UNIVERSALLY COMPOSABLE ZERO-KNOWLEDGE
ARGUMENTS AND COMMITMENTS FROM SIGNATURE
CARDS

DENNIS HOFHEINZ, JÖRN MÜLLER-QUADE, AND DOMINIQUE UNRUH

Abstract. In the Universal Composability framework many cryptographic
tasks cannot be built from scratch. Additional “helping” functionalities are
needed to realise zero-knowledge or bit commitment. However, all the ad-
ditional functionalities presented in the literature so far have to be specially
designed as a “helping” functionality and cannot directly serve any other pur-
pose without endangering the universal composability.

In this work, we introduce the concept of catalysts. Informally a func-
tionality $C$ is a catalyst for a functionality $F$ if $F$ can be implemented given the
primitive $C$ and the functionality $C$ can still directly be used by other applica-
tions without any additional precautions.

We prove that catalysts exist for zero-knowledge and bit commitment. And,
what is more, we show that a signature card, which is in accordance with the
German law [Bun01] can be used as such a catalyst.

1. Introduction and Related Work

The framework of universal composability (UC) allows the modular design of
cryptographic protocols. A cryptographic application may be constructed from
ideal functionalities which are secure by assumption. These ideal functionalities can
later be replaced by real protocols which securely implement the ideal functionalities
in question [Can01].

However universal composability is a very strict notion of security and the cryp-
tographic tasks of zero-knowledge arguments as well as bit commitment schemes
cannot be built from scratch in such a framework [CF01, CKL03]. To implement
these tasks, additional “helping” functionalities are needed. One functionality pro-
posed for implementing zero-knowledge protocols and bit commitment is a publicly
known random string [CF01, DN02], a so called common reference string (CRS).
Drawbacks of the CRS approach [Pas03, BCNP04, HMQ04] led to protocols using
different “helping” functionalities, namely random oracles [HMQ04], a public key
infrastructure (PKI), or a key registration authority [BCNP04].

However all these helping functionalities have to be specially designed as a “help-
ing” functionality and cannot directly serve any other purpose without endangering
the universal composability. There exist protocols which can individually be im-
plemented with a CRS, but they cannot together be implemented with one single
CRS. A PKI set up to allow bit commitments can in general not be used as a PKI
by other applications.

In this work, we introduce the concept of catalysts. Informally a functionality $C$
is a catalyst for a functionality $F$ if $F$ can be implemented given the primitive $C$
and the functionality $C$ can still directly be used by other applications. The concept
of catalysts differs from that of a reusable CRS [CF01] or a protocol to stretch one CRS into multiple independent CRS [CR03]. A catalyst $C$ for $F$ can be used to implement $F$ while still allowing arbitrary other applications to access that same instance of $C$ without any additional precautions. This property of a catalyst is captured more formally by giving the environment machine direct access to the catalyst.

**Definition 1.1.** Let $\pi$ be a protocol realising the functionalities $F$ and $C$ using $C$. We say that $C$ is used as a catalyst if $\pi$ realises $C$ by just relaying all requests and the respective answers directly to the functionality $C$.

From this definition it is clear that a common reference string is not a catalyst in the protocols found in the literature. There the common reference string is usually present only in the real model, whereas in the ideal model this CRS may be chosen by the simulator to give him an advantage over the real adversary. This advantage makes the simulation possible: The simulator chooses the common reference string together with some trapdoor information allowing him e.g. extractability from zero-knowledge proofs without rewinding, which must be impossible in the real model. If the CRS is used as a catalyst then this CRS would be chosen by the ideal functionality $C$ even in the ideal model. Furthermore, the environment has relayed access to the CRS in the ideal model, which makes it impossible for the simulator to present a fake CRS.

In this work we prove that catalysts exist for zero-knowledge and bit commitment (and following [CLOS02] for all well formed functionalities). And, what is more, we show that a signature card, which is in accordance with the German law [Bun01] can be used as such a catalyst. This is of practical importance, as an infrastructure of signature cards is about to be set up in several nations of the EU. Our work proves that this infrastructure can be used to securely implement additional applications without negative side effects.

2. **Signature Cards**

A signature card is a tamperproof device which can be used to digitally sign documents with an existentially unforgeable signature scheme and which ensures that the secret key cannot be extracted from the card. These properties are demanded e.g. by the German signature law [Bun01].

The signature cards are issued by a registration authority which also keeps a register of the signature verification keys associated to the protocol participants. Therefore the ideal functionality $F_{\text{SigCard}}$ representing signature cards and the registration authority can be queried for the public keys of protocol participants.

Furthermore, we assume that a signature card can be used by only one application (subparty) at a time. In the functionality $F_{\text{SigCard}}$ this is ensured by commands to change the possession.

For a given signature scheme $\mathcal{S}$ with a key generation algorithm, a signing algorithm and a signature verification algorithm, the functionality $F_{\text{SigCard}}$ is specified as follows:

3. **Zero-Knowledge Arguments based on Signature Cards**

In this work we will give a protocol SC-ZK implementing the functionality $F_{\text{ZK}}$. In the following protocol we will make use of witness indistinguishable arguments
Functionality \( \mathcal{F}_{\text{SigCard}} \)

For a signature scheme \( \mathcal{S} \), \( \mathcal{F}_{\text{SigCard}} \) proceeds as follows, running with parties \( P_1, \ldots, P_n \) and an adversary \( \mathcal{S} \).

**Initialisation:** For each party \( P_i \), use the key generation algorithm of \( \mathcal{S} \) to generate a public-secret-key pair \( (pk_i, sk_i) \) and set \( \text{possessor}_i := \perp \).

**Get public key:** When receiving a message \((\text{getkey}, sid, j)\) from some subparty \( P \), send the public key \( pk_j \) to \( P \). Before delivering the key, ask the adversary (non-immediate delivery).

**Signature Generation:** Upon receiving a message \((\text{sign}, sid, m)\) from some subparty \( P_i \) of \( P \), if \( P = \text{possessor}_i \), generate a signature \( \sigma \) using \( sk_i \), store the tuple \( (i, m, \sigma) \), and send \((\text{signature}, sid, m, \sigma)\) to \( P_i \).

**Signature Verification:** Upon receiving a message \((\text{verify}, sid, P_i, m, \sigma)\) from \( P_i \) do: If a tuple \( (i, m, \sigma) \) is stored set \( f = 1 \) else set \( f = 0 \). Then if the public key of \( P_i \) was already delivered to \( P_j \) in some prior “Get public key” step, send \((\text{verified}, sid, m, f)\) to \( P_j \).

**Possession:** Upon receiving a message \((\text{seize}, sid)\) from subparty \( P \) (where \( P \) is a subparty of \( P \) or \( P = \mathcal{S} \)), if \( \text{possessor}_i = \perp \), set \( \text{possessor}_i := P \) and send \((\text{seized}, sid)\) to subparty \( P \). Otherwise send \((\text{occupied}, sid)\) to subparty \( P \).

**Dispossession:** Upon receiving a message \((\text{release}, sid)\) from subparty \( P \) (where \( P \) is a subparty of \( P \) or \( P = \mathcal{S} \)), if \( \text{possessor}_i = P \), set \( \text{possessor}_i := \perp \).

Figure 1. The signature functionality \( \mathcal{F}_{\text{SigCard}} \)

Functionality \( \mathcal{F}_{\text{ZK}} \)

\( \mathcal{F}_{\text{ZK}} \) proceeds as follows, running with a prover \( P \), a verifier \( V \) and an adversary \( \mathcal{S} \).

- Upon receipt of an input \((\text{prove}, sid, p, w)\) with \( p(w) = \text{true} \) from party \( P \), send \((\text{prove}, sid, |p|)\) to \( \mathcal{S} \). As soon as \( \mathcal{S} \) allows the delivery, send \((\text{proven}, sid, p)\) to \( V \). (Only the first prove-request is heeded, subsequent ones are ignored.)

Figure 2. The zero-knowledge proof functionality \( \mathcal{F}_{\text{ZK}} \)

of knowledge (WIAOK). These arguments enjoy three important properties: First, arguments are computationally convincing proofs, i.e., a computationally limited prover can make the verifier accept with only a negligible probability. Second, they are arguments of knowledge, which means that from a prover which is giving convincing arguments a witness can be extracted (usually via rewinding) \[BG93\]. Third, the arguments of knowledge are witness indistinguishable: The protocol runs are (computationally) indistinguishable for different witnesses \[Gol01\, Chap. 4.6\]. Such WIAOK exist under the assumption that one-way functions exist.

- All communication is done through a secure channel that only leaks the length of the messages\[^1\].
- The environment may access the functionality \( \mathcal{F}_{\text{SigCard}} \) through other subparties than the prover and verifier subparty.

\[^1\]These again can be implemented e.g., using authenticated channels and a composable key-exchange.
• When receiving an input \((prove, sid, p, w)\), where \(p\) is a predicate such that \(p(w)\) is true, the prover \(P\) sends \(p\) to the verifier \(V\).
• The verifier \(V\) seizes its signature card. If it cannot seize the card, it terminates.
• The verifier generates a random nonce \(N\) of \(k\) bit length (where \(k\) is the security parameter). This nonce is sent via a secure channel to the prover \(P\).
• The prover \(P\) requests the public key \(pk_V\) and \(pk_P\) of \(V\) and \(P\) from \(F_{SigCard}\). Then it seizes its signature card, signs \(w\), and releases its signature card. Then it proves to the verifier that there exists a triple \((w, s_w, s_N)\), such that \(\text{verify}_{pk_P}(1^k, s_w, w) \land p(w) \lor \text{verify}_{pk_V}(1^k, s_N, N)\) using a witness indistinguishable argument of knowledge (WIAOK).
• If the verifier accepts the argument of knowledge, it terminates with output \((proven, sid, p)\). Additionally, it releases its signature card in any case.

We further require that the length of the messages transmitted during the WIAOK only depends on the length of the predicate \(p\). Given any WIAOK, this can easily be achieved by padding the messages.

Possible variants of this protocol might include the use of timeouts, so that an unresponsive prover does not lead to an eternally locked card of the verifier. Or several proofs could be done in parallel, sharing one locked signature card. Note that the possibility of locking a signature card implies that the protocol SC-ZK can terminate unsuccessfully if some other application seizes the card. Therefore the definition of the property of non-triviality \([\text{CLOS02, BHMQU05}]\) has to be changed. Non-triviality for protocols in this work demands that the protocol terminates successfully if no party is corrupted, no message is blocked, and the signature card is not seized by any other application.

**Theorem 3.1 (Security of SC-ZK, informal statement).** If \(\mathcal{G}\) is an existentially unforgeable signature scheme, protocol SC-ZK using the functionality \(F_{SigCard}\) securely implements the functionalities \(F_{ZK}\) and \(F_{SigCard}\) with respect to static adversaries. Here \(F_{SigCard}\) is used as a catalyst.

The basic idea of the proof is as follows: If the verifier is corrupted, the simulator has to generate a realistic proof only knowing the predicate. However, the simulator has access to the corrupted verifier’s signature card, so it can generate a valid verifier signature \(s_N\) for \(N\) and thus prove \(\text{verify}_{pk_P}(1^k, s_w, w) \land p(w) \lor \text{verify}_{pk_V}(1^k, s_N, N)\) for arbitrary \(w\) and \(s_w\).

If the prover is corrupted, the simulator has to extract a valid witness for \(p\). Note that \(P\) can not possibly learn a valid verifier signature \(s_N\) for \(N\) before the end of the WIAOK (since the verifier’s signature card is possessed by the verifier between generation of \(N\) and the end of the proof). So it must know a witness \(w\) with \(p(w)\) and a signature \(s_w\) for \(w\). It can only learn such a \(s_w\) by using the signature card. Since the simulator learns all accesses by corrupted parties to the signature card, it also learns \(w\).
**Functionality $\mathcal{F}_{\text{COM}}$**

$\mathcal{F}_{\text{COM}}$ proceeds as follows, running with a sender $S$, a receiver $R$ and an adversary $A$.

1. Upon receiving an input $(\text{commit}, sid, m)$ from sender $S$, send the value $(\text{receipt}, sid)$ to the adversary. As soon as the adversary permits, send $(\text{receipt}, sid)$ to $R$.
2. Upon receiving an input $(\text{unveil}, sid)$ from $S$, send $(\text{unveil}, sid, m)$ to the adversary. As soon as the adversary permits, send $(\text{unveil}, sid, m)$ to the receiver.

All but the first commit and unveil messages are ignored.

**Figure 3.** The commitment functionality $\mathcal{F}_{\text{COM}}$

4. **Commitments based on Signature Cards**

We present the protocol SK-COM-ZK for commitments. It turns a computationally binding and computationally hiding non-oblivious commitment scheme $\mathcal{C}OM_0$ into a universally composable commitment scheme. To ensure extractability a technique introduced in [HMQ04] is used.

There are general constructions to obtain a universally composable commitment from universally composable zero-knowledge schemes. However, the protocol given below is especially efficient as no invocations of $\mathcal{F}_{\text{ZK}}$ are necessary in the commit phase. This is of interest for applications using many commitments most of which are never unveiled.

**Commit phase:**

- All communication is done through a secure channel that only leaks the length of the messages.
- The environment may access the functionality $\mathcal{F}_\text{SigCard}$ through other subparties than the sender and verifier subparty.
- Upon input $(\text{commit}, sid, m)$ the sender $S$ sends a message $(\text{start}, sid)$ to the receiver $R$.
- The receiver $R$ generates a random nonce $N$ of $k$ bit length (where $k$ is the security parameter). This nonce is sent via a secure channel to the sender $S$.
- The sender $S$ generates a signature $\sigma_m$ for $(N, m)$ and commits to $(N, m, \sigma_m)$ using randomness $r$, and a signature $\sigma_r$ for $(N, r)$ and commits to $(N, r, \sigma_r)$ using randomness $r'$ (thereby temporarily seizing the signature card, and terminating if this is not possible). (By “commits”, we mean that the sender runs the commitment protocol $\mathcal{C}OM_0$ with the receiver.)
- The receiver outputs $(\text{receipt}, sid)$.

**Unveil phase:**

---

2 With deterministic verification function

3 A non-oblivious commitment scheme is one where the sender of the commitment always knows the value it commits to. In other words, by rewinding the sender, one can extract the committed value (analogous to the definition of proofs and arguments of knowledge). Non-oblivious commitment schemes exist if one-way functions exist. See [Gol01] Section 4.9.2.1 for definitions and constructions.
Upon input \((\text{unveil}, \text{sid})\), the sender sends \(m\) to the receiver through the secure channel, and then using \(\mathcal{F}_{\text{ZK}}\) proves: There exist \(r, r', \sigma_m, \text{ and } \sigma_r\), s.t. 
\[
\text{verify}_{pk_S}(1^k, \sigma_m, (N, m)) \text{ and } \text{verify}_{pk_S}(1^k, \sigma_r, (N, r)) \text{ evaluate to true,}
\]
and that committing to \((N, m, \sigma_m)\) and \((N, r, \sigma_r)\) using randomness \(r\) and \(r'\) resp. results in the commitments transmitted in the commit phase.

When the receiver gets the information from \(\mathcal{F}_{\text{ZK}}\) that the statement from the preceding step is true, it outputs \((\text{unveil}, \text{sid}, m)\).

Similar to the case of \(\mathcal{F}_{\text{ZK}}\), we require that the length of all messages depends only on the length, but not on the content of \(m\), which can be enforced by a suitable padding.

The same variations (timeouts, parallel executions) as in the case of commitments are possible here, too.

**Theorem 4.1** (Security of SC-COM-ZK using \(\mathcal{F}_{\text{ZK}}\), informal statement). If \(S\) is an existentially unforgeable signature scheme\(^4\), protocol SC-COM using the functionalities \(\mathcal{F}_{\text{SigCard}}\) and \(\mathcal{F}_{\text{ZK}}\) securely implements the functionalities \(\mathcal{F}_{\text{COM}}\) and \(\mathcal{F}_{\text{SigCard}}\) with respect to static adversaries. Here \(\mathcal{F}_{\text{SigCard}}\) is used as a catalyst.

Proof idea: If the receiver is corrupted, the simulator only has to generate commitments to random data (of appropriate length) in the commit phase. In the unveil phase, since we use \(\mathcal{F}_{\text{ZK}}\), no actual proof is received by the receiver, so the simulator does not have to fake a proof. If the sender is corrupted, the simulator must extract the message \(m\). But since the sender has to sign both \((N, m)\) and the randomness \(r\) used for the first commitment using the card \(\text{during the commit phase}\)^5 these are learned by the simulator. Therefore the simulator knows candidates for \((N, m, \sigma_m)\) and \(r\) and can check which of them opens the first commitment, thus finally learning \(m\).

Now we can replace all calls to \(\mathcal{F}_{\text{ZK}}\) in SC-COM-ZK by SC-ZK, getting a protocol SC-COM. Using the composition theorem, which preserves the property of being a catalyst, and Theorem 3.1 we get

**Corollary 4.2** (Security of SC-COM, informal statement). If \(S\) is an existentially unforgeable signature scheme\(^6\), protocol SC-COM using the functionality \(\mathcal{F}_{\text{SigCard}}\) securely implements the functionalities \(\mathcal{F}_{\text{COM}}\) and \(\mathcal{F}_{\text{SigCard}}\) with respect to static adversaries. Here \(\mathcal{F}_{\text{SigCard}}\) is used as a catalyst.

---

\(^4\)With deterministic verification function

\(^5\)Here we use the fact, that the commitment is non-oblivious. Then, when the sender commits successfully to \((N, m, \sigma_m)\) or \((N, r, \sigma_r)\) one can use a knowledge-extractor to get \(\sigma_m\) and \(\sigma_r\). If the sender succeeds in unveiling without signing in the commit phase, the knowledge-extractor would output \(\sigma_m\) and \(\sigma_r\) that it did not sign and thus contradict the existential unforgeability of \(S\).

In fact, using a commitment scheme that is not non-oblivious, the protocol would in general be insecure: Assume that \(S\) is deterministic in the sense that for each public key and each message there is at most one valid signature. Construct a commitment scheme \(C'\) from a normal commitment scheme \(C\) as follows: To commit to \(v\), one commits using \(C\) to a formula \(\varphi\) that is satisfied only by \(v\) and to a proof \(\pi\) that \(\varphi\) has only one satisfying assignment. To unveil, one sends \(v\) and unveils \(\varphi\) and \(\pi\). This scheme \(C'\) is still a commitment scheme, but one can commit to \((N, m, \sigma_m)\) and \((N, r, \sigma_r)\) by simply describing \(\sigma_m\) and \(\sigma_r\) as the unique signature of \(m\) wrt. public key \(pk\). Only for the unveil phase the knowledge of the signatures is actually needed.

\(^6\)With deterministic verification function
4.1. **On the functionalities** $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{COM}}$. The functionalities $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{COM}}$ differ from those given in [Can01]. This has several reasons: First, for simpler presentation, we have restricted to the case where sender and recipient (resp., prover and verifier) are fixed. Further, we consider *secret* functionalities, i.e., the adversary does not learn the unveiled commitment or the predicate, respectively. Finally, due to changes in the scheduling in the current version of [Can05], the adversary is explicitly asked when to deliver. In [Can01] this was implicitly ensured by the scheduling.

Finally, the more complex definition of $\mathcal{F}_{\text{COM}}$ used in [HMQ04] is not necessary here, since we only consider static corruption.

5. **Non-Transferability**

Cryptographic mechanisms can themselves lead to a certain type of insecurity. For example, a signed message $m$ sent from a party $A$ to another party $B$ can also convince a third party $C$ of the fact that $A$ signed $m$. This is undesirable when $B$ is the only one that $A$ wants to send $m$ to. (Imagine $m$ = “Yes, my dear friend $B$, I really fancy $C$’s wife.” — If $B$ gets angry with $A$ now, he may blackmail $A$ with $A$’s signature to $m$.) Therefore, it is sometimes preferable to have zero-knowledge proofs and commitments which cannot be transferred together with a proof.

An additional advantage of the protocol proposed in this work over protocols based on a public CRS is this property of non-transferability. For the protocols proposed here it is even in the real model possible to generate fake zero-knowledge arguments or fake commitments which look valid to any third party. This allows a party to deny having generated a zero-knowledge proof or a commitment. Hence the real protocol generates no evidence which could be used against an uncorrupted initiator of a zero-knowledge argument or a commitment.

**Theorem 5.1** (informal statement). Protocol SC-ZK (which implements $\mathcal{F}_{\text{ZK}}$) and protocol SC-COM (which implements $\mathcal{F}_{\text{COM}}$) are non-transferable.

**Acknowledgements.** We thank the referee for pointing out a that the hiding and binding (but not non-oblivious) commitments are not sufficient for Theorem 4.1, cf. footnote 5.

**References**


Dennis Hofheinz, CWI, Cryptology and Information Security Group, Kruislaan 413, Amsterdam, The Netherlands
E-mail address: Dennis.Hofheinz@cwi.nl

Jörn Müller-Quade and Dominique Unruh, IAKS, Arbeitsgruppe System sicherheit, Fakultät für Informatik, Universität Karlsruhe, Germany.
E-mail address: muellerq@ira.uka.de, unruh@ira.uka.de