Scientific Annals of Computer Science vol. 21, 2011, pp. 1–36

UC and EUC Weak Bit-Commitments Using Seal-Once Tamper-Evidence

Ioana Boureanu[1] Serge Vaudenay[2]

Abstract

Based on tamper-evident devices, i.e., a type of distinguishable, sealed envelopes, we put forward weak bit-commitment protocols which are UC-secure. These commitments are weak in that it is legitimate that a party *could* cheat. Unlike in several similar lines of work, in our case, the party is not obliged to cheat, but he has ability to cheat if and when needed. The empowered party is the sender, i.e., the protocols are also *sender-strong*.

We motivate the construction of such primitives at both theoretical and practical levels. Such protocols complete the picture of existent receiver-strong weak bit-commitments based on tamper-evidence.

We also show that existent receiver-strong protocols of the kind are not EUC-secure, i.e., they are only UC-secure. Further, we put forward a second formalisation of tamper-evident distinguishable envelopes which renders those protocols and the protocols herein EUC-secure.

We finally draw most implication-relations between the tamper-evident devices, our weak sender-strong commitments, the existent weak receiver-strong commitments, as well as standard commitments.

The mechanisms at the foundation of these primitives are lightweight and the protocols yielded are end-to-end humanly verifiable.

Keywords: universal composability, tamper-evidence, commitment ...

1 Introduction

Why Tamper-Evidence and UC? A way to ensure strong security guarantees of a primitive is to show that it is secure in the UC (universal composability) framework [7, 9]. In this formalism, the security proven in one single session is inherited when the protocol is executed over multiple

[1]HEIG-VD, Switzerland, email: `ioana.carlson@heig-vd.ch`

[2]EPFL, Switzerland, email: `serge.vaudenay@epfl.ch`

parallel or sequential sessions. If only a minority of the participants are corrupted in a polynomial-time multi-party computation, then the corresponding functionality can be UC-realized. Another way to achieve such a strong ideal emulation is to give access to all participants to an extra helping functionality called setup [7]. The latter empowering of parties yields the so-called UC hybrid models. (The reader can consult Appendix A, for a short summary of the UC frameworks and of the techniques in a UC proof.)

UC hybrid models, with setups of tamper-evident (TE), tamper-resistant (TR) devices and other means of restricted/isolated communication/computation, have been employed to UC-realize bit-commitments, oblivious transfers, coin flipping, polling schemes [17, 19, 16, 20, 21, 23, 22, 4], etc. For example, in [17], Katz opens for the creation and exchange of stateful tamper-proof hardware tokens used in a commitment protocol, which is UC-secure under the DDH assumption. In [10], two-party computation can also be UC-realized, using only stateless tokens and assuming the existence of oblivious transfer. Mateus and Vaudenay [19] allow a more permissive flow of hardware TR devices from creators to users and backwards than the one in [10], yet they obtain very similar results in their trusted agent model [19]. Similar protocols are constructed by Moran and Naor, in [23], using tamper-resistant hardware tokens that can be passed in one direction only.

We note that the distinction of having UC-commitments which place the strength on the sender or, on the contrary, place their strength on the receiver has been underlined [23] within this context of using tamper-resistant hardware as UC-setup. Based on sealed envelopes and sealed locks, i.e., simpler, not tamper-resistant, but just tamper-evident devices, Moran and Naor designed UC-secure protocols [22, 20, 21]. All their constructions empowered the protocol-receiver with cheating powers, i.e., the sender was "weak". In that line, the tamper-evident devices were created by the receiver of the commitments; we will refer to this as *receiver-strong*.

Moran and Naor proved that a type of distinguishable envelopes were the least complex tamper-evident devices that would entail UC-secure weak bit-commitments (WBC) [22] (i.e., they put forward a hierarchy of simple tamper-evident devices, of which some were not sufficient for WBC).

In this paper, we will work in the UC framework as it was introduced in [7], i.e., the communication channels are assumed to be secure. Another possibility would be to work in the UC model as it was refined in later years [9], where the channels are insecure but authentication-of-origin needs to be assumed on top: enforced via an extra setup (which is most often the case), or built-in into the protocols. Like in the case of similar works reminded above, our results hold in the first mentioned UC model [7], and modification would be needed to operate in the second [9].

Why Sender-strength (SS)? Why weak bit-commitments (WBC)? Why SS WBC? "Is it preferable to trust Vic or Peggy? We do not know, but it sure is nice to have the choice. [5]"; this statement by Brassard, Chaum and Crépeau, enunciated in their seminal work on commitments, is still standing today. In fact, Moran and Naor in [22] formulate the open question of finding such lightweight, UC-secure (weak) bit-commitment protocols that in turn place their strength on the sender's side, i.e, *sender-strong*.

In fact, there are several practical and theoretical reasons that motivate the existence of *weak* bit-commitments and also specific ones calling for their *sender-strong* version. *Sender-strong weak bit-commitments* are firstly interesting in traditional theoretical lines (i.e., outside of the UC-framework), where they are easier to construct (see Section 3.5).

- In trapdoor commitments [15], the output of the commitment-phase conceals the committed value from an informational theoretical point of view. But, the sender is only computationally bound to this committed value and, when given a trapdoor, he can in fact open a commitment in any possible way. Such primitive are sufficient [15] (i.e., no perfectly binding commitment is needed) to build more interesting cryptographic protocols in traditional lines. From this set of protocols, we mention concurrently secure and resettable identification schemes [2], and deniable authentication protocols [13, 15]. The above protocols have at their basis the construction of specialized ZK proofs. In our case, we move from the knowledge of a trapdoor to empowering the sender via a convenient transcript. Not surprising, this sort of commitments are also used to build special kinds of ZK proofs. This is detailed in the point presented below.

- In a different setting, in [5] Brassard *et al.* showed that "chameleon" bit-commitments[3] would imply, in traditional cryptography, zero-knowledge (ZK) proofs of knowledge where the verifier sends independent bits. For these ZK proofs of knowledge (PoK) to be provable secure against adaptive adversaries, content-equivocable bit commitments are further needed [1] (i.e., in order to equivocate, the sender has at hand only a transcript of the communication between him and the receiver and nothing else).

[3]These are commitments where the sender could cheat at the decommitment phase if given extra information.

Practice also imposes situations of sender-strength. For instance, consider the cases where the sender/committer should not have to trust the receiver in any way (e.g., it should only be the sender/committer who is required to create and seal the envelopes used in an envelope-based commitment scheme). This may be the case if the receiver is thought to have access to side-channels attacks (i.e., the receiving voting authority uses some special techniques to change the values hidden inside envelopes without resealing). Or, further, take the example of anonymous auctioning protocols [11, 18, 14], where the receiver/auctioning-house and the sender/auctioneer mutually ignore their identities throughout most phases of the protocol. Hence, the receiver Bob should not start by sending to some committer Alice the envelopes to be used in her commitment (as Bob ignores Alice's existence), but Alice should in turn commit to the maximum bid that she intends to place by possibly using self-made, tamper-evident "envelopes".

Contributions & Further Motivation. In this paper, we pursue the following directions. Primarily, we create sender-strong UC-secure WBC, i.e., weak bit-commitments that place the (adversarial) strength on the committer side and that are UC-secure. For this, we use UC setup slightly different to that of [22]. I.e., for reasons to be discussed, we firstly needed to put forward a new formalisation of distinguishable envelopes. In Section 2.2, we also describe a hierarchy of ideal functionalities for sender-strong weak bit-commitments and then we UC-realize them. In this fashion, we can relate the WBCs UC-realized herein both with traditional weak bit-commitments [1] of theoretical importance (e.g., see our $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$), and with weak bit-commitments UC-created in [22] with distinguishable envelopes (see our $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$). The differences between these target-functionalities lie mainly in learning that equivocation is possible (yet not obligatory) at the commitment phase ($\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$) or the opening phase ($\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$) vs. cheating only when the committer has not yet been caught abusing the protocol ($\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$).

We present two functionalities of distinguishable envelopes, differing in that the second one models envelopes created for a designated recipient. We now motivate the choices made with respect to this. There are many ways to formalise tamper-evident containers [22], reflecting the different requirements of the possible physical implementations of such devices. Moran and Naor model distinguishable envelopes which allow for creator-forgeability (i.e, the creator of the envelope can re-seal it without breaking the tamper-evidence). We argue however that sender-strong (weak) commitments only make sense

if they are *computationally hiding* and somewhat binding, i.e., the receiver has no special abilities and the sender *could* equivocate his openings, if and when needed. Hence, the amplified *partially* hiding and partially binding weak commitments in [22] would not serve well the sender-strong setting. Moreover, we conjecture that it is not possible to construct *UC-secure, hiding* sender-strong bit-commitment protocols, using the TE envelopes in [22]. Thus, herein, we do not allow the resealing of envelopes by their creator. I.e., we model *seal-once* distinguishable tamper-evident envelopes (or, envelope allowing *one-seal* only), and [22] formalises a *multi-seal* TE envelope.
Note: It may seem easy to construct commitments using this formalisation, but –in the UC framework– this is not trivial at all. We also show eventually that many of these formalisation "collapse" in one another. Please see Section 3.5 and Section 4 for details.

We noted that the protocols built with envelopes à la Moran and Naor [22] were not EUC-secure, i.e., only UC-secure. We concluded that if we further allowed for purported destinator of envelopes, then the corresponding DE-based protocols obtained both here and in [22] lead to protocol which are EUC-secure and not only UC-secure. Our second distinguishable envelope functionality ($\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$) serves this very purpose.

We finally draw several implication-relations between the following: our first functionality of distinguishable envelopes ($\mathcal{F}^{DE}_{\mathtt{OneSeal}}$), the standard UC-functionality of bit-commitment ($\mathcal{F}^{BC}$) and those of WBC (already existing and newly introduced herein).

Note: Part of this material was present in [3]. Notably, none of the proofs or the extended explanations on the protocols or functionalities were included in that version; these are introduced for the first time in here. Also, the `OpenEnablesCheat` in the current manuscript protocol differs from in [3].

2 Setup and Target UC Functionalities

We begin by formalising tamper-evident distinguishable envelope through an ideal UC functionality, which is similar to the one formalised in [22]. To relate more closely to Moran and Naor's work [22], we then introduce a weak bit-commitment functionality $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$, $q \in (0, 1)$, which is similar to that of [20, 22]. In this functionality, a sender decides whether to cheat at the very beginning (i.e., in a protocol, at the phase of envelope sealing) and the probability of potential cheating is controlled by the fact that the sender can be caught in certain cases (i.e., in a protocol, due to certain choices by the receiver). Then, we give functionalities $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ and $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$, which are different from $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ (i.e., a sender can decide to equivocate his commitment only at some point during the commitment phase or at some point during the opening phase, respectively). These $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ and $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionalities are closer to standard weak bit-commitments [12, 1] and are better suited to both the theoretical and practical motivations mentioned in the introduction (e.g., the sender only decides to cheat in his commitments within an auctioning protocol once the receiver has already proven to be untrustworthy).

2.1 UC-Setup Functionalities Modelling Tamper-Evident Envelopes

The $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ Functionality for Tamper-Evident Distinguishable Sealed Envelopes

The functionality $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ models a tamper-evident "envelope", distinguishable by some obvious mark (e.g., barcode, serial number, colour, etc.). Protocol parties can simply open such containers, but *any such* opening will be obvious to other parties who receive the "torn" envelope.

The functionality stores a table of the form (id, $value$, $holder$, $state$), indexed by id. For one fixed identifier id, the corresponding values are denoted as follows: $value_{id}$, $holder_{id}$ and $state_{id}$. In the presence of parties $P_1, \ldots, P_n$ and an ideal adversary $\mathcal{I}$, a run of the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ ideal functionality is described as follows.

Seal(id, $value$). Let this command be received from party P_i. It creates and seals an envelope. If this is the first **Seal** message with id id, the functionality stores the tuple (id, $value$, P_i, **sealed**) in the table. If this is not the first command of type **Seal** for envelope id, then the functionality halts.

Send(id, P_j). Let this command be received from party P_i. This command encodes the sending of an envelope held by P_i to a party P_j. Upon receiving this command from party P_i, the functionality verifies that there is an entry in its table which is indexed by id and has $holder_{id} = P_i$. If so, it outputs (**Receipt**, id, P_i, P_j) to P_j and $\mathcal{I}$ and replaces the entry in the table with (id, $value_{id}$, P_j, $state_{id}$).

Open id. Let this command be received from party P_i. This command encodes an envelope being opened by the party that currently holds it. Upon receiving this command, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it sends (**Opened**, id, $value_{id}$) to P_i and $\mathcal{I}$. It also replaces the entry in the table with (id, $value_{id}$, $holder_{id}$, **broken**).

Verify id. Let this command be received from party P_i. This command denotes P_i's verification of whether or not the seal on an envelope has been broken. The functionality verifies that an entry indexed by id appears in the table and that $holder_{id} = P_i$. It sends (**Verified**, id, $state_{id}$) to P_i and to $\mathcal{I}$.

One difference from the corresponding functionality presented in [22] is that the creator of an envelope cannot re-seal it, i.e., he cannot forge the value stored initially inside the envelope. Hence, the wording "`OneSeal`" refers the above functionality and the expression "`MultiSeal`" denotes the TE envelopes in [22]. (For consistency, in Appendix B, the reader can find the tamper-evident envelope functionality corresponding to [22]. We denote it $\mathcal{F}^{DE}_{\texttt{MultiSeal}}$.)

Based only on creator-forgeable/multi-seal tamper-evident envelopes, we were not able so far to UC-realize sender-strong weak bit-commitments as we wanted them, i.e., UC-simulatable somewhat binding and *not partially hiding, but computationally hiding* bit commitment. We have not yet refuted the existence of such a construction either.

Our change from $\mathcal{F}^{DE}_{\texttt{MultiSeal}}$ to $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ brings the latter functionality closer to regular commitment than the tamper-evident functionality in [22] was. It is relatively easy to see that *regular* bit-commitments can be immediately constructed using one distinguishable tamper-evident envelope (see Section C of the Appendix, for the $\mathcal{F}^{BC}$ UC-functionality of regular bit-commitments). The relation with the regular commitment functionality is however not symmetric, as Section 4 will detail (i.e., if $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ implies BC, it is not necessarily the case that $\mathcal{F}^{BC}_{\texttt{OneSeal}}$ implies DE). But, as we said in the introduction, it is of stand-alone theoretical importance to be able to construct *"error-tolerant"* bit-commitments which are sender-strong, i.e., *q-weak bit-commitments.*

Another, less significant difference from the corresponding functionality presented in [22] is that the $\mathcal{F}^{DE}$ introduced above does not output tuples containing the creator's identity. This would have been of no interest for the protocols constructed in the following and would hinder EUC-security proofs given later.

2.2 Target UC Functionalities of Bit-Commitment

We now describe our target functionalities $\mathcal{F}^{q-\text{WBC}}_{\star}$ that respectively model different weak bit-commitment (WBC) protocols, where we have $\star \in \{\texttt{EscapeThenMayCheat}, \texttt{LearnAtCommitment}, \texttt{LearnAtOpening}\}$. In this fashion, we can relate the WBCs UC-realized herein both with traditional weak bit-commitments [1] of theoretical importance (e.g., see our $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$), and with weak bit-commitments UC-created in [22] with distinguishable envelopes (see our $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$). The differences between these target-functionalities lie mainly in learning that equivocation is possible (yet not obligatory) at the commitment phase ($\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtCommitment}}$) or the opening phase ($\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$) vs. cheating only when the committer has not yet been caught abusing the protocol ($\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$).

The $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality idealising q-weak bit-commitment.

Let $q \in (0,1)$. The functionality maintains a variable *bit*, where *bit* ranges over $\{0,1,\Box\}$.

Commit b. When the **Commit** b command ($b \in \{0,1\}$) is sent to the functionality by a sender S, the value b is recorded in the variable *bit*. The $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality outputs **Committed** to the receiver R and to the ideal adversary $\mathcal{I}$. Further commands of this type or of type **EquivocatoryCommit** below are ignored by the functionality.

EquivocatoryCommit. When the **EquivocatoryCommit** command is sent to the functionality, the $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality replies to the sender and the ideal adversary with a $\perp$ message, with probability $1-q$. With probability q, the functionality sets the variable *bit* to the value $\Box$, outputs **Committed** to the sender, the receiver and to the ideal adversary. Further commands of this type or of type **Commit** above are ignored by the functionality.

AbortCommit. When the **AbortCommit** command is sent to the functionality, the $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality replies to the sender, to the receiver, and to the ideal adversary with a $\perp$ message (denoting an abnormal end of the execution). Further commands are ignored.

Open. Upon receiving the command **Open** from the sender, the functionality verifies that the sender has already sent the **Commit** b command. Then, the $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality outputs (**Opened**, *bit*) to the receiver and to the ideal adversary. Further commands are ignored by the functionality.

EquivocatoryOpen c. Upon receiving the **EquivocatoryOpen** c command from the sender, with $c \in \{0,1\}$, the functionality verifies that $bit = \Box$. Then, the functionality $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ outputs (**Opened**, c) to the receiver and to the ideal adversary. Further commands are ignored by the functionality.

In this functionality, the binding property of commitments can be defied. It corresponds to the weak bit-commitment functionality used by Moran and

Naor [22], but it applies to the sender-strong case. In that sense, a dishonest player decides to try and open his commitment to any value even from the very beginning of the protocol and he can be successful in doing so with a probability of $q \in (0,1)$, once he has not been caught red-handed.

Note that the WBC functionality presented above and the ones to be presented further model single bit commitments. Yet, they can easily be extended to respective functionalities for multiple commitments: i.e., each **Commit** b command sent by a sender S aimed at a receiver R would become **Commit**(id, b, R) and each corresponding functionality would store a tuple $(id, sender, receiver, value)$ for each commitment, doing the respective checks.

The $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ functionality idealising q-weak bit-commitment.

Let $q \in (0,1)$. The functionality maintains a tuple $(bit, equiv)$, where bit ranges over $\{0,1\}$ and $equiv$ ranges over $\{$“Yes”, “No”$\}$.

Commit b. When the **Commit** b command ($b \in \{0,1\}$) is sent to the functionality, the value b is recorded in the variable bit. With probability q the value “Yes” is stored in $equiv$ or, with probability $1-q$ the value “No” is stored in $equiv$. The $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ functionality outputs **Committed** to the receiver and to the ideal adversary. The $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ functionality outputs the updated value of $equiv$ to the sender and to the ideal adversary. Further commands of this type are ignored by the functionality.

Open. Upon receiving this command, the functionality verifies that the sender has already sent the **Commit** b command. Then, the $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ functionality outputs (**Opened**, bit) to the receiver and to the ideal adversary. Further commands are ignored by the functionality.

EquivocatoryOpen. Upon receiving this command, the functionality verifies that the sender has already sent the **Commit** b command. Then, the functionality checks the value of $equiv$. If the value is “Yes”, then $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ outputs (**Opened**, $\overline{bit}$) to the receiver and to the ideal adversary. If the value is “No”, then $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ halts. Further commands are ignored by the functionality.

The $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ functionality mirrors a protocol which allows the sender to cheat by breaking the binding property of the protocol. Note that this cheating possibility is “decided” at the commitment phase, i.e., it is at some point during the commitment phase that the potential cheater *learns* about his opportunity. Also, note that while the cheating is allowed, it does not necessarily need to happen (i.e., there are two distinct opening commands).

In the next functionality equivocation becomes clear only at the opening phase.

The $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionality idealising q-weak bit-commitment.

Let $q \in (0, 1)$.

The functionality maintains a variable *bit*, ranging over $\{0, 1\}$.

Commit. When the **Commit** b command ($b \in \{0, 1\}$) is sent to the functionality, the value b is recorded in the variable *bit*. The $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionality outputs **Committed** to the receiver and to the ideal adversary. Further commands of this type are ignored by the functionality.

Open. Upon receiving this command, the functionality verifies that the sender has already sent the **Commit** b command. Then, the $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionality outputs (**Opened**, *bit*) to the receiver and to the ideal adversary. Further commands are ignored by the functionality.

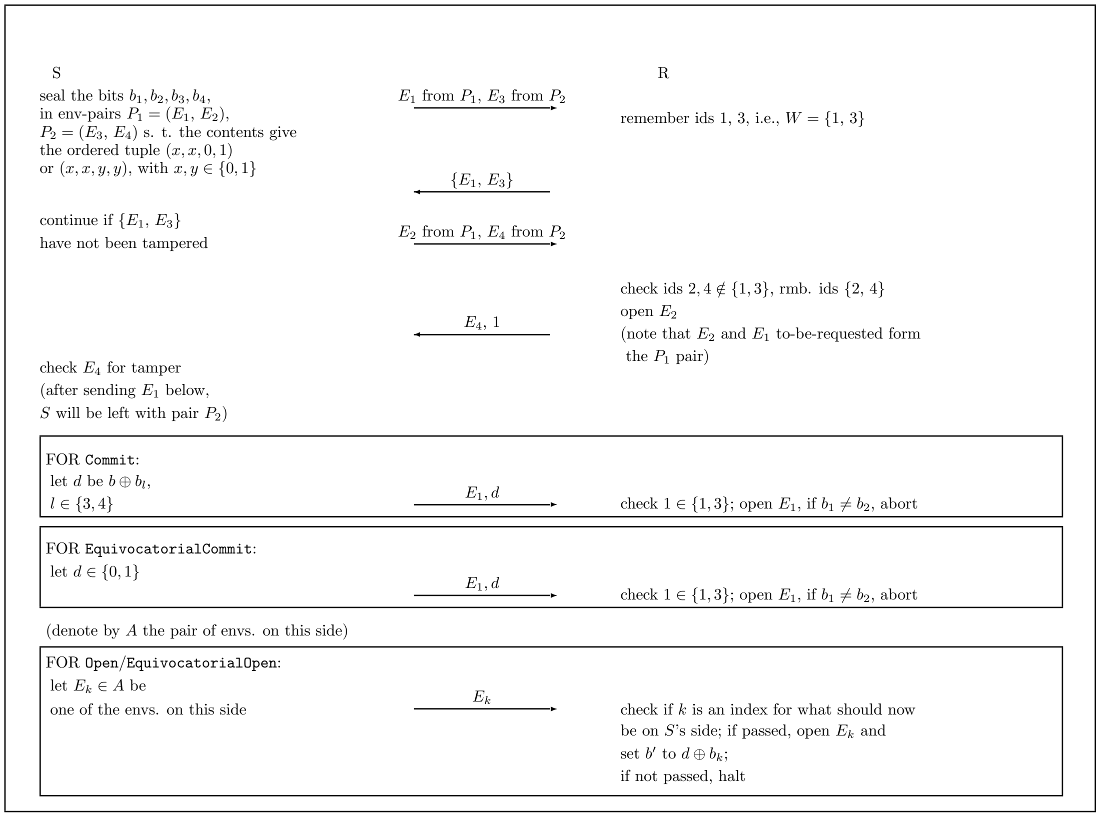
EquivocatoryOpen. Upon receiving this command, the functionality verifies that the sender has already sent the **Commit** b command. With probability q, the $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ outputs (**Opened**, $\overline{bit}$) to the receiver and to the ideal adversary. With probability $1 - q$, the $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ sends $\perp$ to the sender S and the ideal adversary $\mathcal{I}$. Further commands of this type are ignored by the functionality (but commands of type **Open** are still allowed).

The $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionality mirrors a protocol which allows the sender to cheat by breaking the binding property of the protocol, knowingly at some point during the opening phase, i.e., it is at some point during the opening phase that the potential cheater *learns* about his opportunity, similarly to traditional lines in [1]. As aforementioned, note that while the cheating is allowed, it does not necessarily need to happen.

Sender-strong (amplifiable) weak bit-commitments protocols with distinguishable, tamper-evident envelopes that allow only partial hiding are of course easier to UC-construct than those that require perfect hiding (see Section 3.5). As aforementioned, we believe however that sender-strength brings with it the need for more than just partially hiding primitives, hence the "*computationally hiding*" UC functionalities advanced above. We seek to UC-realize just protocols, which proves to be non-trivial.

3 UC (Sender-Strong) Bit-Commitments

Driven by the theoretical and practical motivations presented in the introduction, we now give WBC protocols which are UC-secure, sender-strong



S

seal the bits b_1, b_2, b_3, b_4, in env-pairs $P_1 = (E_1, E_2)$, $P_2 = (E_3, E_4)$ s. t. the contents give the ordered tuple $(x, x, 0, 1)$ or (x, x, y, y), with $x, y \in \{0, 1\}$

$\xrightarrow{E_1 \text{ from } P_1,\ E_3 \text{ from } P_2}$

R

remember ids 1, 3, i.e., $W = \{1, 3\}$

$\xleftarrow{\{E_1,\ E_3\}}$

continue if $\{E_1, E_3\}$ have not been tampered

$\xrightarrow{E_2 \text{ from } P_1,\ E_4 \text{ from } P_2}$

check ids $2, 4 \notin \{1, 3\}$, rmb. ids $\{2, 4\}$
open E_2
(note that E_2 and E_1 to-be-requested form the P_1 pair)

$\xleftarrow{E_4,\ 1}$

check E_4 for tamper
(after sending E_1 below,
S will be left with pair P_2)

FOR `Commit`:
let d be $b \oplus b_l$,
$l \in \{3, 4\}$

$\xrightarrow{E_1, d}$

check $1 \in \{1, 3\}$; open E_1, if $b_1 \neq b_2$, abort

FOR `EquivocatorialCommit`:
let $d \in \{0, 1\}$

$\xrightarrow{E_1, d}$

check $1 \in \{1, 3\}$; open E_1, if $b_1 \neq b_2$, abort

(denote by A the pair of envs. on this side)

FOR `Open`/`EquivocatorialOpen`:
let $E_k \in A$ be
one of the envs. on this side

$\xrightarrow{E_k}$

check if k is an index for what should now be on S's side; if passed, open E_k and set b' to $d \oplus b_k$;
if not passed, halt

Fig.1: The `Pass&MayCheat` Protocol

and use TE distinguishable envelopes as setups.

We will present the protocols `Pass&MayCheat`, `CommitEnablesCheat` and `OpenEnablesCheat` which respectively UC-realize $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$, $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommitment}}$ and $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$, using $\mathcal{F}^{DE}_{\texttt{OneSeal}}$. We then present amplification techniques of such weak BC protocols. The techniques maintain the lightweight character of the constructions. We conclude the section by a strengthening of the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality such that we attain EUC-security [9], i.e., not only UC-security.

3.1 `Pass&MayCheat` — a SS-WBC protocol à la Moran and Naor [22]

The `Pass&MayCheat` Protocol (see Fig.1):

The illustration of the protocol in Fig. 1 is given in a symmetric way (i.e., E_1 and E_2 could be interchanged in their appearances, etc.).

The Phase of Commitment and/or EquivocatoryCommitment.

A sender S seals four envelopes and creates two pairs out of them such that each pair contains the set $\{x, x\}$ of values, for a randomly fixed $x \in \{0, 1\}$. For a never-cheating sender, each pair "contains" its own value x. For a sender who may equivocate later, one pair contains the set $\{x, x\}$ of values, for a randomly fixed $x \in \{0, 1\}$ and the other pair contains the values $\{0, 1\}$. The senders sends two envelopes, one from each pair, to the receiver R. (E.g., The pairs are: 1st pair $P_1 = (E_1, E_2)$, 2nd pair $P_2 = (E_3, E_4)$ and S sends, e.g., E_1, E_3 to R). Then, the receiver R stores the identifiers of the envelopes in a register W. (I.e., it stores $(1, 3)$, given the illustrated execution by S in Fig.1.). Then, R sends them back without opening them.

At this step, the sender S verifies that the recently returned envelopes have the seals unbroken. If this is not so, he halts. Otherwise, he sends the two envelopes not sent before. (I.e., If seals are unbroken, then S sends the remaining E_2, E_4 as per Fig.1 .)

At step four, the receiver verifies that the envelopes received do not have the ids stored already. If their ids have been already stored, he halts. Otherwise, he opens one of these envelopes, sends back the other one without opening it, together with the value of an *id* stored already in W. The latter is in sign of requesting back the envelope with that *id*. The receiver also stores the ids of the envelopes seen this time round. (I.e., R opens, e.g., E_2, sends back E_4 and the id, e.g., 1, as $1 \in W$. Thus, R would request back envelope E_1.)

Given the steps of the protocol so far, note that the envelope opened last together with the one requested last form an initial pair, which will now be found at R's end. Also, once the sender has sent this lastly requested envelope, the sender will be left with the other of the initial pairs at his end.

The sender S verifies that the recently returned envelope has the seal unbroken. If this is not so, he halts. Otherwise, he sends the one requested envelope to R. The non-equivocating sender sends the value $d{=}b \oplus b_l$, where b is the bit he is committing to and b_l is the bit hidden inside each envelope in the pair to be found at his side. (I.e., If the seal is unbroken, then S sends the requested E_1, $d{=}b \oplus b_l$, where b_l is in E_3 and/or in E_4). The equivocating sender just sends a bit-value d.

Finally, the receiver R opens the last envelope received and checks if the values at its side are equal. If not, he aborts. (I.e., If E_1 and E_2 do not contain the same value, then R aborts).

Let A denote the pair of envelopes to be found at this stage on the sender side.

The Phase of Opening and/or EquivocatoryOpening.

The never-cheating sender S sends one envelope E_k in the remaining pair A, i.e., $E_k \in A$ or $k \in \{i, j\}$. In turn, an equivocatorial open goes as follows: the sender S sends from the remaining pair A the envelope E_k that contains the bit $d \oplus c$, where c is the bit that the sender wants to open to.

Then, the receiver R checks that E_k is in the set A (by checking the ids). If this is the case, then R opens the envelope E_k and he assigns $d \oplus b_k$ to the commitment-bit b', where b_k is the value that R finds inside E_k. Otherwise, the receiver halts.

Explanations on the `Pass&MayCheat` protocol.

Consider that the sender S would not like to equivocate, i.e., his envelopes contains the set $\{x, x\}$ of values per pair. Firstly, consider that his value d is calculated as if no equivocation is to take place. In such circumstances, at the opening phase, this sender is obliged to open correctly, i.e., to the initially committed bit. Secondly, consider now that S does not form d as it is required, but that the receiver R were to follow the protocol. In this case, S may not be able to open.

Assume now that the sender S may want to equivocate, i.e., S's pairs of envelopes contain the set $\{x, x\}$ and $\{0, 1\}$, respectively. In this case, it is down to the choices of R whether this sender is able to continue the protocol; obviously, S may be caught and the protocol halts in half of the cases (the possibility that a randomly chosen bit x is equal either to 0 or to 1, depending which was the opening of R). In the other cases, S will clearly be able to open the value d to any bit-value, since the pair A of envelopes left at his end will contain x and $\overline{x}$.

The fact that receiver-forgeability is noticeable and the fact that the protocol is designed in a staggered way mean that R can neither open envelopes that he is not supposed to open nor open too many envelopes (i.e., if he wanted to break the hiding property).

The seal-once character of the envelopes prevents the sender S from changing a value x once it is stored inside the envelopes. I.e., Say that S reaches step 4 of the commitment phase and realizes that he will be caught by R, then S cannot act and change such a value x to prevent being caught.

Theorem 3.1 *In a hybrid UC-model, where the setup is the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality, the `Pass&MayCheat` protocol UC-realizes the $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ functionality.*

Proof: We technically need to prove that any attack that happens in the real world can be simulated in the ideal world where the $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ is running. We divide this in two (logical) parts: $\mathcal{A}$ corrupts the sender (Alice) and $\mathcal{A}$ corrupts the receiver (Bob). Other cases are trivial.

$\mathcal{A}$ corrupts the sender (Alice).

The commitment phase. $\mathcal{I}$ simulates $\mathcal{A}(Alice)$, its interaction with $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ and the protocol on Bob's side. We distinguish three cases.

I. $\mathcal{A}$ creates two pairs of envelopes, each containing the values $\{x, x\}$, for some $x \in \{0, 1\}$.

   $\mathcal{I}$'s simulation of Bob will receive envelopes and send them back as per the protocol.

   If $\mathcal{A}$ checks that the envelopes returned by Bob are indeed sealed, then $\mathcal{I}$ simulates a (**Verified**, id, ok) reply sent by $\mathcal{F}^{DE}_{\texttt{OneSeal}}$.

   $\mathcal{I}$ continues any simulation until $\mathcal{A}(Alice)$ sends a bit d to Bob.

   $\mathcal{I}$ chooses a bit b'' such that $b''{=}d \oplus x$, where x is the value inside the envelopes left on the side of the $\mathcal{A}(Alice)$. The ideal adversary $\mathcal{I}$ sends **Commit** b'' to the $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality.

II. $\mathcal{A}$ creates two pairs of envelopes, one containing the values $\{x, x\}$ and the values $\{0, 1\}$, for some $x \in \{0, 1\}$.

   The ideal adversary $\mathcal{I}$ sends **EquivocatoryCommit** to the $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality. If the functionality answers $\perp$, then the ideal adversary $\mathcal{I}$ simulates Bob opening such that he is eventually seeing the $\{0, 1\}$-pair, and he is then halting to $\mathcal{A}$. Otherwise, he simulates Bob sending, in stage, the pair containing $\{0, 1\}$ back to $\mathcal{A}$.

   If $\mathcal{A}$ checks that the envelopes returned by Bob are indeed sealed, then $\mathcal{I}$ simulates a (**Verified**, id, ok) reply sent by $\mathcal{F}^{DE}_{\texttt{OneSeal}}$.

   $\mathcal{I}$ continues any simulation until $\mathcal{A}(Alice)$ sends a bit d to Bob.

III. $\mathcal{A}$ creates two pairs of envelopes, each containing the values $\{0, 1\}$. In this case, $\mathcal{I}$ sends **AbortCommit** to the functionality and simulates Bob halting in front of $\mathcal{A}$.

The opening phase.
$\mathcal{I}$ awaits for Bob to be sent an envelope E_k from $\mathcal{A}$. The simulation now depends on what $\mathcal{A}$ did at the commitment phase (i.e., recall that $\mathcal{I}$ distinguished two cases based on the values sealed by $\mathcal{A}$, which he knew).

If it was case I above, then $\mathcal{I}$ sends **Open** to the $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ functionality.

If it was case II above, then $\mathcal{I}$ sends **EquivocatoryOpen** c to the $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ functionality, where c is calculated as $d \oplus b_k$ with b_k the value hidden inside envelope E_k.

Lemma 3.1 *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

The proof of the above lemma follows from the detailed simulation above.

$\mathcal{A}$ corrupts the receiver (Bob).

In this case, $\mathcal{I}$ simulates the real-world interaction of *Alice* with $\mathcal{A}(Bob)$ and with $F^{DE}_{OneSeal}$ and $\mathcal{I}$ corrupts dummy-Bob in the ideal-world.

Note that, in all this, $\mathcal{I}$ does not need to "commit" to the contents of the simulated envelopes but at the time of the opening.

We begin with the simulation of the commitment phase. The ideal adversary $\mathcal{I}$ waits for **Committed** or $\perp$ to be sent by the functionality $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$. (No matter what command **EquivocatoryCommit** or **Commit** was sent to $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ by dummy-Alice, $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ will send just **Committed** or $\perp$).

If $\perp$ was sent by $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$, then $\mathcal{I}$ creates four simulated envelopes, each pair containing $\{0, 1\}$. No matter what opening $\mathcal{A}(Bob)$ makes, $\mathcal{I}$ will simulate an abort from the commitment phase (i.e., send **AbortCommit** to $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$).

Now consider the case that **Committed** was sent by $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$. Then, $\mathcal{I}$ prepares four simulated envelopes for $\mathcal{A}(Bob)$, the content of which is not determined at this stage (as we anticipated above): if $\mathcal{A}(Bob)$ opens two envelopes from one initially formed pair, then $\mathcal{I}$ gives results (simulating $F^{DE}_{OneSeal}$) consistent with *Alice* not trying to equivocate or passing the equivocation.

Equivalently, for the first envelope to open from an arbitrarily fixed pair, $\mathcal{I}$ makes it open to a random value; for the second envelope from such the same pair, $\mathcal{I}$ makes it looks as if it had the same value. Namely, when $\mathcal{A}(Bob)$ opens two envelopes, their content is set to the same random bit and the two remaining envelopes are set to two different random bits. Again, this simulates a successful equivocatorial commitment: i.e., the ideal adversary $\mathcal{I}$ needs to choose a permutation $\pi \in S_4$ such that $\pi{=}(x, x, 0, 1)$, to place in the simulated envelopes in a delayed way. (As expected, $\mathcal{I}$ will eventually see the value of the bit committed by dummy-Alice and, with this strategy, $\mathcal{I}$ will be able to equivocatorially open to that bit.)

If $\mathcal{A}(Bob)$ opened an envelope that he should not have opened (i.e., both in one packet of two sent in step 2), then $\mathcal{I}$ sends $Halt$ to the $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality. If $\mathcal{I}$ got to this point, then he sends $d \in \{0, 1\}$ to $\mathcal{A}(Bob)$.

We continue with the simulation of the opening phase. As anticipated above, the ideal adversary $\mathcal{I}$ waits for (**Opened**, b) to be sent by the $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality.

It is now that $\mathcal{I}$ needs to send $\mathcal{A}(Bob)$ an envelope E_k containing a bit b_k such that E_k is consistent with the alleged permutation $\pi \in S_4$ that $\mathcal{I}$ used in the commitment phase (b_k appears in the permutation), E_k is consistent (w.r.t. ids) with $\mathcal{A}(Bob)$'s opening, and $b_k = d \oplus b$. Note that these constraints can always be satisfied giving the simulation in the commitment phase by $\mathcal{I}$. So, $\mathcal{I}$ sends this envelope E_k to $\mathcal{A}(Bob)$, which will "accept" the opening.

From the simulation above, together with the lemma within, it follows that the PMC protocol is UC-secure, realizing the $\frac{1}{2}$-weak bit-commitment functionality $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$. $\square$

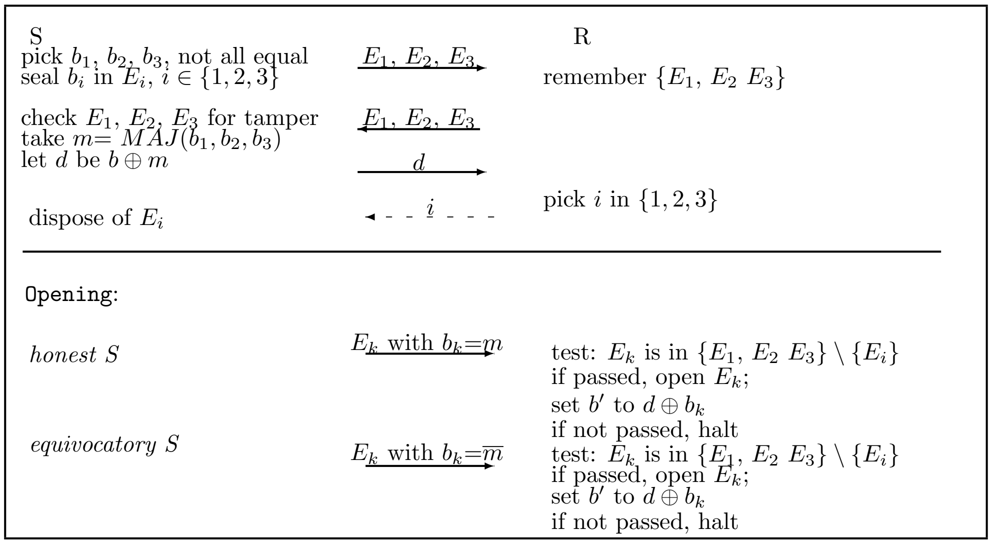
3.2 The `CommitEnablesCheat` and `OpenEnablesCheat` Protocols

In Fig.2 we present the `CommitEnablesCheat` protocol and in Fig.3 we give the `OpenEnablesCheat` protocol. The detailed explanations on these follow hereafter.

The `CommitEnablesCheat` Protocol (see Fig. 2): The Commitment Phase. The sender wants to commit to a bit b and proceeds as it follows.

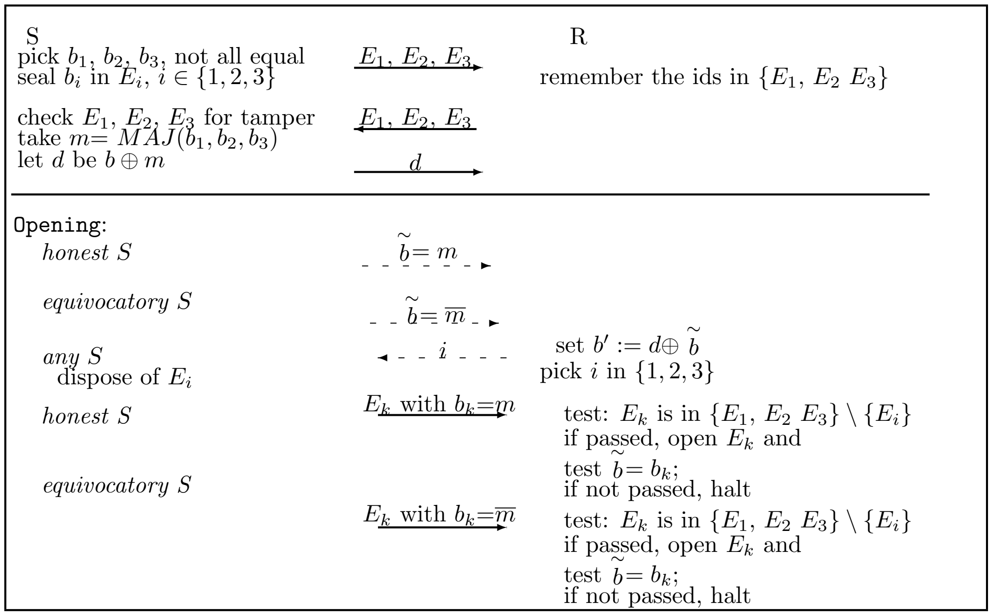
First, the sender S creates 3 sealed envelopes denoted E_1, E_2, E_3 respectively containing the bits denoted b_1, b_2, b_3, such that not all bits are equal. The sender sends the envelopes over to the receiver R.

Then, the receiver memorises the ids of the envelopes in the set $\{E_1, E_2, E_3\}$ and sends them back to the sender.



S		R
pick b_1, b_2, b_3, not all equal seal b_i in E_i, $i \in \{1,2,3\}$	$\xrightarrow{E_1,\ E_2,\ E_3}$	remember $\{E_1,\ E_2\ E_3\}$
check E_1, E_2, E_3 for tamper take m= $MAJ(b_1, b_2, b_3)$ let d be $b \oplus m$	$\xleftarrow{E_1,\ E_2,\ E_3}$ $\xrightarrow{d}$	
dispose of E_i	$\xleftarrow{i}$	pick i in $\{1,2,3\}$
`Opening`:		
honest S	$\xrightarrow{E_k \text{ with } b_k{=}m}$	test: E_k is in $\{E_1,\ E_2\ E_3\} \setminus \{E_i\}$ if passed, open E_k; set b' to $d \oplus b_k$ if not passed, halt
equivocatory S	$\xrightarrow{E_k \text{ with } b_k{=}\overline{m}}$	test: E_k is in $\{E_1,\ E_2\ E_3\} \setminus \{E_i\}$ if passed, open E_k; set b' to $d \oplus b_k$ if not passed, halt

Fig.2: The `CommitEnablesCheat` Protocol



S		R
pick b_1, b_2, b_3, not all equal seal b_i in E_i, $i \in \{1,2,3\}$	$\xrightarrow{E_1,\ E_2,\ E_3}$	remember the ids in $\{E_1,\ E_2\ E_3\}$
check E_1, E_2, E_3 for tamper take m= $MAJ(b_1, b_2, b_3)$ let d be $b \oplus m$	$\xleftarrow{E_1,\ E_2,\ E_3}$ $\xrightarrow{d}$	
`Opening`:		
honest S	$\xrightarrow{\widetilde{b}= m}$	
equivocatory S	$\xrightarrow{\widetilde{b}= \overline{m}}$	
any S dispose of E_i	$\xleftarrow{i}$	set $b' := d \oplus \widetilde{b}$ pick i in $\{1,2,3\}$
honest S	$\xrightarrow{E_k \text{ with } b_k{=}m}$	test: E_k is in $\{E_1,\ E_2\ E_3\} \setminus \{E_i\}$ if passed, open E_k and test $\widetilde{b}= b_k$; if not passed, halt
equivocatory S	$\xrightarrow{E_k \text{ with } b_k{=}\overline{m}}$	test: E_k is in $\{E_1,\ E_2\ E_3\} \setminus \{E_i\}$ if passed, open E_k and test $\widetilde{b}= b_k$; if not passed, halt

Fig.3: The `OpenEnablesCheat` Protocol

The sender now verifies that the envelopes sent back are untampered with. Then, he computes m as the majority of the bits sealed inside, i.e., $m{=}MAJ(b_1, b_2, b_3)$. The sender wants to commit to a bit b. He calculates $d{=}b \oplus m$. Then, the sender sends d to the receiver.

At this step, the receiver sends the identifier i of an envelope that the sender should dispose of, i.e., $i \in \{1, 2, 3\}$. (This means that the content of envelope i will not count further in the protocol.) Let the set $A{=}\{E_1, E_2, E_3\} \setminus \{E_i\}$ denote the set of remaining envelopes.

Finally, the sender disposes of envelope i. (Note that after this the sender can equivocate if the remaining envelopes contain different bits.)

The *equiv* value is $\frac{2}{3}$.

The Opening Phase.

The non-equivocating sender sends an envelope E_k such that $b_k{=}m$.

The equivocating sender sends an envelope E_k such that $b_k{=}\overline{m}$.

Then, the receiver tests that $E_k \in A$ and if so, he sets b', the commitment bit, as follows: $b'{=}d \oplus b_k$. If the test fails, the receiver halts.

Note that by being asked to discard[4] an envelope at the opening phase instead of in step 4 of the commitment phase, the idea behind protocol `CommitEnablesCheat` can be shaped to obtain a protocol where the equivocation becomes clear only at the opening time. The protocol obtained in this way is hereby denoted `OpenEnablesCheat`. The protocols `CommitEnablesCheat` and `OpenEnablesCheat` are graphically represented in Fig. 2 and Fig. 3, respectively.

The `OpenEnablesCheat` protocol is very similar to the other protocol, `CommitEnablesCheat`, as we said. We will hereby only state the main differences between the two. In the `OpenEnablesCheat` protocol, the discarding of the envelope is made at the opening phase. In a sense, the sender first "announces" how he will open his commitment by sending $\tilde{b}$ (i.e., if it is an equivocal opening the sender will send inside $\tilde{b}$ the negation of m, otherwise he sends m). Then, the receiver finally asks for the discarding of the one the envelopes that is at the end of the sender. After this discarding, the sender has to send E_k, i.e., the envelope with the k identifier. Depending which discarding was called for, the sender may or may not be able to send over one envelope E_k that contains the same b_k as it was announced via $\tilde{b}$. Clearly, these steps are emulating the $\mathcal{F}_{LearnAtOpening}$ functionality.

Once again, observe that—unlike in the `Pass&MayCheat` protocol— the committer of the `CommitEnablesCheat` and `OpenEnablesCheat` protocols

[4] A possible way of implementing discarding is sending the emptied envelope back to the receiver.

can cheat with some probability (i.e., $\frac{2}{3}$), and this possibility is down to a mere choice of the receiver and it is not influenced by receiver being caught cheating.

These requirements sound similar to looking for a means in which Alice would commit to a bit b using a BSC (binary symmetric channel) with noise level q [6]. However, all previous constructions [6, 12, 24] addressing this are not sender-strong, but receiver-strong. Moreover, they are designed within traditional lines, i.e., are not UC-secure. On a different note, those classical solutions are based error-correction codes (ECC) and/or pseudo-random generators (PRNG). In our case, PRNGs and ECCs are out of scope: we intend cryptographically lightweight primitives. And, `CommitEnablesCheat` and `OpenEnablesCheat` above deliver sender-strong, *UC-secure*, simple and human operable primitives.

Explanations on the `CommitEnablesCheat` and `OpenEnablesCheat` protocols.

We detail only on the `CommitEnablesCheat` protocol above, counting on that the explanations on `OpenEnablesCheat` would very similar. We will now show that the protocol `CommitEnablesCheat` is complete. Assume that the parties adhere to the rules the protocol. In step 3, we would surely have $m{=}x$, if S had followed step 2 and S had produced the envelopes correctly (i.e., a permutation of $\{x, x, \overline{x}\}$, $x \in_U \{0,1\}$ is sealed inside them). In this case, in the remaining set A there would always be an envelope E_k containing the value x that opens the commitment correctly. This is irrespective of the value b_i (be it x or $\overline{x}$). There is a probability of $\frac{2}{3}$ for S to be able to open his commitment to the flipped bit (i.e., point 2 in the opening phase). Namely, this is in the cases where an envelope with value $\overline{x}$ is present in the remaining set A. Also, it is clear that S gains no benefit from not playing by the rules. Theorem 3.2 present these more formally, within the UC setting.

Theorem 3.2 *In a hybrid UC-model, where the setup is the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality, the `CommitEnablesCheat` and `OpenEnablesCheat` protocols UC-realize the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ and the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionalities, respectively.*

Due to heavy similarities between the case of `CommitEnablesCheat` and the case of `OpenEnablesCheat`, the proof of Theorem 3.2 is given in the for `CommitEnablesCheat` only.

Proof: We technically need to prove that any attack that happens in the real world can be simulated in the ideal world where the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtCommit}}$ is running. We divide this in two (logical) parts: $\mathcal{A}$ corrupts the sender (Alice) and $\mathcal{A}$ corrupts the receiver (Bob). The same sort of respective simulations by $\mathcal{I}$ as in the previous proof are in place.

$\mathcal{A}$ corrupts the sender (Alice). Hence, it is $\mathcal{A}$ who creates and sends the 3 envelopes, i.e., interacts with the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality. Note that $\mathcal{I}$ intercepts the communication between $\mathcal{A}$ and the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality. So, $\mathcal{I}$ knows when $\mathcal{A}$ is cheating.

The commitment phase.

I $\mathcal{A}$ has sent a valid pack of envelopes and the contents of the envelopes are an arbitrary but fixed permutation of $(x, x, \overline{x})$, where $x \in_U \{0,1\}$ (i.e., $b_1{=}x$, $b_2{=}x$ and $b_3{=}\overline{x}$, up to a permutation).

Bob will receive the envelopes and send them back.

If $\mathcal{A}$ checks that the envelopes returned by Bob are indeed sealed, then $\mathcal{I}$ simulates a $(\textbf{Verified}, id, ok)$ reply sent by $\mathcal{F}^{DE}_{\texttt{OneSeal}}$.

$\mathcal{I}$ continues any simulation until $\mathcal{A}(Alice)$ sends a bit d to Bob.

$\mathcal{I}$ picks $m^I{=}MAJ(b_1, b_2, b_3)$ (i.e., on this input, $m^I{=}x$). $\mathcal{I}$ chooses a bit b'' such that $b''{=}d \oplus m^I$. The ideal adversary $\mathcal{I}$ sends **Commit** b'' to the $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$ functionality. The functionality replies with a value for *equiv*, which is "yes" with probability $\frac{2}{3}$ and "no" with probability $\frac{1}{3}$. If *equiv* is "yes", $\mathcal{I}$ picks $i \in \{1,2,3\}$ such that $b_i{=}x$ and otherwise he picks i such that $b_i{=}\overline{x}$. $\mathcal{I}$ simulates Bob in sending i to $\mathcal{A}$.

II $\mathcal{A}$ has sent an invalid pack of envelopes, with all value inside equal to x, where $x \in_U \{0,1\}$.

$\mathcal{I}$ acts as in the case I above, but he sets *equiv* always to "no".

The opening phase.
$\mathcal{I}$ awaits for Bob to be sent an envelope E_k from $\mathcal{A}$ to see how $\mathcal{A}$ wants to open. The simulation now depends on what $\mathcal{A}$ did at the commitment phase (i.e., recall that $\mathcal{I}$ distinguished two cases based on the envelopes sealed by $\mathcal{A}$, which he knew).

If it was case I of the commitment phase and $b_k{=}m^I$, then $\mathcal{I}$ will send an **Open** command to the $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$.

If it was case I of the commitment phase and $b_k{=}\overline{m^I}$, then $\mathcal{I}$ will send an **EquivocatoryOpen** command to the $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$. (Note that because of the simulated i in the last step of the commitment, the ideal adversary is able to open the bit b'' in the same way that the adversary would open his d.)

If it was case II of the commitment phase, then $b_k{=}m^I$ and $\mathcal{I}$ will send an **Open** command to the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommit}}$.

Lemma 3.2 *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

The proof of the above lemma follows from the detailed simulation above.

$\mathcal{A}$ corrupts the receiver (Bob)

In this case, $\mathcal{I}$ will have to create and send simulated envelopes for $\mathcal{A}(Bob)$. Note that the ideal adversary does not need to commit to the contents of containers from the beginning, since they influence the view of the environment only when they are actually open.

The commitment phase.

$\mathcal{I}$ sends 3 simulated envelopes to $\mathcal{A}$.

$\mathcal{I}$ continues the simulation until $\mathcal{A}$ sends the envelopes back. If $\mathcal{A}$ does not send the envelopes back or they are tampered with, $\mathcal{I}$ assigns $\{x, x, \overline{x}\}$ values to the envelopes at random and continues the simulation in the opening. $\mathcal{A}$ will be stuck in the protocol, eventually.

The simulation through $\mathcal{I}$ continues until it receives **Committed** from $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommit}}$.

$\mathcal{I}$ chooses d at random and sends this value d to $\mathcal{A}(Bob)$.

Consider that, at the end of this phase, $\mathcal{I}$ will identify the simulated, remaining envelopes by the set $\{1, 2, 3\} \setminus \{i\}$, where i is picked at random. (There is no better strategy for $\mathcal{A}$ to pick i without opening envelopes.).

The opening phase.

$\mathcal{I}$ waits until it receives (**Opened**, b') from $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommit}}$.

Let $b^{\mathcal{I}}_k$ be $d \oplus b'$.

The ideal adversary $\mathcal{I}$ will send an envelope $k \in \{1, 2, 3\} \setminus \{i\}$ simulated and containing $b^{\mathcal{I}}_k$.

Lemma 3.3 *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the receiver, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

The proof of the above lemma follows from the detailed simulation above.

From the simulation above, together with the two lemmas within, it follows that the `CommitEnablesCheat` protocol is UC-secure, realizing the $\frac{2}{3}$-weak bit-commitment functionality $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommit}}$. □

3.3 Amplifying q-WBC Sender-Strong Protocols

Let ✠ $\in \{$`Pass&MayCheat`, `CommitEnablesCheat`, `OpenEnablesCheat`$\}$. Let $\star \in \{$`EscapeThenMayCheat`, `LearnAtCommitment`, `LearnAtOpening`$\}$.

By using k instances of a q-weak sender-strong protocol of the ✠-kind of protocols, we can obtain a protocol `Amplified_`✠ protocol that UC-realizes $\mathcal{F}^{q^k-WBC}_{\star}$. Hence, for a conveniently large k, we can attain regular bit-commitments. See the formalisations below.

The `Amplified_Pass&MayCheat` Protocol:

(Equivocatory) Commitment Phase.

The sender commits, all equivocally or all normally, to a bit b_j in k sequential rounds, each time using the $\mathcal{F}^{q-WBC}_{\texttt{EscapeThenMayCheat}}$ functionality, $j \in \{1, \ldots, k\}$. The j-th such functionality is denoted $\mathcal{F}^{q-WBC}_{\texttt{EscapeThenMayCheat};\ j}$.

Each functionality $\mathcal{F}^{q-WBC}_{\texttt{EscapeThenMayCheat};\ j}$ to which **EquivocatoryCommit** was sent, outputs to its sender `Committed`, with probability q and $\perp$ otherwise. If $\perp$ is sent, then the receiver aborts.

(Equivocatory) Opening Phase.

The sender opens all commitments, equivocally or not, using the $\mathcal{F}^{q-WBC}_{\texttt{EscapeThenMayCheat};\ j}$ functionalities. The receiver halts if the openings are not all the same.

Theorem 3.3 *Let $q \in (0,1)$ and λ be a security parameter. By using $k{=}\Omega(\lambda)$ instances of an $\mathcal{F}^{q-WBC}_{\star}$ functionality, we can construct a protocol* ***`Amplified_`✠*** *that UC-realizes the $\mathcal{F}^{BC}$ functionality, where we have $\star \in \{$`EscapeThenMayCheat`, `LearnAtCommitment`, `LearnAtOpening`$\}$ and ✠ $\in \{$`Pass&MayCheat`, `CommitEnablesCheat`, `OpenEnablesCheat`$\}$.*

In particular, the protocol `Amplified_Pass&MayCheat` UC-realizes the $\mathcal{F}^{q^k-WBC}_{\mathit{EscapeThenMayCheat}}$ functionality.

For the regular BC functionality, $\mathcal{F}^{BC}$, section C of the Appendix. The `Amplified_Pass&MayCheat` BC protocol is trivially following out of `Amplified_Pass&MayCheat`, i.e., where equivocation is not possible. By letting $k{=}\frac{\log\varepsilon}{\log q}$ in Theorem 3.3, we make `Amplified_Pass&MayCheat` a ε-weak bit-commitment, with ε arbitrarily close to 0. However, for protocol `Amplified_Pass&MayCheat` to UC-realize $\mathcal{F}^{BC}$, we need a k to be of linear-size in the security parameter λ. Proofs that weak bit-commitment protocols in the above sense can be amplified to regular bit-commitments exist already, e.g., [24]. The proofs therein follow long-established lines, i.e., not the UC framework [5]. Also, they often refer to receiver-strong protocols and generally use more convoluted primitives, e.g., pseudo-random generators, error-correcting codes, outside our lightweight interests. Our proof is done in the UC framework and, as we can see, the protocol respects the sender-strong aspects sought-after herein. For simplicity, the proof is split between the three different cases: for the case of `Amplified_Pass&MayCheat`, see first lemma below; for the case of `Amplified_CommitEnablesCheat`, see the second lemma below; the `Amplified_OpenEnablesCheat` protocol is similar to `Amplified_CommitEnablesCheat` .

Lemma 3.4 *The protocol* `Amplified_Pass&MayCheat` *UC-realizes the* $\mathcal{F}^{q^k-WBC}_{\mathtt{EscapeThenMayCheat}}$ *functionality.*

$\mathcal{A}$ corrupts the receiver. Note that the receiver cannot cheat using the functionalities provided. Hence, there is no attack in the real world to be simulated in the ideal world.

$\mathcal{A}$ corrupts the sender.

Commitment Phase.

Let the bits used by $\mathcal{A}$ be denoted $b_1, \ldots, b_k$. Moreover, let $I \subseteq \{1, \ldots, k\}$ denote the indices of those bits sent through the command **Commit** and $J \subseteq \{1, \ldots, k\}$ denote the indices of those bits sent through the command **EquivocatoryCommit** to different instances $\mathcal{F}^{q-WBC}_{\mathtt{EscapeThenMayCheat};\,\ell}$, $\ell \in \{1, \ldots, k\}$. Also, let $equiv_j$ be the answer that $\mathcal{I}$ simulates for $\mathcal{A}$ to receive from each functionality $\mathcal{F}^{q-WBC}_{0;\,j}$, $j \in J$ (recall that $equiv_j{=}$`Committed` with probability q and $equiv_j{=}\bot$, otherwise).

[5]Similar proofs of amplifications may exist in the UC framework, however they would not be with respect to the $\mathcal{F}^{q-WBC}_{i}$ functionalities as introduced in Section 2.

The ideal adversary $\mathcal{I}$, upon seeing these commands, replies nothing to the ones of the **Commit** type and replies $equiv_j$=`Committed` to the commands of type **EquivocatoryCommit**, for all $j \in J$.

In the case that there is some $j \in J$ such that $equiv_j = \perp$, then $\mathcal{I}$ sends `AbortCommit` to $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$.

There are two cases completely describing the corrupt adversary's behaviour:

I. $card(I) \neq 0$, i.e., $\mathcal{A}$ has sent some **Commit** commands[6];

II. $card(I)$=0, i.e., $\mathcal{A}$ has only sent **EquivocatoryCommit** commands.

Case I above is completely described by the following sub-cases, depending on whether $\mathcal{A}$ could ever possibly open his commitments to the same bit:

I.1. In this case, there are two bits b_i and $b_{i'}$ of different values both sent through **Commit** commands, i.e., the set I of indices is non-empty and $\exists i, i' \in I, i \neq i'$ such that $b_i \neq b_{i'}$. $\mathcal{I}$ sends **Commit**(0) to $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$ and stores that this was a special case.

I.2. In this case, all bits indexed in the set I have the same value. Let us denote this value b_i, $i \in I$. However, the ideal adversary $\mathcal{I}$ knows that $equiv_j$=`Committed` for $\mathcal{A}$, for all $j \in J$. $\mathcal{I}$ sends **Commit**(b_i) to the ideal functionality $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$.

In case II above, the ideal adversary $\mathcal{I}$ sends **EquivocatoryCommit** to the $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$ functionality. $\mathcal{I}$ gets an *equiv* answer back from the $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$ functionality. If the *equiv* reply is negative, then $\mathcal{I}$ halts (advising the real-world receiver to halt by an abort message as from an $\mathcal{F}^{q-WBC}_{0;\,j}$ functionality, $j \in J$).

Opening Phase.

The opening phase follows on from the distinct cases discussed in the commitment phase.

If it was case I.1 and $\mathcal{I}$ has not halted in the commitment phase, then $\mathcal{I}$ halts now. Note that this models the real-world scenario, as —in this case– $\mathcal{A}$ will never be able to open to the same bit as he has sent two different, un-flippable bits, i.e., sent under **Commit** commands. I.e., the real-world receiver will halt also at most at the end of the k openings.

[6]The notation $card(S)$ denotes the cardinality of a set S.

If it was case I.2 and $\mathcal{I}$ has not halted in the commitment phase, then $\mathcal{I}$ sends **Open** to the $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$ functionality. Note that in this case, the opening will have the same b_i value as in the real world.

If it was case II and $\mathcal{I}$ has got a positive *equiv* back, then upon the opening c of $\mathcal{A}$, $\mathcal{I}$ sends **EquivocatoryOpen (c)** to the $\mathcal{F}^{q^k-WBC}_{\texttt{EscapeThenMayCheat}}$ functionality.

Lemma 3.5 *In the case above, the following holds. For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

The proof of the above lemma follows immediately from the detailed simulation above. □

Lemma 3.6 *In a hybrid UC-model, where a linear number of k instances of the $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment}}$ are available as setup, the $\mathcal{F}^{BC}$ can be UC-realized, where $q \in (0,1)$.*

Proof: Note that the receiver cannot cheat using the functionalities provided. Hence, there is no attack in the real world to be simulated in the ideal world.

$\mathcal{A}$ corrupts the sender.

Commitment Phase.

Let the bits used by $\mathcal{A}$ be denoted $b_1, \ldots, b_k$ in respective (**Commit**, b_l) commands sent to the instances $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment};\ \ell}$, $\ell \in \{1, \ldots, k\}$.

$\mathcal{I}$ flips coins such that for each $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtCommitment};\ l}$, it simulates an $equiv_\ell$ reply being "yes" with probability q and "no" otherwise, for $\ell \in \{1, \ldots, k\}$.

Then, having the bits b_l and the values $equiv_l$, the ideal adversary $\mathcal{I}$ looks at the cases that $\mathcal{A}$ can be in:

I. $\mathcal{A}$ could only open to the bit 0 ($b_\ell = 0$ whenever $equiv_\ell$="no" and there may be other $equiv_{\ell'}$ equal to "no");

II. $\mathcal{A}$ could only open to the bit 1 ($b_\ell = 1$ whenever $equiv_\ell$="no" and there may be other $equiv_{\ell'}$ equal to "no");

III. $\mathcal{A}$ could only open to any (this is the case if all $equiv_\ell$="yes");

IV. $\mathcal{A}$ cannot open to a consistent bit (this is the case if some $equiv_\ell$="no" with $b_\ell = 0$ and $equiv_{\ell'}$="no" with $b_{\ell'} = 1$).

Note that for k as in the current lemma, the challenge protocol for the UC-environment is a game that is indistinguishable for the game where case III never arises. Using the difference lemma [25], we conclude that we can ignore the simulation by $\mathcal{I}$ in this case.

In case I and II above, $\mathcal{I}$ sends (**Commit**, b) to the $\mathcal{F}^{BC}$ functionality, b=0 in the first case and and b=1 in the second case.

In case IV above, $\mathcal{I}$ sends (**Commit**, b) to the $\mathcal{F}^{BC}$ functionality, where b is a bit picked at random.

Opening Phase.

If it was case I or case II of the commitment phase, then $\mathcal{I}$ simply sends **Open** to the $\mathcal{F}^{BC}$ functionality (and this will reflect exactly the bit opened in the real-world).

If it was case IV of the commitment phase, then $\mathcal{I}$ sends a halting message to the receiver (that presumably the protocol to realize contains).

Lemma 3.7 *In the case above, the following holds. For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

The proof of the above lemma follows immediately from the detailed simulation above. □

3.4 EUC-Insecurity of the `CommitEnablesCheat` Protocol in the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$-hybrid model

Lemma 3.8 *In a hybrid EUC-model, where the setup is the $\mathcal{F}^{DE}_{\mathit{OneSeal}}$ functionality, the protocol `CommitEnablesCheat` does not EUC-realizes the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathit{LearnAtCommitment}}$ functionality.*

Proof: We will show that for a certain environment $\mathcal{Z}$ and a certain adversary $\mathcal{A}$, there is no ideal adversary $\mathcal{I}$ that perfectly simulates the protocol run by $\mathcal{A}$ to the environment $\mathcal{Z}$.

In an EUC $\mathcal{F}^{DE}_{\texttt{OneSeal}}$-hybrid model where the adversary corrupts the sender in the EUC real world, assume the following commitment phase execution.

The environment $\mathcal{Z}$ runs the **Commit** b protocol with b selected at random and $\mathcal{A}$ just relays messages and envelopes between $\mathcal{Z}$ and the receiver R. After the environment $\mathcal{Z}$ learns that R has actually received the **Committed** message, $\mathcal{Z}$ runs the **Open** protocol and compares the bit at R's end to the actual chosen bit b. Clearly, $\mathcal{I}$ cannot guess the bit b and cannot simulate perfectly the opening to b (i.e., decommit to b with probability 1). $\square$

A version of Lemma 3.8 can be given for the rest of our WBC protocols and those in [22], i.e., the DE functionality as it is brings EUC-insecurity to the other WBCs too.

3.5 (Stronger) Universally Composable Security

In obtaining the UC-simulatability proof, details as small as the order of the messages in the commitment-phase of the weak protocol `CommitEnablesCheat` above and/or the amount of randomness given to the sender have huge impacts. To give but an example, imagine a protocol like the `CommitEnablesCheat` protocol where step 3 of the commitment phase becomes step 4 and vice-versa. This minute change in the `CommitEnablesCheat` protocol makes it lose its UC-security (i.e., the UC-realizability of $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ is lost), while it keeps the protocol perfectly hiding and binding with probability $\frac{2}{3}$ in the classical sense.

So, while it may seem easy to manipulate DEs to get weak BCs, it is the case that sender-strength and UC-security together are not trivially attained. (It is indeed easier to construct q-weak bit-commitments using a formalisation of distinguishable envelopes, when no UC-security or asymmetrical strengths are required.)

Let us move to EUC-security. For a wrap-up on GUC (Generalised UC) or EUC (Externalised UC) see the Appendix, Section A.2. Imagine an EUC model with the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$-setup. Further, consider an environment that creates the envelopes and gives them to the adversary. Then, in the ideal world, the simulator cannot "extract" the bit b to commit to and thus he is bound to fail to indistinguishably simulate the commitment phase. So, on these grounds, none of the protocols presented so far, nor those constructed previously in [22] for the receiver-strong case are EUC-secure or GUC-secure.

To remedy the above (i.e., make the protocols herein and those in Moran and Noar's work [22] EUC-secure), we present a slightly modified $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$-setup.

$\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$: A Stronger Functionality for Tamper-Evident Distinguishable Sealed Envelopes

This functionality stores tuples of the form (id, $value$, $holder$, $state$). The values in one entry indexed with id, like before.

SealSend(id, $value$, P_j). Let this command be received from an envelope-creator party P_i. It seals an envelope and sends its id to the future holder P_j. If this is the first **Seal** message with id id, the functionality stores the tuple (id, $value$, P_j, **sealed**) in the table. The functionality sends (id, P_i) to P_j and to $\mathcal{I}$. (Optionally, it can send (id, $sealed$) to P_i and to $\mathcal{I}$). If this is not the first command of type **Seal** for envelope id, then the functionality halts.

Send(id, P_j). Let this command be received from a holder-party P_i. This command encodes the sending of an envelope held by P_i to a party P_j. Upon receiving this command from party P_i, the functionality verifies that there is an entry in its table which is indexed by id and has $holder_{id} = P_i$. If so, it outputs (**Receipt**, id, P_i, P_j) to P_j and $\mathcal{I}$ and replaces the entry in the table with (id, $value_{id}$, P_j, $state_{id}$).

Open id. Let this command be received from a holder-party P_i. This command encodes an envelope being opened by the party that currently holds it. Upon receiving this command, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it sends (**Opened**, id, $value_{id}$) to P_i and $\mathcal{I}$. It also replaces the entry in the table with (id, $value_{id}$, $holder_{id}$, **broken**).

Verify id. Let this command be received from a holder-party P_i. This command denotes P_i's verification of whether or not the seal on an envelope has been broken. The functionality verifies that an entry indexed by id appears in the table and that $holder_{id} = P_i$. It sends (**Verified**, id, $state_{id}$) to P_i and to $\mathcal{I}$.

The essential modification from $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ to the $\mathcal{F}^{\texttt{purpoted}DE}_{\texttt{OneSeal}}$ is that in the latter functionality an envelope is built for an intended holder. To this end, the purported destinator receives a notification of the form (id, $creator$), which indicates that his faced with a newly created envelope. If these envelopes were to be constructed by some real manufacturer, this company would ship its products to known/registered addresses. Thus, we believe that this modification is reasonable. A much stronger enhancement would have been to store or disclose the identities of the creators. This would be similar to signing the TE devices. We do not adhere to this behaviour for $\mathcal{F}^{\texttt{purpoted}DE}_{\texttt{OneSeal}}$.

With this amendment, at a high level, we deter relay attacks. A receiver will expect envelopes freshly created for him and parties will not be able to pass on, as creators, envelopes produced by other participants, i.e., $\mathcal{Z}$ will not be able to prepare envelopes for $\mathcal{A}$, to be used as if $\mathcal{A}$ was their creator. Thus, the issue raised by Lemma 3.8 is rectified.

Theorem 3.4 *In a hybrid EUC-model, where the setup is the $\mathcal{F}^{purpotedDE}_{OneSeal}$ functionality, the protocol* `CommitEnablesCheat` *EUC-realizes the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{LearnAtCommitment}$ functionality.*

Note: We have to note that the protocol in the theorem above is in fact the `CommitEnablesCheat` slightly changed to accommodate the use of purported DE instead of simple DE. I.e., it is where it is the setup functionality (e.g., a sort of creating-authority) which sends the created envelopes to the destinator, not the logical creator; and, further, the destinator must check the ID of the logical creator as announced by the functionality.

The proof of the above theorem follows from the proofs of Theorem 3.2 and that of Lemma 3.8, combined with the fact that $\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$-envelopes have a specified entity as their destination and this entity knows this fact upon the creation of the envelopes. We conjecture that Theorem 3.4 holds even in the case of adaptive adversaries.

A version of Theorem 3.4 can be given for the rest of our WBC protocols and those in [22], i.e., purported DE can bring EUC-security to the other WBCs too.

4 Implication-Relations between (Weak) Bit-Commitments and Distinguishable Envelopes in UC

From the work in [22] and the line herein, we can summarise that receiver-strong and sender-strong weak UC bit-commitment (amplifiable to BC) can be UC-realized with $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ (or $\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$). We already know that the ideal functionality of $\mathcal{F}^{BC}$ (see $\mathcal{F}_{COM}$ in [7]) is a sufficient UC-setup to realize ZK. The ultimate question in this context would be whether, e.g., $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ is sufficient for UC-secure ZK. But, we could first study the possible separations (within UC) between all these ideal functionalities, i.e., for BC, WBC, DE.

Firstly, we can UC-realize an $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$, $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ or an $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$ functionality, for some $q \in (0, 1)$ by using just a multiple commitment $\mathcal{F}_{MCOM}$ [7] UC-setup. (For the $\mathcal{F}_{MCOM}$ [7] functionality, please refer to Section C of the Appendix). In other words, to get a UC-secure sender-strong weak bit-commitment, we only need several instances of the regular bit-commitment functionality $\mathcal{F}^{BC}$. Namely, three $\mathcal{F}^{BC}$ functionalities (see Section C of the Appendix) are sufficient to UC-realize a $\frac{2}{3}$-WBC which is sender-strong. Imagine the protocol `CommitEnablesCheat` where three commitments using $\mathcal{F}_{MCOM}$ or using three instances of $\mathcal{F}^{BC}$ respectively and uniformly replace the three envelopes used inside. Let this transformed protocol be called $\mathcal{P}$. Obviously, $\mathcal{P}$ is a UC-secure SS-WBC build via $\mathcal{F}_{MCOM}$ or $\mathcal{F}^{BC}$. The same mechanism would work to transform `OpenEnablesCheat` instead of `CommitEnablesCheat` and we would UC-realize $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$, using a $\mathcal{F}_{MCOM}$ or $\mathcal{F}^{BC}$ setup.

Secondly, all sender-strong weak BCs UC-imply regular BCs (following from Theorem 3.3).

Thirdly, we conjecture that a bit-commitment setup is not enough to UC-realize the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ distinguishable tamper-evident envelope functionality. This is mainly due to the difference in opening commitments and opening "envelopes". The first are always open by their creator. If BC would UC-imply DE, then DE should also always be finally opened by their creator. Or, the latter could be possible only if the creator of the DEs would always know who the current holder of the DE is. This is not the case, hence our conjecture.

If we take into about the amplification proofs as well, then we have rendered the picture of the UC-realisability of different flavours of sender-strong weak BC with tamper-evident envelopes, have drawn of their relation with (almost) regular BC and with receiver-strong weak BCs by Moran and Naor [22]. Also, we now can see that all weak BCs are equivalent to regular BCs, at some level.

We leave the EUC or the GUC correspondents of the implications enumerated above as open questions. Also, it is in our future interests to investigate if flavours of tamper-evident devices can UC-construct ZK proofs of knowledge without passing through BC protocols.

5 Conclusions

We conclude that quite simple, distinguishable, sealed envelopes can create sender-strong (weak) bit-commitments protocols. This answers several practical needs [11, 18, 14], but also we can view it as answering a quicker variant of the open question in Moran and Naor's work [22].

We have also shown that the protocols in [22] are not EUC-secure but only UC-secure. We then showed how to modify the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality given in Moran and Naor's work [22] such that we also create (weak) bit-commitment protocols that are EUC-secure. We illustrated lightweight amplification proofs of our WBC protocols. The implications between UC weak BCs, UC regular BCs and distinguishable tamper-evident UC envelopes were finally discussed.

References

[1] D. Beaver. Adaptive Zero Knowledge and Computational Equivocation (Extended Abstract). In *The 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 629–638, 1996.

[2] Mihir Bellare, Marc Fischlin, Shafi Goldwasser, and Silvio Micali. Identification protocols secure against reset attacks. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 495–511, London, UK, UK, 2001. Springer-Verlag.

[3] Ioana Boureanu and Serge Vaudenay. Several weak bit-commitments using seal-once tamper-evident devices. In Tsuyoshi Takagi, Guilin Wang, Zhiguang Qin, Shaoquan Jiang, and Yong Yu, editors, *Provable Security - 6th International Conference - ProvSec*, volume 7496 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2012.

[4] Ioana Boureanu and Serge Vaudenay. Input-aware equivocable commitments and uc-secure commitments with atomic exchanges. In *The 7th International Conference on Provable Security (ProvSec)*, pages 121–138. Springer Berlin Heidelberg, Melaka, Malaysia, October 2013.

[5] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer Systems Science*, 37:156–189, October 1988.

[6] C. Crépeau. Efficient Cryptographic Protocols Based on Noisy Channels. In *Advances in Cryptology, Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques – EUROCRYPT*, Lecture Notes of Computer Science, pages 306–317, Berlin, Heidelberg, 1997. Springer-Verlag.

[7] R. Canetti. A Unified Framework for Analyzing Security of Protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(16), 2001.

[8] R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Proceedings of the 17th IEEE workshop on Computer*

Security Foundations, pages 219–239, Washington, DC, USA, 2004. IEEE Computer Society.

[9] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally Composable Security with Global Setup. Cryptology ePrint Archive, Report 2006/432, 2006. http://eprint.iacr.org/.

[10] N. Chandran, V. Goyal, and A. Sahai. New Constructions for UC Secure Computation Using Tamper-Proof Hardware. In *Advances in Cryptology, Proceedings of the 27th Annual International Conference on Theory and Application of Cryptographic Techniques – EUROCRYPT*, pages 545–562, 2008.

[11] C. Chin-Chen and C. Ya-Fen. Efficient Anonymous Auction Protocols with Freewheeling Bids. *Computers & Security*, 22(8):728–734, 2003.

[12] I. Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In *Advances in Cryptology, Proceedings of the 9th Annual International Cryptology Conference*, CRYPTO, pages 17–27, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[13] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 409–418, New York, NY, USA, 1998. ACM.

[14] G.Dane. The Implementation of an Auction Protocol over Anonymous Networks. http://research.microsoft.com/en-us/um/people/gdane/papers/partiiproj-anonauctions.pdf, 2000.

[15] R. Gennaro. Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In *CRYPTO*, pages 220–236, 2004.

[16] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding Cryptography on Tamper-Proof Hardware Tokens. In *Theory of Cryptography*, pages 308–326, 2010.

[17] J. Katz. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In *Theory and Application of Cryptographic Techniques*, pages 115–128, 2007.

[18] H. Kikuchi, M. Harkavy, and J. D. Tygar. Multi-round Anonymous Auction Protocols. In *In Proceedings of the 1st IEEE Workshop on*

Dependable and Real-Time E-Commerce Systems, pages 62–69. Springer-Verlag, 1998.

[19] P. Mateus and S. Vaudenay. On Tamper-Resistance from a Theoretical Viewpoint. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems(CHES)*, volume 5747 of *Lecture Notes in Computer Science*, pages 411–428. Springer, 2009.

[20] T. Moran and M. Naor. Basing Cryptographic Protocols on Tamper-Evident Seals. In L. Caires et al., editor, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 285–297. Springer-Verlag, Jul 2005.

[21] T. Moran and M. Naor. Polling with Physical Envelopes: A Rigorous Analysis of a Human-Centric Protocol. In S. Vaudenay, editor, *Advances in Cryptology, Proceedings of the 25th Annual International Conference on Theory and Application of Cryptographic Techniques – EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 88–108. Springer Berlin / Heidelberg, May 2006.

[22] T. Moran and M. Naor. Basing Cryptographic Protocols on Tamper-Evident Seals. *Theoretical Computer Science*, 411:1283–1310, March 2010.

[23] T. Moran and G. Segev. David and Goliath Commitments: UC Computation for Asymmetric Parties Using Tamper-Proof Hardware. In *Theory and Application of Cryptographic Techniques*, pages 527–544, 2008.

[24] M. Naor. Bit Commitment Using Pseudo-Randomness. *Journal of Cryptology*, 4:151–158, 1991.

[25] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. manuscript, 2006.

A Overview on the UC and Enhanced UC Frameworks

A.1 General Approach to UC proofs

At some high level, a UC proof that a protocol is secure means to show that the environment ($\mathcal{Z}$) cannot distinguish between the execution in the "real world" from the execution in the "ideal world" [7].

The "ideal world" contains "dummy" parties, the "target" ideal functionality (that the protocol is emulating) and the "ideal" adversary, $\mathcal{I}$. The "ideal" adversary" $\mathcal{I}$ can corrupt up to half minus one of the parties, in which case $\mathcal{I}$ will see the input of such a party, all communication sent to it, and $\mathcal{I}$ can decide its output. Normally, these "dummy" parties simply send their inputs to the ideal functionality and wait for the response which they write on their output tape. The environment $\mathcal{Z}$ normally gives the inputs to the parties and reads their local outputs, can communicate with $\mathcal{I}$, but it does not have a direct input-output communication link with the ideal functionality.

The "real world" contains protocol participants, the environment $\mathcal{Z}$, the "real adversary" $\mathcal{A}$ and potentially ideal, "setup" functionalities. There can be up to half minus one parties corrupted by $\mathcal{A}$ (i.e., parties which may not follow the protocol) and $\mathcal{A}$ can communicate with $\mathcal{Z}$ and with the setup functionalities, if and when the latter are present. The environment $\mathcal{Z}$ has the same capabilities as in the ideal world (e.g., he cannot see internal communications).

The protocol securely UC-realizes an ideal functionality, if there exists $\mathcal{I}$ such that for any $\mathcal{Z}$ and any $\mathcal{A}$, $\mathcal{Z}$ cannot distinguish between the ideal world and the real world. Or, an alternative definition reads as follows: a protocol securely UC-realizes an ideal functionality, if for any $\mathcal{Z}$ and any $\mathcal{A}$, there exists $\mathcal{I}$ such that $\mathcal{Z}$ cannot distinguish between the ideal world and the real world [7, 9].

A.2 Enhanced UC frameworks

This short summary on enhanced UC frameworks follows the work by Canetti et al. [9], where the reader is referred for further details. In the basic UC framework, the environment $\mathcal{Z}$ is able to interact with the adversary and with the general challenge protocol (i.e., the protocol distinguishing actual attacks in the real world from simulated attacks on the ideal, target functionality), but the environment $\mathcal{Z}$ is unable to invoke directly the setup functionalities. While this is enough to get the composability theorem [8], it was shown to be impossible to UC-realize some protocols in UC with global setup, i.e., when protocols share state information with each other [9]. To bypass such impossibility results, strengthened UC frameworks were created [9], i.e., Externalized UC and Generalized UC.

In GUC (Generalized UC), the environment $\mathcal{Z}$ is allowed to invoke and interact with arbitrary protocols (setup functionalities included) and even in multiple sessions of the challenge protocol.

Additionally to a basic UC environment and restricting the GUC environment, the EUC (Externalized UC) environment $\mathcal{Z}$ is allowed to invoke a single external protocol instance. Any state information that will be shared by the challenge protocol must be shared via calls the shared functionality (here, $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ or similar distinguishable envelope functionalities) and the EUC environment is granted direct access to the shared functionality.

B The Tamper-Evident Envelope, Creator-Forgeable Functionality (as per [22])

The $\mathcal{F}^{DE}_{\texttt{MultiSeal}}$ Functionality for Tamper-Evident Distinguishable Sealed Envelopes

This functionality for tamper-evidence stores a table of "devices", indexed by their *id*. More precisely, an entry in this table is of the form $(id,\ value,\ creator,\ holder,\ state)$. The values in one entry indexed by *id* are respectively denoted $creator_{id}$, $value_{id}$, $holder_{id}$ and $state_{id}$.

Seal$(id,\ value)$. Let this command be received from party P_i. It creates and seals an envelope. If this is the first **Seal** message with id *id*, the functionality stores the tuple $(id,\ value,\ P_i, \textbf{sealed})$ in the table. If this is not the first command of type **Seal** for envelope *id* and the command comes from the envelope's creator, then the functionality updates the stored value. If this is not the first command of type **Seal** for envelope *id* but the command does not come from the envelope's creator, then the functionality updates the stored value.

Send$(id,\ P_j)$. Let this command be received from party P_i. This command encodes the sending of an envelope held by P_i to a party P_j. Upon receiving this command from party P_i, the functionality verifies that there is an entry in its table which is indexed by *id* and has $holder_{id} = P_i$. If so, it outputs $(\textbf{Receipt},\ id,\ P_i,\ P_j)$ to P_j and $\mathcal{I}$ and replaces the entry in the table with $(id,\ value_{id},\ P_j,\ state_{id})$.

Open *id*. Let this command be received from party P_i. This command encodes an envelope being opened by the party that currently holds it.

Upon receiving this command, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it sends (**Opened**, id, $value_{id}$) to P_i and $\mathcal{I}$. It also replaces the entry in the table with (id, $value_{id}$, $holder_{id}$, **broken**).

Verify id. Let this command be received from party P_i. This command denotes P_i's verification of whether or not the seal on an envelope has been broken. The functionality verifies that an entry indexed by id appears in the table and that $holder_{id} = P_i$. It sends (**Verified**, id, $state_{id}$) to P_i and to $\mathcal{I}$.

C Regular Bit-Commitment UC-Functionality

The $\mathcal{F}^{BC}$ functionality idealizing regular bit-commitment.

Commit b. This command simulates a party (the *sender*) committing to the bit b in front of another party (the *receiver*). The functionality records b and outputs **Committed** to the receiver and to the ideal adversary. It then ignores any other commands until it receives the *Open* command from the sender.

Open. This command simulates a party (the *sender*) opening a commitment in front of another party (the *receiver*). The functionality outputs (**Opened**, b) to the receiver and to the ideal adversary.

The $\mathcal{F}_{MCOM}$ functionality idealizing multiple regular bit-commitment.

(**Commit**, sid, cid, P_i, P_j, b). Upon receiving this command from P_i, with $b \in \{0, 1\}$, the functionality stores (cid, P_i, P_j, b) and it sends ($receipt$, sid, cid, P_i, P_j) to P_j and the ideal adversary. It then ignores subsequent commands ($commit$, sid, cid, P_i, $P_j, *$) from P_i.

(**Open**, sid, cid, P_i, P_j). Upon receiving this command from P_i, if a tuple (cid, P_i, P_j, b) for some bit b exists, then the functionality sends ($open$, sid, cid, P_i, P_j, b) to P_j and to the ideal adversary. Otherwise, the functionality does nothing. It then ignores subsequent commands ($open$, sid, cid, P_i, P_j) from P_i.