a composable security framework appear in the literature. Canetti and Krawczyk [CK01] and Shoup [Sho99] perhaps provide the first such guarantees in the context of authenticated key exchange, but their modeling of the public key infrastructure is quite rudimentary and does not allow analyzing the long-term signature and certification module separately from the rest of the protocol.

Other attempts at composable analysis were made in [CK02] and later in [Can04] within the Universally Composable (UC) security framework of [Can01]. (The second work is more directly focused at analyzing the simple $\phi_{\text{auth}}$.)

However, these works have the following significant drawback: They treat the public-key infrastructure—namely, the public record with the public information provided by each actor—as a construct that is local to each specific protocol instance and unavailable for use outside that protocol instance. This modeling is inadequate for representing the PKI model as envisioned by Diffie and Hellman and used in practice—where the public information is *globally* available. Instead, this analysis guarantees security only when each instance of the analyzed protocol uses its own independent instance of a PKI.

This is the case even if the PKI is modeled as joint to a number of instances of the authentication protocol in question, and composition is argued via Universal Composition with Joint State (JUC) [CR03]. Indeed, even there the PKI is modeled not as a global entity but rather as an entity that is local to a specific collection of instances of some specific protocol.

The works of [MTC13,KMO+14], which are set in the Abstract Cryptography setting of [MR11], have a similar modeling shortcoming: the public key infrastructure is modeled as local to the protocol instance. Furthermore, as argued below, this discrepancy is not merely aesthetic; rather, it has real security implications.

Long-lived, global trusted information that is shared among all parties, protocols, and protocol instances in the system are addressed in the Global UC (GUC) framework [CDPW07]. That framework is similar to the ("basic") UC framework, but directly models trusted entities that are globally available throughout the system regardless of any specific protocol to be analyzed. Authentication protocols with global PKI are analyzed in [DKSW09,Wal08]. However, these works consider only authentication protocols that provide additional properties on top of authenticity: only protocols that provide the *non-transferability* (or, *deniability*) property are considered. This leaves us with the following fundamental question:

> How to formulate the basic composable security requirements from plain PKI-based authentication and key exchange protocols? In particular, how to justify signature-based protocols such as $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$?

**A litmus test: the transferability problem.** The discrepancy between the security modeling of [CK02,Can04,CR03,MTC13,KMO+14] and real implementations of PKI

infrastructure is illustrated by the following issue: while real-life PKI-based authentication is transferable (i.e., non-deniable), ideal authentication is not.[1]

In detail, ideal authentication is defined as a deniable task that leaves "no trace"; it passes a message from the sender to the receiver, but the receiver is unable to subsequently prove to a third party that the authentication had in fact happened. In contrast, some PKI-based authentication protocols (and, in particular, protocol $\phi_{\text{auth}}$) allow the receiver to obtain a *transferable* and non-repudiable proof of communication (e.g., a signature), which can be verified by anyone against the global PKI. Hence, PKI-based authentication protocols are transferable (non-deniable) whenever the PKI is globally available. Moreover, this transferability gap is independent of the security model in use. This was formalized by [DKSW09], which proves that no protocol based on a plain PKI can realize the ideal authentication functionality. Still, in [Can04,CR03,MTC13,KMO+14], protocol $\phi_{\text{auth}}$ (or variants thereof) securely realize an ideal process that guarantees non-transferable authentication. (Note that moving to a stronger modeling of PKI, where registering parties are required to prove knowledge of a secret key associated with the registered public value, does not solve the problem. Indeed, protocol $\phi_{\text{auth}}$ remains transferable even with such stronger PKI.)

We stress that transferability, or lack thereof, is not the main concern of this work; it only serves an example of the inadequacy of the current models of composable security in capturing the security requirements of PKI-based authentication and key exchange.

**What about game-based modeling?** The above line of reasoning concentrates on models that provide composable security, more specifically models that define security by way of emulating an ideal process. Can we avoid the difficulties described above by putting general composability aside and instead using game-based modeling of authentication and key exchange? This is an interesting research direction. Indeed, we are not aware of any game-based modeling of authentication and key exchange that directly considers global PKI that can be used (and abused) by arbitrary other applications.

### 1.1 Our results

We provide a framework for analyzing security of authentication and key exchange protocols that use a globally-available PKI. Our framework adequately represents global PKIs. Specifically, we concentrate on authentication and justifying the security of transferable protocols. To exemplify our framework, we analyze protocols $\phi_{\text{ke}}$ and $\phi_{\text{auth}}$, which previously could not be justified in a realistic security model. In particular:

(a) We model global PKI as a globally-available bulletin-board that provides minimal guarantees of binding between strings and identities, without requiring or promising any knowledge or secrecy.
(b) We relax the UC authentication and key exchange functionalities of [CK02,Can04] to be non-deniable. Our functionalities $\mathcal{F}_{\text{cert-auth}}$ and $\mathcal{F}_{\text{cert-ke}}$ allow the adversary to obtain "global" certificates on messages that have the session id of $\mathcal{F}_{\text{cert-auth}}$ or

[1] We use the terms "transferability" and "deniability" interchangeably, where they refer to properties of message authentication.

$\mathcal{F}_{\text{cert-ke}}$ as a prefix. (A global certificate is one that can be verified by any entity in the system.) In particular, the adversary may obtain a global certificate on the message to be authenticated. This coupling eliminates the authentication functionality's deniability, without affecting authenticity.

We remark that the underlying technical trick in $\mathcal{F}_{\text{cert-auth}}$ is reminiscent of the one in the relaxed key exchange functionality of [DKSW09]. However, there, one needs a PKI that is only partially-global and a very specific non-deniable protocol to realize that functionality. In contrast, our goal in this work is to analyze basic protocols with a completely-global PKI.

(c) We prove security of the natural public-key-based protocols $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$. The protocols require no setup beyond a bulletin-board and GUC-securely realize the authentication and key exchange functionalities $\mathcal{F}_{\text{cert-auth}}$ and $\mathcal{F}_{\text{cert-ke}}$, respectively.

To the best of our knowledge, *this is the first treatment of authentication with a realistic modeling of PKI* as a global construct that can be used by arbitrary protocols.

While we concentrate on protocol $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$ for simplicity and clarity, our treatment can be naturally extended to deal with other PKI-based authentication and key exchange protocols.

**Review of UC and GUC.** We first briefly review the UC and GUC frameworks. Informally, UC security is defined via a challenge to distinguish between actual attacks, performed by an adversary $\mathcal{A}$ on protocol $\pi$ and simulated attacks, performed by a simulator $\mathcal{S}$ on protocol $\phi$. The model allows the attacks to be orchestrated by an environment $\mathcal{Z}$ that has an I/O interface to the parties running the challenge protocol ($\pi$ or $\phi$) and is allowed to freely communicate with the attacker (without knowing whether it is $\mathcal{A}$ or $\mathcal{S}$). However, the environment $\mathcal{Z}$ is *constrained* to execute only a single instance of the challenge protocol. In this execution model, protocol $\pi$ is said to *UC-emulate* the protocol $\phi$ if for any adversary $\mathcal{A}$ attacking a protocol $\pi$ there exists a simulator $\mathcal{S}$ attacking protocol $\phi$ such that no environment can successfully distinguish these two possible scenarios.

The GUC challenge experiment is similar to the basic UC experiment, only with an *unconstrained* environment. In particular, now $\mathcal{Z}$ is allowed to invoke and interact with arbitrary protocols, and even multiple sessions of the challenge protocol. The protocols invoked by $\mathcal{Z}$ may share subroutines with challenge protocol instances. GUC emulation is defined analogously to basic UC emulation. The UC and GUC frameworks are presented more rigorously in Section 2.

**Our methods.** We develop a general framework for analyzing PKI-based authentication and key-exchange protocols. Our framework consists of an ideal message authentication functionality (or ideal key-exchange functionality) coupled with a long-lived certificates functionality.

For simplicity we concentrate on the authentication protocol. The treatment of the key exchange protocol is analogous. We formulate an ideal authentication functionality that does not impose unnecessary requirements (such as deniability) on the implementing protocols. The functionality, denoted $\mathcal{F}_{\text{cert-auth}}$, is a sender-receiver functionality that on input $m$ from the sender not only delivers $m$ to the receiver but also allows the adversary to see legitimate signatures on messages of its choice, which $\mathcal{F}_{\text{cert-auth}}$ obtains from

the ideal certificates functionality $\mathcal{G}_{\text{cert}}$. (This does not affect $\mathcal{F}_{\text{cert-auth}}$'s authenticity promises since $\mathcal{F}_{\text{cert-auth}}$ delivers the original $m$ to the receiver.) This is done as follows:

The adversary determines the message to be signed and hands it to $\mathcal{F}_{\text{cert-auth}}$; then, $\mathcal{F}_{\text{cert-auth}}$ requests a signature (on behalf of the sender) on the message affixed with the session identifier. The signature obtained by the adversary is thus tied to a specific $\mathcal{F}_{\text{cert-auth}}$ session and cannot be used in other sessions. Since the signature seen by the adversary is correctly generated and can be successfully verified by any entity in the system, deniability (or, non-transferability) is no longer guaranteed. Nonetheless, the essence of authentication—binding an action to some long-lived entity—remains guaranteed. That is, $\mathcal{F}_{\text{cert-auth}}$ guarantees that if a recevier accepts a message from a given sender, then that sender sent that message to the receiver. Therefore, any protocol that GUC-reaizes $\mathcal{F}_{\text{cert-auth}}$ guarantees authenticated message transmission in the same way.

Observe that $\mathcal{F}_{\text{cert-auth}}$ allows the adversary to obtain, as a side-effect, the sender's signature on almost any message. This might seem weak, and almost contradictory to authentication. We note however that (a) $\mathcal{F}_{\text{cert-auth}}$ still guarantees authenticity, as argued above, and (b) other standard definitions of securiy for authentication protocols (e.g., the definition of authentication based on a local PKI) also allow the same side effects. We simply make this point explicit.

We note that a somewhat similar mechanism is used by [DKSW09] to augment the key exchange functionality with the secret keys of the parties. However, there the secret keys are made unavailable beyond the key exchange protocol, which is the opposite of our purpose here. Indeed, the goal in [DKSW09] is close to diametrically opposite to the goal of this work: Dodis *et al.* study deniable protocols, whereas we study real-life, non-deniable protocols.

We also show that standard EU-CMA signatures together with a globally-available PKI precisely capture the guarantees provided by $\mathcal{G}_{\text{cert}}$, and can be used in its stead. That is:

(a) We define a global ideal certificate functionality $\mathcal{G}_{\text{cert}}$ that is parametrized by a party identity (PID). That is, $\mathcal{G}_{\text{cert}}$ is willing to provide certificates on chosen messages to any session of that PID. The verification service is provided to any PID in the system. The authentication functionality $\mathcal{F}_{\text{cert-auth}}$ will provide certificates generated by $\mathcal{G}_{\text{cert}}$ to the adversary.
(b) To realize $\mathcal{G}_{\text{cert}}$, we define a signing module $\mathcal{G}_{\Sigma}$, parametrized by a PID, that holds the secret key (of some signature scheme) and similarly to $\mathcal{G}_{\text{cert}}$ is willing to provide signing service to any session of that PID. Similarly to [CK01], our signing module enables modeling "key knowledge" and "signing capabilities" separately. Separation of long-term key handling and signing module from session module is an essential part of security modeling of key-exchange and secure sessions: it preserves security of sessions even when other sessions using the same public-key are compromised. This was not done previously in any UC-based framework.
(c) We show a GUC-secure realization of ideal certificates $\mathcal{G}_{\text{cert}}$ from standard EU-CMA signatures (where the secret key is kept in the signing module).

We exemplify the usability of our model by analyzing $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$, the signed key exchange protocol of Diffie-Hellman ISO 9798-3, within it and showing it GUC-realizes

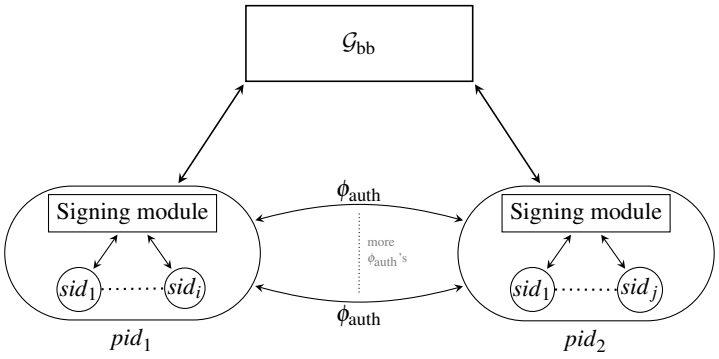



**Fig. 1:** A snapshot of an authentication in the system. The signing module together with $\mathcal{G}_{\text{bb}}$ is an instantiation of $\mathcal{G}_{\text{cert}}$. Each party participates in multiple executions of $\phi_{\text{auth}}$, one per session. Each session may involve a different interlocutor (not limited to $pid_1$ and $pid_2$). The bulletin-board $\mathcal{G}_{\text{bb}}$ is shared with many other protocols executing in the system. The parties also obtain signatures from their local signing module instances upon demand.

$\mathcal{F}_{\text{cert-auth}}$ and $\mathcal{F}_{\text{cert-ke}}$, respectively. (The complete realization of $\mathcal{F}_{\text{cert-auth}}$ within our framework is depicted in Figure 1).

To this end, we formalize new composition theorems that allow reduction between global functionalities. The first theorem (in Section 3) shows that a secure realization of functionality $\mathcal{G}$ is sufficient for replacing any use of $\mathcal{G}$ (as a global functionality) with $\mathcal{G}$'s implementation:

**Theorem 1 (informal statement).** *Let $\pi$ be a protocol with access to global functionality $\mathcal{G}$. If a functionality $\mathcal{F}$ GUC-realizes $\mathcal{G}$, then $\pi$ using global $\mathcal{F}$ GUC-realizes $\pi$ using global $\mathcal{G}$.*

Our second composition theorem presents the necessary conditions, required from a pair of global functionalities, such that any secure protocol GUC-realizing some task using globally one of the functionalities would remain equally secure using the other:

**Theorem 2 (informal statement).** *Let $\pi$ and $\phi$ be protocols with access to global functionality $\mathcal{G}$. If $\pi$ GUC-realizes $\phi$, the functionality $\mathcal{F}$ GUC-realizes $\mathcal{G}$ and $\mathcal{G}$ GUC-realizes $\mathcal{F}$, then $\pi$ GUC-realizes $\phi$ with access to global functionality $\mathcal{F}$.*

Since the operation of replacing one global functionality by another was not considered before, we extend the definition of GUC-emulation. The extended definition admits not only previous results, but also allows arguing these theorems formally. Although the composition proof is simple, the terminology is vital for our analysis.

### 1.2 Related work

Due to the fundamentality of the problem, there has been a vast line of works on secure authentication and its equivalent problem of key exchange. PKI-based authentication can be examined from three different angles: the composability guarantees of the model, the

modeling of the PKI, and the deniability guarantees of the ideal authentication. We concentrate on composable settings, where the authentication (or key exchange) maintains its security guarantees when used as a component in building complex protocols.

**UC-based models.** Many works [CK02,FAK08,CG10,AF10] analyze key agreement and key exchange protocols in the UC framework. However, like [Can04], they also model the PKI as local to the protocol instance. Another line of works in UC prohibit honest participants from engaging in multiple sessions concurrently [LBdM07,BLdMT09] or assume password-based security and erasures [DF12]. Likewise, here the PKI modeling does not allow external protocols to access the PKI.

Dodis *et al.* [DKSW09,Wal08] study deniable authentication in a GUC setting. They prove it impossible to securely realize standard message authentication in GUC with merely a standard PKI. To overcome this impossibility result, they present a non-transferable authentication protocol based on symmetric keys. The symmetric keys are obtained from a non-standard PKI. However, their protocol has two drawbacks: Its security proof requires a strong PKI (namely, key registration with proof of knowledge of the secret key) and their protocol is somewhat less efficient than $\phi_{\text{auth}}$. Most importantly, that framework cannot be used to justify the security of $\phi_{\text{auth}}$ as a basic authentication protocol.

**The Abstract Cryptography (AC) model.** Maurer *et al.* [MTC13] implement authenticated channels in the Abstract Cryptography setting of [MR11]. Their construction is composable, uses the canonical signature-based authentication protocol ($\phi_{\text{auth}}$) and assumes a standard PKI. Still, similarly to Canetti [Can04], these works treat the PKI as a *local* functionality that services only a single instance of an authentication protocol. Indeed, their abstraction of an authentication channel is deniable, while their protocol is PKI-based.

Kohlweiss *et al.* [KMO+14] study the TLS protocol in the same setting and analyze three key exchange modes of TLS. Of them, one uses symmetric keys and two use a standard PKI. However, as with [Can04] and [MTC13], their PKI is private to the protocol. Thus, their modeling does not adequately capture global PKIs.

**Game-based models.** The work of [CK01] develops a game-based framework for analyzing the key exchange problem. Later, [BFS+13] proposed a framework with stronger composability guarantees to enable analysis of the TLS protocol. However, both frameworks allow only limited composition and model the PKI as a setup inaccessible by other protocols.

**Other models.** Kidron and Lindell [KL07] study impossibility results in a number of public-key models. However, none of the considered public-key models are in a global setting, and thus do not address the issue at hand. Barak *et al.* [BCL+05] study what notion of security is achievable in a PKI-less setting. Their work does not address the setting of global PKI.

**Invisible adaptive attacks.** Nielsen and Strefler [NS14] point out a weakness in definitions of security in the GUC model, called *invisible adaptive attacks* and propose a general way to fix the weakness. We demonstrate in Section 6that our protocols satisfy not only [NS14] definition even a stronger (and simpler) definition proposed in this work.

## 2 Overview of Generalized UC Security

To provide the proper setting for the authentication, we now review the original UC [Can01,Can00] (referred to as basic UC) and Generalized UC [CDPW07] frameworks.[2] We will focus on the notion of protocol *emulation*, wherein the objective of a protocol $\pi$ is to imitate another protocol $\phi$. In this work, the entities and protocols we consider are polynomial-time bounded Interactive Turing Machines (ITMs), in the sense detailed in [Can01].

**Systems of ITMs.** To capture the mechanics of computation and communication among entities, the UC framework employs an extension of the ITM model [GMR89]. A computer program (such as run by a participant in a protocol, or by an adversary) is modeled in the form of an ITM. An execution experiment consists of a system of ITMs which are instantiated and executed, with multiple instances possibly sharing the same ITM code. A particular executing ITM instance running in the network is referred to as an ITI. Individual ITIs are parameterized by the program code of the ITM they instantiate, a party ID (pid) and a session ID (sid). We require that each ITI can be uniquely identified by the identity pair id = (pid,sid), irrespective of the code it may be running. All ITIs running with the same code and session ID are said to be a part of the same protocol session, and the party IDs are used to distinguish among the various ITIs participating in a particular protocol session.

**The Basic UC Framework.** At a very high level, the intuition behind security in the basic UC framework is that any adversary $\mathcal{A}$ attacking a protocol $\pi$ should learn no more information than could have been obtained via the use of a simulator $\mathcal{S}$ attacking protocol $\phi$. Furthermore, we would like this guarantee to hold even if $\phi$ were to be used as a subroutine in arbitrary other protocols that may be running concurrently in the networked environment and after we substitute $\pi$ for $\phi$ in all the instances where it is invoked. This requirement is captured by a challenge to distinguish between actual attacks on protocol $\phi$ and simulated attacks on protocol $\pi$. In the model, attacks are executed by an environment $\mathcal{Z}$ that also controls the inputs and outputs to the parties running the challenge protocol. The environment $\mathcal{Z}$ is *constrained* to execute only a single instance of the challenge protocol. In addition, the environment $\mathcal{Z}$ is allowed to interact freely with the attacker (without knowing whether it is $\mathcal{A}$ or $\mathcal{S}$). At the end of the experiment, the environment $\mathcal{Z}$ is tasked with distinguishing between adversarial attacks perpetrated by $\mathcal{A}$ on the challenge protocol $\pi$, and attack simulations conducted by $\mathcal{S}$ with protocol $\phi$ acting as the challenge protocol instead. If no environment can successfully distinguish these two possible scenarios, then protocol $\pi$ is said to *UC-emulate* the protocol $\phi$.

**Balanced environments.** In order to keep the notion of protocol emulation from being unnecessarily restrictive, we consider only environments where the amount of resources given to the adversary (namely, the length of the adversary's input) is at least some fixed polynomial fraction of the amount of resources given to all protocols in the system. From now on, we only consider environments that are balanced.

[2] We relate to the 2013 version of [Can00] and explicitly mention in the text the relevant differences from previous versions.

**Definition 1 (UC-emulation).** *Let $\pi$ and $\phi$ be multi-party protocols. We say that $\pi$ UC-emulates $\phi$ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any (constrained) environment $\mathcal{Z}$, we have:*

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

Defining protocol execution this way is sufficient to capture the entire range of network activity that is observable by the challenge protocol but may be under adversarial control. Therefore, the UC framework admits a very strong composition theorem, which guarantees that arbitrary instances of $\phi$ that may be running in the network can be safely substituted with any protocol $\pi$ that UC-emulates it. More formally,

**Definition 2 (Subroutine-respecting protocols; [Can00]).** *We say that a protocol $\pi$ is* subroutine-respecting *if the following properties hold with respect to every instance of $\pi$ in any execution of any protocol $\rho$ that makes subroutine calls to $\pi$:*

*(a) No ITI which is a subsidiary of this instance passes inputs or outputs to an ITI which is not a party or subsidiary of this instance.*
*(b) At first activation, each ITI that is currently a subsidiary of this instance, or will ever become one, sends a special message to the adversary, notifying it of its own code and identity, as well as the code $\pi$ and SID of this instance. We call this requirement* subroutine publicness.[3]

**Theorem 3 (UC-Composition).** *Let $\rho$,$\pi$ and $\phi$ be protocols such that $\rho$ makes subroutine calls to $\phi$. If $\pi$ UC-emulates $\phi$ and both $\pi$ and $\phi$ are subroutine-respecting, then protocol $\rho^{\pi/\phi}$ UC-emulates protocol $\rho$.*

**The Generalized UC Framework.** As mentioned above, the environment $\mathcal{Z}$ in the basic UC experiment is unable to invoke protocols that share state in any way with the challenge protocol. In many scenarios, the challenge protocol produces information that is shared by other network protocol sessions. For example, protocols may share information via a global setup such as a public Common Reference String (CRS) or a standard Public Key Infrastructure (PKI). The basic UC framework discussed above does not address this kind of shared state; moreover, the UC composition theorem does not hold for non-subroutine-respecting protocols (i.e., protocols that share state information with other protocol sessions). Still, we would like to analyze such protocols in a modular way. To overcome this limitation, [CDPW07] propose the Generalized UC (GUC) framework. The GUC challenge experiment is similar to the basic UC experiment, only with an *unconstrained* environment. In particular, now $\mathcal{Z}$ is allowed to invoke and interact with arbitrary protocols, and even multiple sessions of the challenge protocol. Some of the protocol sessions invoked by $\mathcal{Z}$ may even share state information with challenge protocol sessions, and indeed, those protocol sessions might provide $\mathcal{Z}$ with information related to the challenge protocol instances that it would have been unable to obtain otherwise. To distinguish this from the basic UC experiment, we denote the output of an unconstrained environment $\mathcal{Z}$, running with an adversary $\mathcal{A}$ and a challenge

[3] While natural, these properties are necessary for Theorem 4 and the composition to go through. The reader is referred to [Can00] for further details.

protocol $\pi$ in the GUC protocol execution experiment, by $\text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. GUC emulation is defined analogously to the definition of basic UC emulation outlined above:

**Definition 3 (GUC-emulation).** *Let $\pi$ and $\phi$ be multi-party protocols. We say that $\pi$ GUC-emulates $\phi$ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any (unconstrained) environment $\mathcal{Z}$, we have:*

$$\text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{GEXEC}_{\phi,\mathcal{S},\mathcal{Z}}.$$

**The External-subroutine UC Framework.** The great generality provided by the GUC framework also raises difficulties in proving security of protocols in it. Observing real scenarios, it turns out to be sufficient to model shared state information via the use of "shared functionalities", which are simply functionalities that may interact with more than one protocol session (such as the PKI functionality). For clarity, we distinguish the notation for shared functionalities by adding a bar. We call a protocol $\pi$ that only shares state information via a single global functionality $\bar{\mathcal{G}}$ a *$\bar{\mathcal{G}}$-subroutine respecting* protocol (Definition 2 is extended to allow communication with $\bar{\mathcal{G}}$). Moreover, a $\bar{\mathcal{G}}$-externally constrained environment is subject to the same constraints as the environment in the basic UC framework, only it is additionally allowed to invoke a single ITI that runs the code of $\bar{\mathcal{G}}$. Thus, any state information that will be shared by the challenge protocol must be shared via calls to $\bar{\mathcal{G}}$ (i.e., challenge protocols are $\bar{\mathcal{G}}$-subroutine respecting), and the environment is specifically allowed to access $\bar{\mathcal{G}}$. Although $\mathcal{Z}$ is once again constrained to invoking a single instance of the challenge protocol, it is now possible for $\mathcal{Z}$ to internally mimic the behavior of multiple sessions of the challenge protocol, or other arbitrary network protocols, by making use of calls to $\bar{\mathcal{G}}$ wherever shared state information is required. We allow the environment direct access to shared state information. This security notion is called External-subroutine UC (EUC) security. The EUC-security notion collapses to UC-security for subroutine-respecting protocols (Definition 2).

Given a $\bar{\mathcal{G}}$-subroutine respecting protocol $\pi$, we denote the output of the environment in the EUC protocol experiment by $\text{EXEC}_{\pi,\bar{\mathcal{G}},\mathcal{D},\mathcal{Z}}$. The EUC-emulation definition presented here is an extension of the emulation definition appearing in [CDPW07]. The new definition allows a protocol $\pi$ to emulate $\phi$ using a different shared functionality than $\phi$ uses. More formally,

**Definition 4 (EUC-emulation).** *Let $\pi$ and $\phi$ be multi-party protocols, where $\pi$ is $\bar{\mathcal{F}}$-subroutine respecting and $\phi$ is $\bar{\mathcal{G}}$-subroutine respecting. We say that $\pi$ EUC-emulates $\phi$ if for any adversary $\mathcal{A}$ there exists a adversary $\mathcal{S}$ such that for any $\bar{\mathcal{F}}$-externally constrained environment $\mathcal{Z}$, we have:*

$$\text{EXEC}_{\pi,\bar{\mathcal{F}},\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\phi,\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}.$$

Note that a $\bar{\mathcal{F}}$-subroutine respecting $\pi$ communicates with the global functionality $\bar{\mathcal{F}}$ (similarly, $\phi$ with $\bar{\mathcal{G}}$). We remark that, in the underlying model, the substitution of $\bar{\mathcal{G}}$ for $\bar{\mathcal{F}}$ is done by changing the control function (so that messages addressed to $\bar{\mathcal{F}}$ are implicitly delivered to $\bar{\mathcal{G}}$ instead), in a similar manner to the changes effected thereto when substituting $\phi$ for $\pi$ in UC or GUC.

**Ideal protocols ([Can01,Can00]).** Let $\mathcal{F}$ be an ideal functionality and *sid* be its session ID. The ideal protocol $\mathsf{IDEAL}_{\mathcal{F}}$ for $\mathcal{F}$ is defined as follows: Whenever a dummy party is activated with input $v$, it writes $v$ onto the input tape of the ideal functionality $\mathcal{F}_{(sid,\perp)}$ (recall that this message includes the extended identity of the calling ITI). Messages delivered by the adversaries, including corruption messages, are ignored. Whenever a dummy party receives a value $v$ from $\mathcal{F}$ on its subroutine output tape, it writes this value on the subroutine output tape of an ITI instructed by $\mathcal{F}$. Specifying the output destination enables an ideal functionality $\mathcal{F}$ to communicate with another (shared) ideal functionality $\bar{\mathcal{Q}}$ via the dummy party. Such functionality $\mathcal{F}$ is called $\bar{\mathcal{Q}}$-subroutine respecting functionality. We say that a functionality $\mathcal{F}$ EUC-realizes an functionality $\mathcal{G}$ if $\mathsf{IDEAL}_{\mathcal{F}}$ EUC-emulates $\mathsf{IDEAL}_{\mathcal{G}}$. GUC-realization is defined analogously.

Since the class of $\bar{\mathcal{G}}$-subroutine respecting protocols captures a broad range of real-life protocols, we focus our attention on those. For this class of protocols, [CDPW07] shows that GUC-emulation is equivalent to EUC-emulation.

**Theorem 4 ([CDPW07]).** *Let $\mathcal{G}$ be some ideal functionality and let $\pi$ and $\phi$ be $\bar{\mathcal{G}}$-subroutine respecting protocols. Then $\pi$ GUC-emulates $\phi$, if and only if $\pi$ EUC-emulates $\phi$.*

Although it is not stated in [CDPW07], subroutine publicness of $\phi$, as described in Definition 2, is necessary for the equivalence to hold.

As a special case, if the challenge protocol does not share any state information (i.e., it is subroutine-respecting according to [Can01]), then Theorem 4 states that GUC- and UC-security are equivalent.

## 3 The Global Functionality Composition Theorem

Suppose a protocol $\rho$ uses another protocol $\phi$ as a subroutine. Global UC [CDPW07] shows that we can replace the use of $\phi$ with any protocol $\pi$ that GUC-emulates it. This replacement maintains the security of the composed protocol, even if both the calling protocol $\rho$ and the subroutine protocol ($\phi$ or $\pi$) have access to the same instance of a global ideal functionality. However, it is unknown whether it is safe to replace the global functionality with something "equivalent". Such a replacement would be useful, for example, for designing protocols using an efficient signatures scheme (with keys that can be used concurrently by any other protocols) and analyzing their security using an ideal signatures functionality.

In this section we provide a new composition theorem that handles security of global functionality replacement. Informally, the theorem states that a protocol that shares state via a global functionality $\bar{\mathcal{G}}$ remains secure if we replace this functionality with a different (presumably weaker) global functionality $\bar{\mathcal{F}}$, provided that $\mathcal{F}$ is a secure implementation of $\mathcal{G}$. The theorem holds even if the global functionalities share state via a third global functionality. (In Section 4, this theorem is used to substitute an ideal certification functionality, which shares state via a global PKI functionality, by EU-CMA signatures.)

**Theorem 5 (Generalized Functionality Composition).** *Let $\mathcal{G}$,$\mathcal{F}$ be $\bar{\mathcal{Q}}$-subroutine respecting functionalities, for some ideal functionality $\mathcal{Q}$. Let $\pi$ be a $\bar{\mathcal{G}}$-subroutine respecting protocol. If $\mathcal{F}$ EUC-realizes $\mathcal{G}$, then $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ GUC-emulates $\pi$.*

*Proof.* We denote by $\pi$ and $\pi'$ the protocols $\pi^{\bar{\mathcal{G}}}$ and $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ respectively. We first prove that $\pi'$ EUC-emulates $\pi$ and then show that GUC-emulation follows. We make use of an equivalent formulation of emulation with respect to dummy adversaries. Thus, denoting the dummy adversary by $\mathcal{D}$, we wish to construct an adversary $\mathcal{S}$ such that:

$$\text{EXEC}_{\pi',\bar{\mathcal{F}},\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\pi,\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}}$$

for any $(\bar{\mathcal{F}},\bar{\mathcal{Q}})$-constrained environment $\mathcal{Z}$. Since $\mathcal{F}$ EUC-realizes $\mathcal{G}$ there is an adversary $\mathcal{S}_{\mathcal{F}}$ such that

$$\text{EXEC}_{\mathcal{F},\bar{\mathcal{Q}},\mathcal{D},\mathcal{Z}_{\mathcal{F}}} \approx \text{EXEC}_{\mathcal{G},\bar{\mathcal{Q}},\mathcal{S}_{\mathcal{F}},\mathcal{Z}_{\mathcal{F}}} \tag{1}$$

for any $\bar{\mathcal{Q}}$-constrained environment $\mathcal{Z}_{\mathcal{F}}$. That is, $\mathcal{S}_{\mathcal{F}}$ expects to interact with $\mathcal{G}$ and $\bar{\mathcal{Q}}$, and translates it to mimic the action of the corresponding execution of $\mathcal{F}$ and $\bar{\mathcal{Q}}$ from the viewpoint of any environment $\mathcal{Z}_{\mathcal{F}}$. We present and analyze $\mathcal{S}$. (We note that the construction of $\mathcal{S}$ and the proof of its validity are reminiscent of the treatment in [CDPW07]. Still, the context is quite different.) The construction idea is to internally run a single copy $\mathcal{S}_{\mathcal{F}}$ to mimic all the calls to $\mathcal{F}$ and route all relevant messages through this adversary. In addition, the adversary $\mathcal{S}$ behaves as follows:

(a) forwarding all messages intended for $\bar{\mathcal{F}}$ sent by the environment $\mathcal{Z}$ to its internal simulation of $\mathcal{S}_{\mathcal{F}}$, as well as forwarding any messages from $\mathcal{S}_{\mathcal{F}}$ back to $\mathcal{Z}$ as appropriate.
(b) forwarding all other messages sent by the environment $\mathcal{Z}$ to the external participants of $\pi$ or to $\bar{\mathcal{Q}}$, as well as forwarding any incoming messages from $\pi$ and $\bar{\mathcal{Q}}$ (and other protocols in the system) back to $\mathcal{Z}$ as appropriate.
(c) forwarding all messages of $\mathcal{S}_{\mathcal{F}}$ to the functionality $\bar{\mathcal{G}}$ and back, as appropriate. This is done using the subroutine publicness property, as explained in Definition 2).

A graphical description of $\mathcal{S}$ can be found in Figure 2(a).

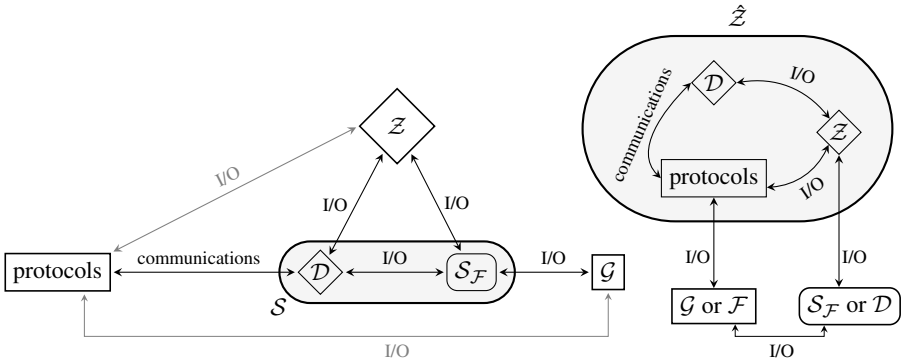
In order to prove that $\mathcal{S}$ satisfies the required, we perform a standard proof by contradiction. Assume there exists an environment $\mathcal{Z}$ capable of distinguishing the interaction with $\mathcal{S}$ and $\pi$ from the interaction with $\mathcal{D}$ and $\pi'$. We show how to construct an environment $\hat{\mathcal{Z}}$ such that

$$\text{EXEC}_{\pi,\bar{\mathcal{G}},\mathcal{S},\mathcal{Z}} = \text{EXEC}_{\mathcal{G},\bar{\mathcal{Q}},\mathcal{S}_{\mathcal{F}},\hat{\mathcal{Z}}}$$

and

$$\text{EXEC}_{\pi',\bar{\mathcal{F}},\mathcal{D},\mathcal{Z}} = \text{EXEC}_{\mathcal{F},\bar{\mathcal{Q}},\mathcal{D},\hat{\mathcal{Z}}}.$$

The environment $\hat{\mathcal{Z}}$ will internally run $\mathcal{Z}$ and behave as follows: Any message from $\mathcal{Z}$ to $\mathcal{F}$ is forwarded to the external adversary. Any output from the external adversary is forwarded back to $\mathcal{Z}$. Any other message from $\mathcal{Z}$ is internally simulated. That is, $\hat{\mathcal{Z}}$ internally executes the dummy adversary $\mathcal{D}$ and honestly simulates any uncorrupted




**(a)** The simulator $\mathcal{S}$ interacting with the protocol $\pi$ and the global functionality $\mathcal{G}$.

**(b)** The environment $\hat{\mathcal{Z}}$ constructed out of $\mathcal{Z}$. The entire system is executed inside $\hat{\mathcal{Z}}$ except for the interaction with the global functionality, which is either $\mathcal{F}$ or $\mathcal{G}$.

**Fig. 2:** The simulator $\mathcal{S}$ and the distinguishing environment $\hat{\mathcal{Z}}$ constructed in the proof.

entity in the execution (i.e., parties of $\pi$ and parties of other protocols). Whenever an internally simulated honest party provides an input to $\mathcal{F}$ or $\bar{\mathcal{Q}}$, the environment $\hat{\mathcal{Z}}$ forwards it externally and the response is forwarded back to the internal honest party. Eventually, the environment $\hat{\mathcal{Z}}$ outputs whatever $\mathcal{Z}$ outputs. The environment $\hat{\mathcal{Z}}$ is depicted in Figure 2(b).

It follows from the construction that if the external adversary is $\mathcal{D}$ then $\mathcal{Z}$ interacts with the dummy adversary $\mathcal{D}$, the protocol $\pi'$ and functionality $\mathcal{F}$. If the external adversary is $\mathcal{S}_\mathcal{F}$ then $\mathcal{Z}$ interacts with $\mathcal{D}$ where all of its accesses to $\mathcal{F}$ are replaced with accesses to $\mathcal{G}$ via $\mathcal{S}_\mathcal{F}$. This is exactly the execution of $\mathcal{Z}$ with the adversary $\mathcal{S}$ and the protocol $\pi$ with access to $\mathcal{G}$. Hence, existence of such distinguishing environment $\mathcal{Z}$ contradicts Equation (1) as desired.

Note that the components of $\mathcal{S}$ (i.e., the dummy adversary $\mathcal{D}$ and simulator $\mathcal{S}_\mathcal{F}$) can handle multiple instances of $\pi$ and therefore $\mathcal{S}$ can simulate $\pi'$ with unconstrained environment as well. In other words,

$$\text{GEXEC}_{\pi',\mathcal{D},\mathcal{Z}} \approx \text{GEXEC}_{\pi,\mathcal{S},\mathcal{Z}}.$$

for any unconstrained environment $\mathcal{Z}$.

Informally, secure realization allows replacing any use of an idealized task by an implementation of the task, in a localized manner (that is, without having to consider the rest of the system). In particular, if a protocol $\pi$ securely implements another protocol $\phi$, where $\bar{\mathcal{G}}$ exists in the system, then we intuitively expect $\pi$ to continue to securely implement $\phi$ after we replace $\bar{\mathcal{G}}$ with some $\bar{\mathcal{F}}$ that securely implements $\bar{\mathcal{G}}$. However, this intuition is misleading. Consider, for example, some functionality $\mathcal{F}$ and let $\mathcal{G}$ be as $\mathcal{F}$ but with extra capabilities granted to the adversary. The functionality $\mathcal{F}$ (trivially)

securely implements $\mathcal{G}$, since it is a restriction of $\mathcal{G}$. However, the simulation of $\pi$ might be such that it uses the extra adversarial capabilities given him by $\bar{\mathcal{G}}$. Thus, once we replace $\bar{\mathcal{G}}$ with $\bar{\mathcal{F}}$ the simulation becomes invalid, and moreover, the extra capabilities might be essential to the simulation ability. This hints that in order for the intuition to hold, it must be the case that $\bar{\mathcal{F}}$ and $\bar{\mathcal{G}}$ must have "similar" adversarial interfaces. This is formally captured as follows:

**Theorem 6.** *Let $\mathcal{G}$, $\mathcal{F}$ be $\bar{\mathcal{Q}}$-subroutine respecting functionalities, for some ideal functionality $\mathcal{Q}$. Let $\pi$, $\phi$ be $\bar{\mathcal{G}}$-subroutine respecting protocols. If the following holds:*

*(a) $\pi$ GUC-emulates $\phi$.*
*(b) $\mathcal{F}$ EUC-realizes $\mathcal{G}$ and vice versa.*

*Then $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ GUC-emulates $\phi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$.*

*Proof.* The theorem fully follows from Theorem 5. We denote by $\pi$ and $\phi$ the protocols $\pi^{\bar{\mathcal{G}}}$ and $\phi^{\bar{\mathcal{G}}}$ respectively. More formally, by Theorem 5 and Item (2) we obtain that $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ GUC-emulates $\pi$. Combining this with Item (1) we obtain that $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ GUC-emulates $\phi$. Next, using again Theorem 5 with Item (2) we infer that $\phi$ GUC-emulates $\phi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ and conclude that $\pi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ GUC-emulates $\phi^{\bar{\mathcal{F}}/\bar{\mathcal{G}}}$ as desired.

Such composition enables the GUC-framework to offer full modularity in analyzing protocols.

## 4 Secure Authentication using Signatures

As discussed in the introduction, the standard authentication functionality $\mathcal{F}_{\text{cert-auth}}$ is unimplementable in a GUC setting with fully global PKI since it requires non-transferability (deniability). However, this *de jure* impossibility does not prevent people from using digital signatures in day-to-day communications to achieve an authentication guarantee.

In this section, we bridge the gap between practical and provably secure authentication. We show that the classic, signature-based authentication protocol implements (transferable) authentication using standard public key infrastructure (PKI). That is, we formalize the "Authentication via signatures" paradigm in a GUC setting and present a functionality which encapsulates it.

This has two benefits: it allows for analyzing in the modular setting of GUC real-life protocols that use digital signatures as a building block, and it increases the trust in the signature-based authentication protocol by proving it secure under GUC's strong composition operation.

The proof details are similar to [Can04]; however, the formulation and analysis are done in the GUC framework. Section 4.1 presents a formulation of ideal certificate and ideal signature functionalities ($\bar{\mathcal{G}}_{\text{cert}}$ and $\bar{\mathcal{G}}_{\text{cwk}}$), and shows their equivalence. Section 4.2 shows that EU-CMA signatures provide the same security guarantees as the ideal signature functionality $\bar{\mathcal{G}}_{\text{cwk}}$. Section 4.3 presents and implements the relaxed, non-deniable message authentication functionality $\mathcal{F}_{\text{cert-auth}}$.

### 4.1 Signatures and certificates

We formulate a global ideal functionality, $\bar{\mathcal{G}}_{\text{cert}}$, that provides ideal binding of messages to party identities. The key difference in our setting is that $\bar{\mathcal{G}}_{\text{cert}}$ is accessible at any time, by any party, no matter which protocols it participates in. Another important difference from previous formulations is that the public key lives in a global bulletin-board, to capture the fact that a principal has a single keypair ("secret") which she uses in multiple protocols. Then, we formulate a global signature functionality $\bar{\mathcal{G}}_{\text{cwk}}$ that realizes $\bar{\mathcal{G}}_{\text{cert}}$ given a public bulletin-board $\bar{\mathcal{G}}_{\text{bb}}$.

**Global Functionality $\bar{\mathcal{G}}_{\text{bb}}$**

**Report:** Upon receiving a message $(\mathsf{Register}, v)$ from party $P$, send $(\mathsf{Registered}, P, v)$ to the adversary; upon receiving OK from the adversary, and if this is the first request from $P$, then record the pair $(P, v)$. Otherwise, ignore the new message.

**Retrieve:** Upon receiving a message $(\mathsf{Retrieve}, P_i)$ from some party $P_j$ (or the adversary $\mathcal{S}$), generate a public delayed output $(\mathsf{Retrieve}, P_i, v)$ to $P_j$, where $v = \perp$ if no record $(P_i, v)$ exists.

**Fig. 3:** The bulletin-board certificate authority (CA) functionality. Any ITI can register a single key that would be associated with its identity. Any ITI in the system can request the key of any other ITI.

**The bulletin board functionality.** The global bulletin board functionality, $\bar{\mathcal{G}}_{\text{bb}}$, is presented in Figure 3. The bulletin board accepts only the first registered value, and does not allow to modify or delete it.[4] The bulletin board is authenticated in a sense that it records the value along with the identity of the publisher, but does not perform any checks on the registered value; it simply publicly records the value. Nonetheless, as we will show later, the present minimal formulation suffices for authentication.

**The certification functionality.** The ideal certification functionality, $\bar{\mathcal{G}}_{\text{cert}}$, is presented in Figure 4. The session ID names a distinguished principal, the 'signer'. The functionality provides direct binding between a message and the identity of the signer. (In contrast, $\mathcal{F}_{\text{sig}}$ , which appears in Figure 5, binds a message only to a verification key.) Using common terminology, this corresponds to providing signatures accompanied by "certificates" that bind the verification process to the signer's identity. The functionality generates a key for each new signer; however, the key is used only to register in the bulletin-board. That is, neither signing nor verification is done with respect to this key. Verification (and signing) requests are processed only if the signer is registered in the bulletin-board, however, they are indifferent to the registered value. Lastly, corrupted signers are allowed to dictate the verification result. We note that $\bar{\mathcal{G}}_{\text{cert}}$ is a $\bar{\mathcal{G}}_{\text{bb}}$-subroutine respecting functionality as defined in Section 2.

[4] The modeling of PKI that allows a single public key per identity has been chosen for simplicity of the modeling and presentation. It can be extended in a natural way to handle the case where an entity may register and be authenticated via multiple public keys.

**Global Functionality $\bar{\mathcal{G}}_{\text{cert}}^{pid}$**

Parameterized by a party identity *pid*, global functionality $\bar{\mathcal{G}}_{\text{cert}}^{pid}$ proceeds as follows:

**Signature Generation:** Upon receiving a value $(\mathsf{Sign}, sid, m)$ from $P_{pid}$ do:

(a) Verify that $sid = (pid, sid')$ for some $sid'$. If not, then ignore the request.

(b) If this is the first request then do:

(i) If $P_{pid}$ is honest then generate a verification key (i.e., run the Key Generation procedure described in Figure 5). Upon receiving $(\mathsf{Verification\ Key}, sid, v)$ from the adversary, send $(\mathsf{Register}, pid, v)$ to $\bar{\mathcal{G}}_{\text{bb}}$ (done via an output to $P_{pid}$).

(ii) Else, check that $P_{pid}$ is registered in the $\bar{\mathcal{G}}_{\text{bb}}$ (i.e., send $(\mathsf{Retrieve}, pid)$ and verifying that $v \neq \perp$). If not, then ignore the request.

(c) Send $(\mathsf{Sign}, sid, m)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, 0)$ is recorded. If it is, then output an error message to $P_{pid}$. Else, output $(\mathsf{Signature}, sid, m, \sigma)$ to $P_{pid}$, and record the entry $(m, \sigma, 1)$.

**Signature Verification:** Upon receiving a value $(\mathsf{Verify}, sid, m, \sigma)$ from some party $P$, where $sid = (pid, sid')$ for some $sid'$, check whether a pair $(pid, v)$ is recorded. If not, send $(\mathsf{Retrieve}, pid)$ to $\bar{\mathcal{G}}_{\text{bb}}$, and obtain a response $(\mathsf{Retrieve}, pid, v)$. If $v = \perp$ then output $(\mathsf{Verified}, sid, m, 0)$. Else, record $(pid, v)$ and hand $(\mathsf{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, \phi)$ from the adversary do:

(a) If $(m, \sigma, b')$ is recorded then set $f = b'$.

(b) Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any $\sigma'$ is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.

(c) Else, set $f = \phi$, and record the entry $(m, \sigma, \phi)$.

Output $(\mathsf{Verified}, sid, m, f)$ to $P$.

**Corruption:** Upon receiving a value $(\mathsf{Corrupt}, sid)$ from the adversary, if $sid = (pid, sid')$ mark the party $P_{pid}$ as corrupt.

**Fig. 4:** The certification functionality. The certification functionality is parametrized by a party identity, referred to as the *owner*, and allows only that party to sign messages. The functionality generates a key for the owner when the first signing request arrives. This is done to advertise that party's existence; neither signature nor verification is done with respect to that key.

We model the certificate authority in a simplistic way, by associating each $\bar{\mathcal{G}}_{\text{cert}}$ with an owner PID, and providing certificates to any session of the owner. A more sophisticated modeling could have the certificate authority provide certificates according to some policy provided by the owner. For example, policies that allow sessions of other PIDs to generate certificates would capture a more refined notion of trust ("delegated signers").

**The certification with keys functionality.** The functionality $\bar{\mathcal{G}}_{\text{cwk}}$ is a GUC adaptation of the ideal signature functionality $\mathcal{F}_{\text{sig}}$ of [Can04] (formal description of $\mathcal{F}_{\text{sig}}$ can be found in Figure 5); it is used to realize the certification functionality. For an uncorrupted party it offers the capabilities of signing a message (reserved for the owner PID) and verifying a signature. It also captures the ways in which a corrupted party may deviate: as a signer, a corrupted party may refrain from registering the generated key in the bulletin-board, and as a verifier it may request verification of messages with

**Functionality $\mathcal{F}_{\text{sig}}^{pid}$**

Parameterized by a party identity *pid*, functionality $\mathcal{F}_{\text{sig}}^{pid}$ proceeds as follows:

**Key Generation:** Upon receiving a value $(\mathsf{KeyGen}, sid)$ from some party $P_{pid}$ verify that this is the first request and $sid = (pid, sid')$ for some $sid'$. If not, then ignore the request. Else, hand $(\mathsf{KeyGen}, sid)$ to the adversary. Upon receiving $(\mathsf{Verification\ Key}, sid, v)$ from the adversary, output $(\mathsf{Verification\ Key}, sid, v)$ to $P_{pid}$.

**Signature Generation:** Upon receiving a value $(\mathsf{Sign}, sid, m)$ from $P_{pid}$, verify that $sid = (pid, sid')$ for some $sid'$. If not, then ignore the request. Else, send $(\mathsf{Sign}, sid, m)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to $P_{pid}$ and halt. Else, output $(\mathsf{Signature}, sid, m, \sigma)$ to $P_{pid}$, and record the entry $(m, \sigma, v, 1)$.

**Signature Verification:** Upon receiving a value $(\mathsf{Verify}, sid, m, \sigma, v')$ from party $P$, where $sid = (pid, sid')$ for some $sid'$ verify that a pair $(pid, v)$ is recorded. If not, output $(\mathsf{Verified}, sid, m, 0)$ to $P$. Else, hand $(\mathsf{Verify}, sid, m, \sigma, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, \phi)$ from the adversary do:

(a) If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and $\sigma$ is a legitimately generated signature for $m$, then the verification succeeds.)

(b) Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any $\sigma'$ is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees unforgeability: If $v'$ is the registered one, the signer is not corrupted, and never signed $m$, then the verification fails.)

(c) Else, if there is an entry $(m, \sigma, v', f')$ recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)

(d) Else, let $f = \phi$ and record the entry $(m, \sigma, v', \phi)$

Output $(\mathsf{Verified}, sid, m, f)$ to $P$.

**Corruption:** Upon receiving a value $(\mathsf{Corrupt}, sid)$ from the adversary, if $sid = (pid, sid')$ then mark the party $P_{pid}$ as corrupt.

**Fig. 5:** The basic signature functionality [Can04]. The signature functionality is parametrized by a party identity and allows only this party to generate a key and sign messages. The owner can generate only a single key and sign only with respect to this key. Verifying a signature is done with respect to the signing key generated by the signature functionality. The functionality accepts verification requests from any ITI in the system. The signature functionality lets the adversary determine the signing key, the legitimate signatures, and the results of verifications that use an incorrect key or a different signature. When the signer is corrupted, the functionality allows the verification process to succeed, even if the message was never signed.

**Global functionality $\bar{\mathcal{G}}_{\text{cwk}}^{pid}$ for realizing $\bar{\mathcal{G}}_{\text{cert}}$**

The functionality $\bar{\mathcal{G}}_{\text{cwk}}^{pid}$ internally runs the code of $\mathcal{F}_{\text{sig}}$ and proceeds as follows:

**Signature Generation:** Upon receiving a value $(\mathsf{Sign}, sid, m)$ from $P_{pid}$, do:

(a) Verify that $sid = (pid, sid')$ for some $sid'$. If not, then ignore the request. (That is, verify that it is the legitimate signer for this $sid$.)

(b) If this is the first request then do:

(i) If $P_{pid}$ is corrupted, then verify that $P_{pid}$ is registered in $\bar{\mathcal{G}}_{\text{bb}}$ (otherwise, then ignore the request).

(ii) Generate a verification key, i.e., send $(\mathsf{KeyGen}, sid)$ to $\mathcal{F}_{\text{sig}}$. Upon receiving $(\mathsf{Verification\ Key}, sid, v)$, send $(\mathsf{Register}, pid, v)$ to $\bar{\mathcal{G}}_{\text{bb}}$ (done via an output to $P_{pid}$).

(c) Send $(\mathsf{Sign}, sid, m)$ to $\mathcal{F}_{\text{sig}}$. Upon receiving $(\mathsf{Signature}, sid, m, \sigma)$ from $\mathcal{F}_{\text{sig}}$, output $(\mathsf{Signature}, sid, m, \sigma)$ to $P_{pid}$.

**Signature Verification:** Upon receiving a value $(\mathsf{Verify}, sid, m, \sigma)$, where $sid = (pid, sid')$ for some $sid'$, check whether a pair $(pid, v)$ is recorded. If not, send $(\mathsf{Retrieve}, pid)$ to $\bar{\mathcal{G}}_{\text{bb}}$, and obtain a response $(\mathsf{Retrieve}, pid, v)$. If $v = \perp$ then output $(\mathsf{Verified}, sid, m, 0)$. Else record $(pid, v)$. Next, send $(\mathsf{Verify}, sid, m, \sigma, v)$ to $\mathcal{F}_{\text{sig}}$, and output the response $(\mathsf{Verified}, sid, m, f)$.

**Corrupted Signature Verification:** Upon receiving a value $(\mathsf{Verify}, sid, m, \sigma, v')$ from the adversary, where $sid = (pid, sid')$ for some $sid'$, send $(\mathsf{Verify}, sid, m, \sigma, v')$ to $\mathcal{F}_{\text{sig}}$, and output the response $(\mathsf{Verified}, sid, m, f)$.

**Corruption:** Upon receiving a value $(\mathsf{Corrupt}, sid)$ from the adversary, forward it to $\mathcal{F}_{\text{sig}}$.

**Fig. 6:** The certification with keys functionality. The functionality $\bar{\mathcal{G}}_{\text{cwk}}$ is parametrized by a party identity and internally executes the code of the basic signature functionality $\mathcal{F}_{\text{sig}}$. The functionality does not allow generating a key without signing a message. Key generation is done internally by the functionality. Note that keys of corrupted parties registered with $\bar{\mathcal{G}}_{\text{bb}}$ do not have to match the keys generated by $\mathcal{F}_{\text{sig}}$.

respect to keys of its choice (instead of the key registered in the bulletin-board). The only difference between the two formulations is the inability of a corrupted signer to generate a signing key without providing a message to be signed. Nonetheless, the capabilities of the attacker with respect to the formulations are equivalent. A formal description appears in Figure 6. We note that $\bar{\mathcal{G}}_{\text{cwk}}$ is a $\bar{\mathcal{G}}_{\text{bb}}$-subroutine respecting functionality, as defined in Section 2.

**Lemma 1.** *The functionality $\mathcal{G}_{\text{cwk}}$ EUC-realizes functionality $\mathcal{G}_{\text{cert}}$ and vice versa, with respect to adaptive corruptions.*

*Proof.* First we observe that as long as verification requests are done with the actual verification key, the functionalities are equivalent. To handle the other scenarios, we use the simulator's ability to postpone signature requests of corrupted signers up to the verification moment.

We begin by showing that $\mathcal{G}_{\text{cwk}}$ GUC-realizes functionality $\mathcal{G}_{\text{cert}}$. The simulation here is even simpler than in [Can04] due to the existence of $\bar{\mathcal{G}}_{\text{bb}}$ also in the ideal execution. We make use of an equivalent formulation of GUC-emulation with respect to

dummy adversaries. Thus, denoting the dummy adversary by $\mathcal{D}$, we wish to construct an adversary $\mathcal{S}$ such that:

$$\text{GEXEC}_{\mathcal{G}_{\text{cwk}},\mathcal{D},\mathcal{Z}} \approx \text{GEXEC}_{\mathcal{G}_{\text{cert}},\mathcal{S},\mathcal{Z}} \tag{2}$$

The adversary $\mathcal{S}$ is specified as follows. For signature generation, if the signer is honest then behave as the dummy adversary $\mathcal{D}$. That is, any output of $\mathcal{G}_{\text{cert}}$ and $\bar{\mathcal{G}}_{\text{bb}}$ is forwarded to $\mathcal{Z}$ and any input of $\mathcal{Z}$ is forwarded to $\mathcal{G}_{\text{cert}}$ or $\bar{\mathcal{G}}_{\text{bb}}$, in an appropriate manner. It also records the generated key $v$. If the signer is corrupted, $\mathcal{S}$ behaves as follows: for the first sign request it verifies that the signer is registered in $\bar{\mathcal{G}}_{\text{bb}}$ (if not it ignores the sign request) and simulates the key generation procedure. After recording the generated key $v$ it simulates the signature generation process, without involving $\mathcal{G}_{\text{cert}}$, and records the tuple $(m,\sigma,v,1)$ where $\sigma$ is the signature chosen by $\mathcal{Z}$ (except when a record $(m,\sigma,v,0)$ exists, in which case it outputs an error message). Note that $\mathcal{G}_{\text{cert}}$ does not receive any sign requests from a corrupted signer during the simulation of signature generation. Signing using $\mathcal{G}_{\text{cert}}$ is postponed, and executed only if a verification request is received for this record.

For signature verification, we simulate differently depending on the integrity of the signer and the key used by the verifier. If the signer is honest and some uncorrupted party makes a verification request (or a corrupted party that is using the key registered in $\bar{\mathcal{G}}_{\text{bb}}$) then do the following:

(a) behave as a dummy adversary $\mathcal{D}$ in the retrieve process (if executed).
(b) Once $(\mathsf{Verify},sid,m,\sigma)$ received, append the verification key, which is recorded in $\bar{\mathcal{G}}_{\text{bb}}$, and forward it to to the environment $\mathcal{Z}$. The response of $\mathcal{Z}$ is forwarded back to $\mathcal{G}_{\text{cert}}$. If in the output $f=0$ then record $(m,\sigma,v',0)$.

For corrupted signer, upon receiving a verification request from a honest verifier (or a corrupted verifier that is using the key registered in $\bar{\mathcal{G}}_{\text{bb}}$) do the following:

(a) behave as a dummy adversary $\mathcal{D}$ in the retrieve process (if executed).
(b) if a record $(m,\sigma,v',1)$ exists, where $v'$ is the key registered in $\bar{\mathcal{G}}_{\text{bb}}$, forward a sign request on $m$ to $\mathcal{G}_{\text{cert}}$, pick $\sigma$ to be the signature and delete the record.
(c) behave exactly as in the honest signer honest verifier scenario to emulate the communication with $\mathcal{Z}$. That is, append the verification key, which is recorded in $\bar{\mathcal{G}}_{\text{bb}}$, and forward it to to the environment $\mathcal{Z}$. The response of $\mathcal{Z}$ is forwarded back to $\mathcal{G}_{\text{cert}}$.

In case a verification request is made with a key that does not match the key registered in $\bar{\mathcal{G}}_{\text{bb}}$, independently of the signer's integrity, then simulate the verification process by giving $\mathcal{Z}$ the appropriate $(\mathsf{Verify},sid,m,\sigma,v'')$ and obtaining its response $\phi$. Next, if the tuple $(m,\sigma,v'',b')$ is recorded, set $\phi=b'$, else record $(m,\sigma,v'',\phi)$. In any case, output $(\mathsf{Verified},sid,m,\phi)$. It is important to note that verification requests with $v''\neq v$ are simulated without involving $\mathcal{G}_{\text{cert}}$.

Since the simulator does nor perform any cheating, the simulation is perfect. That is, the environment $\mathcal{Z}$'s view of an interaction with $\mathcal{S}$ and $\mathcal{G}_{\text{cert}}$ is distributed identically to its view of an interaction with parties running protocol $\mathcal{G}_{\text{cwk}}$ in the $\bar{\mathcal{G}}_{\text{bb}}$-hybrid model, even if $\mathcal{Z}$ is computationally unbounded.

Now we show the other direction: $\mathcal{G}_{\text{cert}}$ GUC-realizes functionality $\mathcal{G}_{\text{cwk}}$. Signature generation for a honest signer is simulated by behaving as a dummy adversary $\mathcal{D}$. If the signer is corrupted, we forward the signing request to $\mathcal{G}_{\text{cwk}}$ and pick the key for $\mathcal{F}_{\text{sig}}$ to be the key registered in $\bar{\mathcal{G}}_{\text{bb}}$. In the verification process, as before, retrieve is simulated by behaving as a dummy adversary. Upon receiving $(\mathsf{Verify}, sid, m, \sigma, v)$ from $\mathcal{G}_{\text{cwk}}$, the simulator drops $v$ and forwards the modified message to $\mathcal{Z}$. The response $(\mathsf{Verified}, sid, m, \phi)$ of $\mathcal{Z}$ is forwarded to $\mathcal{G}_{\text{cwk}}$. Note that the simulator ensures that the key in $\bar{\mathcal{G}}_{\text{bb}}$ is the same as the key registered in $\mathcal{F}_{\text{sig}}$ . Therefore, all simulated verification requests are made with respect to the correct key, and hence answered exactly as in the real execution. This follows from the functionalities being identical when the verification is done with the key recorded in $\mathcal{F}_{\text{sig}}$ .

### 4.2 Using EU-CMA signatures for certification

[Can04] shows that realizing $\mathcal{F}_{\text{sig}}$ is equivalent to being EU-CMA secure (existential unforgeability against chosen message attacks; [GMR88]). However, his theorem does not apply to a setting where the keys are reused by arbitrary protocols. This section extends the connection between ideal signatures and EU-CMA security to the GUC setting. Specifically, we show its equivalence to $\bar{\mathcal{G}}_{\text{cwk}}$.

**Unforgeable signatures.** A signature scheme is a triple of PPT algorithms $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$, where sig may maintain local state between activations.

**Definition 5 ([GMR88]).** *A signature scheme $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ is called* EU-CMA *if the following properties hold for any negligible function $\nu$ and all large enough values of the security parameter $\kappa$.*
***Completeness:*** *For any message $m$,* $\Pr\big[(s,v) \leftarrow \mathsf{gen}(1^\kappa); \sigma \leftarrow \mathsf{sig}(s,m); 0 \leftarrow \mathsf{ver}(m,\sigma,v)\big] < \nu(\kappa)$.
***Consistency:*** *For any $m$, the probability that $\mathsf{gen}(1^\kappa)$ generates $(s,v)$ and $\mathsf{ver}(m,\sigma,v)$ generates two different outputs in two independent invocations is smaller than $\nu(\kappa)$.*
***Unforgeability:*** *For any PPT forger $F$,* $\Pr[(s,v) \leftarrow \mathsf{gen}(1^\kappa); (m,\sigma) \leftarrow F^{\mathsf{sig}(s,\cdot)}(v); 1 \leftarrow \mathsf{ver}(m,\sigma,v)$ *and $F$ never asked* sig *to sign $m$*$] < \nu(\kappa)$.

**Signing module.** To capture re-usability of keys within different protocols, we describe a signing module that accepts sign requests from its owner PID. This module can be thought of as a *local* service process, physically running on some local machine, providing signing service to all authorized processes on this machine. This is formally described as an ideal functionality, denoted $\bar{\mathcal{G}}_{\Sigma}^{pid}$, parametrized by a signature scheme $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ and some party ID. The keys' re-usability is modeled by having the functionality be shared among different SIDs, as long as they are owned by the same PID. That is, the functionality $\bar{\mathcal{G}}_{\Sigma}^{pid}$ is a "local" subroutine of this PID and is not accessible by anyone else.

The signing module separates the signing capability from secret key knowledge, and hence allows greater flexibility in terms of corruptions. Corrupting the module captures the scenario of complete privacy loss; corrupting a principal in a single session that uses the module captures a weaker privacy loss, allowing the adversary to sign some messages but not arbitrary messages. In particular, corrupting a session that uses the

module does not provide the adversary with the secret key or with the ability to sign messages of other SIDs. The signing module could be generalized to be selective about which sign requests it honors (for example, as a function of the session id and message contents). For our purpose, it suffices to consider the basic module. Formal description of $\bar{\mathcal{G}}_{\Sigma}^{pid}$ appears in Figure 7.

**Functionality $\bar{\mathcal{G}}_{\Sigma}^{pid}$**

Parameterized by a security parameter $\kappa$, a key generation and sign functions gen and sign respectively, global functionality $\bar{\mathcal{G}}_{\Sigma}^{pid}$ proceeds thusly when running with party $P_{pid}$:

**Signature Generation:** Upon receiving a value $(\mathsf{Sign}, sid, m)$ from $P_{pid}$, do:

(a) Verify that $sid = (pid, sid')$ for some $sid'$. If not, then ignore the request.

(b) If this is the first request run $(s, v) \leftarrow \mathsf{gen}(1^{\kappa})$, record $(s, v)$.
In any case, output $(\mathsf{Signature}, sid, m, \mathsf{sig}(s, m), v)$.

**Corruption-module:** Upon receiving a value Corrupt-module from the adversary, output $s$ if recorded, otherwise ignore.

**Fig. 7:** The signing module. The functionality $\bar{\mathcal{G}}_{\Sigma}^{pid}$ is parametrized by a party identity and some signature scheme. The functionality generates a signing and verification keypair. The signing key is kept inside $\bar{\mathcal{G}}_{\Sigma}$ and used to handle signing requests. The verification key is given outside, similarly to $\bar{\mathcal{G}}_{\text{cwk}}$.

To our knowledge, this is the first modeling of authentication in a composable setting to feature SID-wise corruption; prior works used PID-wise corruptions exclusively.

**The equivalence.** A signature scheme $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ may be translated into a per-PID protocol $\pi_{\Sigma}^{pid}$ that "locally" uses $\mathcal{G}_{\Sigma}^{pid}$. This protocol localizes the signing/verification process and reduces trust in the setup. That is, it is no longer required to trust a global, accessible by many parties, signing functionality; instead, each party can trust merely his local signing module, which is running on his computer.

The protocol $\pi_{\Sigma}^{pid}$ proceeds as follows:

(a) When party $P$ receives an input $(\mathsf{Sign}, sid, m)$, it verifies that $sid = (P, sid')$ for some $sid'$. If not, it ignores the input. Next, it forwards $(\mathsf{Sign}, sid, m)$ to $\mathcal{G}_{\Sigma}^{pid}$. It obtains a verification key $v$ and a signature $\sigma$ on message $m$. If no key is registered, then forward $v$ to $\mathcal{G}_{\text{bb}}$ and outputs $(\mathsf{Signature}, sid, m, \sigma)$.

(b) When party $P$ receives an input $(\mathsf{Verify}, \widehat{sid}, m, \sigma)$, where $\widehat{sid} = (\widehat{pid}, sid')$, it checks whether a pair $(\widehat{pid}, v)$ is recorded. If not, send $(\mathsf{Retrieve}, \widehat{pid})$ to $\mathcal{G}_{\text{bb}}$ and obtain a response $(\mathsf{Retrieve}, \widehat{pid}, v)$. If $v = \perp$ then output $(\mathsf{Verified}, \widehat{sid}, m, 0)$. Else record $(\widehat{pid}, v)$. Next output $\big(\mathsf{Verified}, \widehat{sid}, m, \mathsf{ver}(m, \sigma, v)\big)$.

**Lemma 2.** *Let $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ be a signature scheme. If $\Sigma$ is EU-CMA, then $\pi_{\Sigma}^{pid}$ EUC-realizes $\mathcal{G}_{\text{cwk}}^{pid}$ with respect to adaptive corruptions.*

*Proof.* Note that here $\mathcal{G}_\Sigma$ is a subroutine of $\pi_\Sigma$ and not a global functionality that exists in the simulator's world. Let $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ be some EU-CMA signature scheme. Assume that $\pi_\Sigma^{pid}$ does not realize $\mathcal{G}_{\text{cwk}}^{pid}$, i.e., for any ideal-process adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can tell whether it is interacting with $\mathcal{G}_{\text{cwk}}^{pid}$ and $\mathcal{S}$ in the ideal process, or with $\pi_\Sigma^{pid}$ and the dummy adversary $\mathcal{D}$ in the real-life model. We show that $\Sigma$ violates Definition 5. Since $\mathcal{Z}$ succeeds for any $\mathcal{S}$, it also succeeds for the following "generic" $\mathcal{S}$. The constructed $\mathcal{S}$ directly communicates with $\mathcal{Z}$; then:

(a) Whenever $\mathcal{S}$ receives a message $(\mathsf{KeyGen}, sid)$ from $\mathcal{G}_{\text{cwk}}^{pid}$, it does: If $sid$ is not of the form $(pid, sid')$, then $\mathcal{S}$ ignores this request. Otherwise, $\mathcal{S}$ runs $(s,v) \leftarrow \mathsf{gen}(1^k)$, records $s$, and returns $(\mathsf{Verification\ Key}, sid, v)$ to $\mathcal{G}_{\text{cwk}}^{pid}$.

(b) Whenever $\mathcal{S}$ receives a message $(\mathsf{Sign}, sid, m)$ from $\mathcal{G}_{\text{cwk}}^{pid}$, if there is a recorded signing key $s$, then $\mathcal{S}$ computes $\sigma = \mathsf{gen}(s,m)$, and hands $(\mathsf{Signature}, sid, m, \sigma)$ back to $\mathcal{G}_{\text{cwk}}^{pid}$. Otherwise, it does nothing.

(c) Whenever $\mathcal{S}$ receives $(\mathsf{Verify}, sid, m, \sigma, v)$ from $\mathcal{G}_{\text{cwk}}^{pid}$, it returns $(\mathsf{Verified}, sid, m, \phi)$ where $\phi = \mathsf{ver}(m, \sigma, v)$.

(d) When $\mathcal{Z}$ requests to Corrupt-module, then $\mathcal{S}$ reveals the signing key $s$.

We show that $\Sigma$ is not EU-CMA secure. Assume that scheme $\Sigma$ is both complete and consistent (otherwise the theorem is proven). We demonstrate that it is not unforgeable, by constructing a forger $F$. This is done as follows. $F$ runs a simulated copy of $\mathcal{Z}$, and simulates for $\mathcal{Z}$ an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{G}_{\text{cwk}}$ (where $F$ plays the role of both $\mathcal{S}$ and $\mathcal{G}_{\text{cwk}}$ for $\mathcal{Z}$). However, in the first activation, instead of running gen to obtain the keys $(s,v)$, $F$ hands $\mathcal{Z}$ the public verification key $v$ in $F$'s input. Instead of running the signing algorithm to obtain $\sigma = \mathsf{gen}(s,m)$, $F$ asks its oracle to sign $m$ and obtains the signature $\sigma$. Whenever the simulated $\mathcal{Z}$ activates some uncorrupted party with input $(\mathsf{Verify}, sid, m, \sigma, v)$, $F$ checks whether $(m, \sigma)$ constitutes a forgery (i.e., whether $m$ was never signed before and $\mathsf{gen}(v, m, \sigma) = 1$). If $(m, \sigma)$ is a forgery, then $F$ outputs that pair and halts. Else it continues the simulation. If $\mathcal{Z}$ asks to corrupt the module then $F$ halts with a failure output.

We analyze the success probability of $F$. Let $B$ denote the event that, in a run of $\pi_\Sigma$ with $\mathcal{Z}$ with $sid = (pid, sid')$, the signer $S$ generates a public key $v$, and some party is activated with a verification request $(\mathsf{Verify}, sid, m, \sigma, v)$, where $\mathsf{gen}(m, \sigma, v) = 1$, and $S$ is uncorrupted and never signed $m$. Since $\Sigma$ is complete and consistent, we have that as long as event $B$ does not occur, $\mathcal{Z}$'s view of an interaction with $\pi_\Sigma$ is statistically close to its view of an interaction with $\mathcal{S}$ and $\mathcal{G}_{\text{cwk}}$ in the ideal process. (The views may differ in case of a completeness or consistency error, but these happen only with negligible probability.) However, we are guaranteed that $\mathcal{Z}$ distinguishes with non-negligible probability between the interaction with $\pi_\Sigma$ and the interaction with $\mathcal{S}$ and $\mathcal{G}_{\text{cwk}}$. Thus we are guaranteed that, when $\mathcal{Z}$ interacts with $\pi_\Sigma$, event $B$ occurs with non-negligible probability. It remains to observe that, from the point of view of $\mathcal{Z}$, the interaction with the forger $F$ looks the same as an interaction in the real-life model with $\pi_\Sigma$. Thus, we are guaranteed that event $B$ will occur with non-negligible probability. Notice that event $B$ can occur only before the signer $S$ is corrupted. This means that whenever event $B$ occurs, $F$ outputs a successful forgery.

### 4.3 Defining and realizing non-deniable message authentication

This section shows that the most basic PKI, i.e., bulletin-board, suffices for secure authentication, even if the keys are reused in other arbitrary protocols. This is similar to the last step of [Can04]'s construction, except that we use a weaker authentication functionality—one that lets the adversary obtain a signature of the 'authentication transaction'—to capture non-deniability. (The signature serves as a *transferable* 'proof of transaction'.)

We first formulate a non-deniable ideal authentication functionality $\mathcal{F}_{\text{cert-auth}}$. The non-deniability property is obtained via the usage of ideal certificates. Then, we show that the classic signature-based authentication protocol (presented in Figure 9) GUC-securely realizes this relaxed authentication functionality. Finally, using the composition theorem and the results of Sections 4.1 and 4.2, we obtain an authentication protocol using merely existentially-unforgeable signatures and a global bulletin-board.

**Functionality $\mathcal{F}_{\text{cert-auth}}$**

(a) Upon receiving an input $(\mathsf{Send}, S, R, sid, m)$ from ITI $S$, output $(\mathsf{Sent}, S, R, sid, m)$ to the adversary, and, after a delay, provide the same output to R and halt.
(b) Upon receiving a value $(\mathsf{Corrupt}, S, sid)$ from the adversary, mark $S$ as corrupted.
(c) Upon receiving a value $(\mathsf{Corrupt\text{-}send}, S, R, sid, m')$ from the adversary, if $S$ is marked as corrupted and an output was not yet delivered to $R$, then output $(\mathsf{Sent}, S, R, sid, m')$ to $R$ and halt.
(d) Upon receiving $(\mathsf{External\text{-}info}, S, R, sid, m')$ from the adversary, if an output was not yet delivered to $R$, then output $\big(\mathsf{Sign}, (S, (R, sid)), (m', sid, R)\big)$ to $\bar{\mathcal{G}}_{\text{cert}}^{S}$ (on behalf of $S$) and forward the response to the adversary.
(e) Upon receiving $(\mathsf{Corrupt\text{-}sign}, S, R, sid, m')$ from the adversary, if $S$ is marked as corrupted then output $\big(\mathsf{Sign}, (S, R, sid), (m', sid, R)\big)$ to $\bar{\mathcal{G}}_{\text{cert}}$ and forward the response to the adversary.

**Fig. 8:** The non-deniable authentication functionality. The adversarial ability to obtain legitimate signatures on messages of its choice makes the authentication non-deniable. Signatures are obtained by instructing the dummy party $S$ to communicate with $\bar{\mathcal{G}}_{\text{cert}}$.

**On capturing transferability.** Since the essence of transferability is that "anyone" may become convinced of the message that was authenticated, one might attempt to capture transferability by having $\mathcal{F}_{\text{auth}}$ disclose to any principal in the system, upon request, that an authentication took place; the identities of the originator and recipient; and the contents of the authenticated message. This modeling allows any principal in the system to become convinced in the contents of the authenticated message and the identities of its originator and recipient. However, this modeling of authentication poses unnecessary requirements on the implementing protocol, such as supporting inquiries by third parties in an authenticated manner.

**The non-deniable authentication functionality.** The functionality $\mathcal{F}_{\text{cert-auth}}$, presented in Figure 8, is a non-deniable version of the authentication functionality of

[Can04]. The non-deniability of the functionality is captured by allowing the adversary to request signatures on messages affixed with $\mathcal{F}_{\text{cert-auth}}$'s session id (SID). Including the SID in the signed message binds the signature to the execution at hand, and prevents the adversary from reusing the signatures in other sessions. Later, any entity can verify this signature and be convinced that this message was indeed sent from $S$ to $R$. Our $\mathcal{F}_{\text{cert-auth}}$ is a $\bar{\mathcal{G}}_{\text{cert}}$-subroutine-respecting functionality. We highlight that the signature provided during the authentication process includes the identity of the intended recipient and the session identifier. This has two consequences: it does not guarantee the receiver deniability since it allows to publicly verify not only that a specific message was sent by some ITI, but also the intended recipient's identity; and it also prevents the adversary from relaying signatures between different sessions. The authentication functionality enables a corrupted sender to produce many signature on messages of its choice. This enables corrupting parties without corrupting their signing module. One could define, and realize by a similar protocol, a receiver-deniable version of $\mathcal{F}_{\text{cert-auth}}$. However, receiver-deniable authentication enables the adversary to reroute messages to a destination of its choice.

**Protocol $\phi_{\text{auth}}$ for realizing $\mathcal{F}_{\text{cert-auth}}$**

(a) Upon receiving an input $(\mathsf{Send}, A, B, sid, m)$, party $A$ sets $sid' = (A, B, sid)$, sets $m' = (m, sid, B)$, sends $(\mathsf{Sign}, sid', m')$ to $\bar{\mathcal{G}}_{\text{cert}}$, obtains the response $(\mathsf{Signature}, sid', m', \sigma)$, and sends $(sid, A, m, \sigma)$ to $B$.

(b) Upon receiving $(sid, A, m, \sigma)$, party $B$ sets $sid' = (A, B, sid)$, sets $m' = (m, sid, B)$, sends $(\mathsf{Verify}, sid', m', \sigma)$ to $\bar{\mathcal{G}}_{\text{cert}}$, and obtains a response $(\mathsf{Verified}, sid', m', f)$. If $f = 1$ then $B$ outputs $(\mathsf{Sent}, A, B, sid, m)$ and halts. Else $B$ halts with no output.

**Fig. 9:** The signature-based authentication protocol.

**Lemma 3.** *The protocol $\phi_{\text{auth}}$ GUC-emulates functionality $\mathcal{F}_{\text{cert-auth}}$ with respect to adaptive corruptions.*

*Proof.* The proof here is simpler than the proof of [Can04] due to having the certificate functionality in both the ideal and real executions.
Let $\mathcal{D}$ be the dummy adversary that interacts with parties running $\phi_{\text{auth}}$ in the $\bar{\mathcal{G}}_{\text{cert}}$-hybrid model. We construct an ideal-process adversary (simulator) $\mathcal{S}$ such that the view of any environment $\mathcal{Z}$ from an interaction with $\mathcal{D}$ and $\phi_{\text{auth}}$ is distributed identically to its view of an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{cert-auth}}$. The simulator $\mathcal{S}$ proceeds as follows.

**Simulating the sender.** When an uncorrupted party $A$ is activated with input $(\mathsf{Send}, sid, B)$, $\mathcal{S}$ obtains this value from $\mathcal{F}_{\text{cert-auth}}$. Then, $\mathcal{S}$ replies with $(\mathsf{External\text{-}info}, A, B, sid, m)$ and behaves as $\mathcal{D}$ in the interaction with $\bar{\mathcal{G}}_{\text{cert}}$. That is, $\mathcal{S}$ forwards to $\mathcal{Z}$ the message $\big(\mathsf{Sign}, (A, B, sid), (m, sid, B)\big)$ from $\bar{\mathcal{G}}_{\text{cert}}$, and forwards back to $\bar{\mathcal{G}}_{\text{cert}}$ the obtained signature $\sigma$. Next, $\mathcal{S}$ hands $\mathcal{Z}$ the message $(sid, A, m, s)$ sent from $A$ to $B$. If the sender is corrupted, then all that $\mathcal{S}$ has to do is to behave as the dummy party $\mathcal{D}$ in the interaction with $\bar{\mathcal{G}}_{\text{cert}}$.

**Simulating the verifier.** When $\mathcal{Z}$ instructs to deliver a message $(sid, A, \bar{m}, \sigma)$ to an uncorrupted party $B$, $\mathcal{S}$ first sends $\big(\mathsf{Verify}, (A, B, sid), (\bar{m}, sid, B), \sigma\big)$ to $\bar{\mathcal{G}}_{\text{cert}}$. If $\bar{\mathcal{G}}_{\text{cert}}$ outputs $\big(\mathsf{Verified}, (A, B, sid), (\bar{m}, sid, B), \sigma, f = 1\big)$ then do the following: if the sender is honest, then allow $\mathcal{F}_{\text{cert-auth}}$ to deliver the message which was sent in the ideal process to $B$. If the sender is corrupted, then forward $(\mathsf{Corrupt\text{-}send}, sid, \bar{m})$ to $\mathcal{F}_{\text{cert-auth}}$. In case $f = 0$ do nothing.

It is readily seen that the combined view of $\mathcal{Z}$ and $\mathcal{D}$ in an execution of $\phi_{\text{auth}}$ is distributed identically to the combined view of $\mathcal{Z}$ and $\mathcal{S}$ in the ideal process. Indeed, the only case where the two views may potentially differ is if the receiver obtains $(\mathsf{Verified}, sid', m', \sigma, f = 1)$ from $\mathcal{F}_{\text{cert-auth}}$ for an incoming message $(sid, A, m, \sigma)$, while $A$ is honest and never sent this message. However, if $A$ never sent $(sid, A, m, \sigma)$, then the message $m' = (m, sid, B)$ was never signed by $\bar{\mathcal{G}}_{\text{cert}}$ with session id $(A, B, sid)$; thus, according to the logic of $\bar{\mathcal{G}}_{\text{cert}}$, $B$ would always obtain $(\mathsf{Verified}, sid', m', \sigma, f = 0)$ from $\bar{\mathcal{G}}_{\text{cert}}$.

Now we are ready to fully instantiate the ideal functionalities used for authentication. The resulting authentication protocol is the signature protocol used in practice, which is depicted in Figure 1 along with the minimal PKI required for this task.

**Corollary 1.** *If EU-CMA signatures exist then protocol $\phi_{\text{auth}}^{\bar{\pi}_\Sigma/\bar{\mathcal{G}}_{\text{cert}}}$ GUC-realizes functionality $\mathcal{F}_{\text{cert-auth}}$ with respect to adaptive corruptions.*

*Proof.* By combining Lemma 1 with Theorem 2 we manage to reduce the security of $\mathcal{G}_{\text{cert}}$ to the security of $\pi_\Sigma$. This allows us to combine Lemma 3 with Theorem 5 and conclude that $\phi_{\text{auth}}$ GUC-realizes $\mathcal{F}_{\text{cert-auth}}$, where $\phi_{\text{auth}}$ uses $\bar{\pi}_\Sigma$ with $\bar{\mathcal{G}}_\Sigma$ instead of $\bar{\mathcal{G}}_{\text{cert}}$.

## 5 Non-deniable Key Exchange

We present a non-deniable key exchange functionality $\mathcal{F}_{\text{cert-ke}}$ and show that the classic signed-Diffie-Hellman protocol $\phi_{\text{ke}}$ (see ISO 9798-3, [CK01]), realizes it. The protocol $\phi_{\text{ke}}$ is presented in Figure 10.

**The non-deniable key exchange functionality.** The functionality, presented in Figure 11, is a key exchange functionality coupled with $\bar{\mathcal{G}}_{\text{cert}}$, similarly to $\mathcal{F}_{\text{cert-auth}}$. The main difference between our functionality and [DKSW09] is that we do not guarantee mutual authentication. That is, $\mathcal{F}_{\text{cert-ke}}$ allows a party to have a key also if the other party aborted before establishing a shared key.

**Lemma 4.** *Under the Decisional Diffie-Hellman (DDH) assumption, the protocol $\phi_{\text{ke}}$ GUC-emulates functionality $\mathcal{F}_{\text{cert-ke}}$ with respect to adaptive corruptions.*

*Proof.* Let $p$, $q$, $g$ be as in $\phi_{\text{ke}}$ and let $D = \{g^z\}_{z \in \mathbb{Z}_q^\star}$. We construct a simulator $\mathcal{S}$ that simulates the execution of the protocol with the dummy adversary $\mathcal{D}$ and environment $\mathcal{Z}$. The simulation of uncorrupted parties is done by honestly executing the protocol. That is, the simulator honestly generates the share of the secret key, and obtains the necessary certificates via $\bar{\mathcal{G}}_{\text{cert}}$ of the appropriate party. Once the simulation reaches the output step of party $A$, the simulator provides $\mathcal{F}_{\text{cert-ke}}$ with $(\mathsf{setkey}, sid, S, R, k')$ where $k'$ is set to be the simulated key. More formally,

**Protocol $\phi_{\text{ke}}$**

Parametrized by primes $p$ and $q$ such that $q \mid p-1$, and an element $g$ of order $q$ in $\mathbb{Z}_p^*$, protocol $\phi_{\text{ke}}$ proceeds as follows:

(a) Upon receiving an input $(\mathsf{keyexchange}, sid, A, B)$, party $A$ samples $x \xleftarrow{\$} \mathbb{Z}_q$, and sends $(sid, A, \alpha = g^x)$ to $B$.

(b) Upon receiving $(sid, A, \alpha)$, party $B$ samples $y \xleftarrow{\$} \mathbb{Z}_q$, sets $sid' = (A, (B, sid))$, sets $m' = (\alpha, \beta = g^y, sid, A, B)$, and sends $(\mathsf{Sign}, sid', m')$ to $\bar{\mathcal{G}}_{\text{cert}}^B$, obtains the response $(\mathsf{Signature}, sid', m', \sigma_B)$, sends $(sid, B, \beta, \sigma_B)$ to $A$; computes the key $k = \alpha^y$; and erases $y$.

(c) Upon receiving $(sid, B, \beta, \sigma_B)$, party $A$ sets $sid' = (A, (B, sid))$, sets $m' = (\alpha, \beta, sid, A, B)$, and sends $(\mathsf{Verify}, sid', m', \sigma_B)$ to $\bar{\mathcal{G}}_{\text{cert}}^B$, obtains the response $(\mathsf{Verified}, sid', m', f)$. If $f = 1$ then $A$ sends $(\mathsf{Sign}, sid', m')$ to $\bar{\mathcal{G}}_{\text{cert}}^A$, obtains the response $(\mathsf{Signature}, sid', m', \sigma_A)$, sends $(sid, A, \sigma_A)$ to $B$; computes the key $k = \beta^x$; erases $x$; and outputs $(\mathsf{setkey}, sid, A, B, k)$ and halts. Else $A$ halts with no output.

(d) Upon receiving $(sid, A, \sigma_A)$, party $B$ sends $(\mathsf{Verify}, sid', m', \sigma_A)$ to $\bar{\mathcal{G}}_{\text{cert}}^A$, obtains the response $(\mathsf{Verified}, sid', m', f)$. If $f = 1$ then $B$ outputs $(\mathsf{setkey}, sid, A, B, k)$ and halts. Else $B$ halts with no output.

**Fig. 10:** The non-deniable-authentication-based key exchange protocol.

(a) The simulator samples $x \xleftarrow{\$} \mathbb{Z}_q$ and outputs $(sid, A, \alpha = g^x)$ to $\mathcal{Z}$ as if it was sent by $A$.

(b) Upon receiving $(sid, A, \alpha')$ from $\mathcal{Z}$ as a message to be delivered to $\{0,1\}$ (recall that the channels are unauthenticated and hence $\mathcal{Z}$ can instruct $\mathcal{D}$ to deliver a different message instead). $\mathcal{S}$ samples $y \xleftarrow{\$} \mathbb{Z}_q$, sets $sid' = \big(A, (B, sid)\big)$, sets $m' = (\alpha', \beta = g^y, sid, A, B)$, and sends $(\mathsf{External\text{-}info}, B, sid', m')$ to $\mathcal{F}_{\text{cert-ke}}$, obtains the response $(\mathsf{Signature}, sid', m', \sigma_B)$, sends $(sid, B, \beta, \sigma_B)$ to $\mathcal{Z}$ as if this message was sent by $B$.

(c) Upon receiving $(sid, B, \beta', \sigma_B')$ from $\mathcal{Z}$, the simulator verifies the signature on $m' = (\alpha, \beta', sid, A, B)$ by sending an appropriate input to $\bar{\mathcal{G}}_{\text{cert}}^B$, If the signature is not verified, the simulation of $A$ stops. Otherwise, it sends $(\mathsf{External\text{-}info}, B, sid', m')$ to $\mathcal{F}_{\text{cert-ke}}$, obtains the response $(\mathsf{Signature}, sid', m', \sigma_A)$, and outputs $(sid, A, \sigma_A)$ to $\mathcal{Z}$. It also computes the key $k' = (\beta')^x$, gives input $(\mathsf{setkey}, sid, A, B, k')$ to $\mathcal{F}_{\text{cert-ke}}$, and instructs $\mathcal{F}_{\text{cert-ke}}$ to give output to $A$.

(d) Upon receiving $(sid, A, \sigma_A')$ the simulator verifies the signature on $m' = (\alpha', \beta, sid, A, B)$ by sending an appropriate input to $\bar{\mathcal{G}}_{\text{cert}}^A$. If the signature is not verified, the simulator halts. Otherwise, it instructs $\mathcal{F}_{\text{cert-ke}}$ to give output to $B$.

Upon corruption, the simulator reveals the secret information (if any) associated with the simulated transcript of the newly corrupted party. More concretely, if the environment requests to corrupt party $A$ or party $B$ before $A$ outputs the key, then $\mathcal{S}$ reveals the share $x$ or the simulated key $k'$ respectively; in any other case, it reveals the secret key $k$ provided to it by $\mathcal{F}_{\text{cert-ke}}$.

The analysis of $\mathcal{S}$ considers three possible scenarios:

(a) *No corruption case:* correctness can be violated by $\mathcal{Z}$ only with negligible probability. That is, the only way to have parties in the real execution output different keys is by forging a signature, which can happen negligibly often. In the ideal execution, correctness always holds and thence indistinguishability follows. Conditioned on $\mathcal{Z}$ not forging any signature, the view of $\mathcal{Z}$ in the real execution consists of $\{g^x, g^y, g^{xy}\}$ while in the simulated execution the view is $\{g^x, g^y, g^r\}$ for random $r$. If $\mathcal{Z}$ can distinguish the two executions with non-negligible advantage, then we can construct an adversary $\mathcal{A}$ that internally runs $\mathcal{Z}$ and breaks the DDH assumption.

(b) *Corruption after A produced an output:* this is similar to the no corruption case. After party $A$ produced an output, there is no secret information available (it is erased beforehand) and hence indistinguishability follows as in the no corruption case.

(c) *Corruption before A produced an output:* in both executions the outputted key is distributed identically, since in the ideal execution the uncorrupted party is honestly simulated and the output is set to be the simulated key. Moreover, the secret share $x$ of $A$ (revealed in case $\mathcal{Z}$ requests to corrupt party $A$ after the first message is sent) is distributed identically in both executions. □

**Functionality $\mathcal{F}_{\text{cert-ke}}$**

The functionality $\mathcal{F}_{\text{cert-ke}}$ parametrized by a domain $D$ proceeds as follows:

(a) Upon receiving message of the form $(\mathsf{keyexchange}, sid, S, R)$ from some ITI $S$, if this is the first activation, set $k = \perp$ and send $(\mathsf{keyexchange}, sid, S, R)$ to $\mathcal{S}$. (Otherwise, ignore the message).

(b) Upon receiving a value $(\mathsf{Corrupt}, sid, P)$ from $\mathcal{S}$, mark $P \in \{S, R\}$ as corrupted and output $k$ to $\mathcal{S}$.

(c) Upon receiving a message of the form $(\mathsf{setkey}, sid, S, R, k')$ from the adversary, if either $S$ or $R$ is corrupt, then set key $k = k'$, else set $k \xleftarrow{\$} \mathrm{D}$. Output a delayed message $(\mathsf{setkey}, sid, S, R, k)$ to $S$ and $R$ and halt.

(d) Upon receiving $(\mathsf{External\text{-}info}, P, sid, m')$ from the adversary, where $P \in \{S, R\}$, if $k \neq \perp$ and an output was not yet delivered to either party, output $\big(\mathsf{Sign}, (P, (P', sid)), (m', sid, P)\big)$ to $\bar{\mathcal{G}}^P_{\text{cert}}$ (where $P'$ is the other party), and forward the response to the adversary.

(e) Upon receiving $(\mathsf{Corrupt\text{-}sign}, sid, P, m')$ from the adversary, where $P \in \{S, R\}$, if $P$ is marked as corrupted then output $\big(\mathsf{Sign}, (P, (P', sid)), (m', sid, P)\big)$ to $\bar{\mathcal{G}}^P_{\text{cert}}$ and forward the response to the adversary.

**Fig. 11:** The non-deniable key exchange functionality $\mathcal{F}_{\text{cert-ke}}$. The functionality allows the adversary to request signatures on messages of its choice, together with the session and parties id. This behavior is allowed as long as the key it not outputted, to prevent the functionality from being used beyond the lifetime of the protocol.

## 6 Capturing Invisible Adaptive Attacks

Recently, Nielsen and Strefler [NS14] introduced a concept called Invisible Adaptive Attacks (IAA), which the GUC framework fails to capture, and showed how to immune the GUC model from such attacks, for CRS-style setup assumptions. An IAA is an attack wherein a protocol behave insecurely with respect to some specific values of the global setup, but continues to behave securely under other values of the setup. Since the setup is long-lived and fixed for the lifetime of the system, such protocols should be rejected by the security definition. However, at present, the security definition accepts such protocols, since it examines candidate protocols' behavior only with respect to the average case of the setup-generating algorithm.

The approach of [NS14] for capturing such attacks is to consider worst-case security, i.e., guarantee security with respect to any setup. This is incorporated in the GUC model by letting the environment pick the random coins the setup (e.g. a CRS) uses. For our protocols, IAA security boils down to letting the environment determine the random coins of $\bar{\mathcal{G}}_{\Sigma}^{pid}$. This additional power does not influence the security and the analysis of $\phi_{\text{auth}}$ and $\phi_{\text{ke}}$, since the only possible way to distinguish ideal from real is to forge a signature. However, since the environment is oblivious to the secret keys, its forging ability remains negligible and security continues to hold.

**An alternative defnition.** We also propose an alternative approach for defining security in a way that captures such "invisible attacks". Rather than defining security of a protocol against a worst-case choice of the set-up, we define security of a protocol *relative to a specific CRS,* or more generally *relative to a specific random input for the set-up functionality*. This way, it is possible o capture a setting where the same protocol is considered (or, believed to be) secure with resepect to some setup values, and insecure with respect to others. The approach is similar to the definition of security of a fixed hash function by Rogaway [Rog06]. That is, security is captured by a reduction from knowing a distinguishing environment (with respect to a specific setup value) to breaking a hard problem. This implies that the designer of a protocol is in charge of specifying the hard problem $P$ for the security reduction. The meaning of such reduction is that, as long as solving $P$ is believed to be hard, coming up with a distinguishing environment must be hard as well. It is stressed that here the existence of a reduction is *part of the definition of seccurity* rather than part of the security argument. Furthermore, $P$ can relate either to properties of the set-up itself, or alternatively to other constructs. More formally, let $P$ be some problem; denote by $G(P)$ the game corresponding to $P$; and let $B(P)$ be the probability bound on winning in $G(P)$. For a shared setup $\bar{\mathcal{G}}$ we denote by $str = (s, v)$ a value of $\bar{\mathcal{G}}$ with $s$ and $v$ being the secret and public parts respectively.

**Definition 6 (Reduction-UC).** *Let $\pi$ and $\phi$ be $\bar{\mathcal{G}}$-subroutine respecting multi-party protocols. We say that $\pi$ RUC-emulates $\phi$ with respect to a value $str = (s, v)$ of $\mathcal{G}$ and a problem $P$ if there exist an adversary $\mathcal{S}$ and a reduction $f$ such that for any environment $\mathcal{Z}$ such that if*

$$\text{EXEC}^{s\bar{t}r}_{\pi,\mathcal{D},\mathcal{Z}} \not\approx \text{EXEC}^{s\bar{t}r}_{\phi,\mathcal{S},\mathcal{Z}}$$

*we have that* $\Pr[f(\mathcal{Z}, v)$ *wins in* $G(P)] > B(P)$.

An important observation is that RUC-security implies GUC-security, and the composition theorem easily holds for RUC-security. More formally, the simulator is the same as in the composition theorem proof, the hard problem is the problem the subroutine is defined with respect to, and the reduction is done by running the composition proof to obtain a distinguishing environment for the subroutine protocol and applying to that environment the reduction guaranteed for the subroutine by the RUC security definition. Another important benefit of this definition is that it easily induces a standard GUC-security definition: all we need to do is consider a setup-generating algorithm instead of a specific fixed string. For example, for ACRS this would be the key-generation algorithm. It should be noted that all GUC secure protocols (that we are aware of) are already proven secure by the way of reduction to some hard problem, and therefore RUC-secure. For example, the proof of our authentication and key-exchange protocols is done by a reduction to EU-CMA signatures and the DDH assumption respectively.

## References


AF10. Frederik Armknecht and Jun Furukawa. On the minimum communication effort for secure group key exchange. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2010.

BCL+05. Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14–18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2005.

BFS+13. Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013.

BLdMT09. Mike Burmester, Tri Van Le, Breno de Medeiros, and Gene Tsudik. Universally composable RFID identification and authentication protocols. *ACM Trans. Inf. Syst. Secur.*, 12(4), 2009.

Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, December 2000. Revised edition, July 2013.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

Can04. Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–. IEEE Computer Society, 2004.

CDPW07. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

CG10. Ran Canetti and Sebastian Gajek. Universally composable symbolic analysis of Diffie-Hellman based key exchange. *IACR Cryptology ePrint Archive*, 2010:303, 2010.

CK01. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

CK02. Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

CR03. Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

DF12. Özgür Dagdelen and Marc Fischlin. Intercepting tokens: The empire strikes back in the clone wars. Cryptology ePrint Archive, Report 2012/537, 2012. `http://eprint.iacr.org/`.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DKSW09. Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2009.

EMV11. EMVCo, LLC. *Integrated Circuit Card Specifications for Payment Systems: Book 2: Security and Key Management*, November 2011. Version 4.3.

FAK08. Jun Furukawa, Frederik Armknecht, and Kaoru Kurosawa. A universally composable group key exchange protocol with minimum communication effort. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 392–408. Springer, 2008.

GMR88. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, April 1988.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

KL07. Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. Cryptology ePrint Archive, Report 2007/478, 2007. `http://eprint.iacr.org/`.

KMO+14. Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. (De-)Constructing TLS. *IACR Cryptology ePrint Archive*, 2014:20, 2014.

LBdM07. Tri Van Le, Mike Burmester, and Breno de Medeiros. Universally composable and forward-secure RFID authentication and authenticated key exchange. In Feng Bao and Steven Miller, editors, *ASIACCS*, pages 242–252. ACM, 2007.

MR11. Ueli Maurer and Renato Renner. Abstract cryptography. In Bernard Chazelle, editor, *ICS*, pages 1–21. Tsinghua University Press, 2011.

MTC13. Ueli Maurer, Björn Tackmann, and Sandro Coretti. Key exchange with unilateral authentication: Composable security definition and modular protocol design. *IACR Cryptology ePrint Archive*, 2013:555, 2013.

NS14. Jesper Buus Nielsen and Mario Strefler. Invisible adaptive attacks. Cryptology ePrint Archive, Report 2014/597, 2014. `http://eprint.iacr.org/`.

Rog06. Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

Sho99. Victor Shoup. On formal models for secure key exchange. RZ 3120, IBM Research, April 1999. Version 4, revised November 1999.

Wal08. Shabsi Walfish. *Enhanced Security Models for Network Protocols*. PhD thesis, Courant Institute of Mathematical Sciences, New York University, January 2008.