Several Weak Bit-Commitments Using Seal-Once Tamper-Evident Devices*

Ioana Boureanu and Serge Vaudenay
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
{ioana.boureanu,serge.vaudenay}@epfl.ch

Abstract

Following both theoretical and practical arguments, we construct UC-secure bit-commitment protocols that place their strength on the sender's side and are built using tamper-evident devices, e.g., a type of distinguishable, sealed envelopes. We show that by using a second formalisation of tamper-evident distinguishable envelopes we can attain better security guarantees, i.e., EUC-security. We show the relations between several flavours of weak bit-commitments, bit-commitments and distinguishable tamper-evident envelopes. We focus, at all points, on the lightweight nature of the underlying mechanisms and on the end-to-end human verifiability.

1 Introduction

Most of the recent approaches to primitive-construction employ the universal composability (UC) framework [6] in order to specify and prove the correctness/security of their cryptographic designs. The UC framework is a formalism that allows for cryptographic protocols to be computationally analysed in a single session, yet the security guarantees thereby obtained are preserved when multiple sessions are composed concurrently, in parallel and/or sequentially. See Appendix A, for a short wrap-up on UC frameworks [6, 8] and UC proofs. In [6], Canetti shows that any polynomial-time multi-party functionality is feasible in the UC framework if the majority of participants are honest. Otherwise, feasibility is usually attained if the models are augmented with "setup-assumptions", obtaining the so-called "UC hybrid models" (i.e., extra ideal functionalities are made available to the parties).

UC-formalisations of tamper-evident and tamper-resistant hardware devices have been used as setups to UC-realize different cryptographic primitives, from bit-commitment to polling schemes [14, 16, 13, 17, 18, 20, 19]; the tamper-evidence of a device implies that, if tampered with, the device will signal the inflicted abnormalities, whereas tamper-resistance denotes the impossibility of tampering with the device. Tamper-evidence-based UC-secure protocols [17, 18, 20] also bear lightweight, humanly constructible/verifiable cryptographic mechanisms. To realize UC-secure weak bit-commitment (WBC) protocols, a type of distinguishable tamper-evident envelopes were shown sufficient and necessary (in the sense that simpler functionalities of distinguishable tamper-evident containers are not sufficient to realize bit-commitments) [19]. The protocols thereby constructed placed their "strength" on the receiver's side, i.e., it is the receiver who creates the tamper-evident devices or prepares them. In [19], Moran and Naor raise the 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* protocols. Along similar lines, Brassard, Chaum and Crépeau in foundation papers have long made the question: "Is it preferable to trust Vic or Peggy? We do not know, but it sure is nice to have the choice. [4]".

Contributions. The contributions of this paper are as follows:

- We create weak bit-commitments that place the (adversarial) strength on the committer side, i.e., sender-strong WBC, and that are UC-secure. To achieve this, we require a new formalisation of distinguishable envelopes and use it as a UC setup functionality (see the motivations below).
- We describe a hierarchy of ideal functionalities for sender-strong weak bit-commitments and UC-realize them. In this, we relate better with the existing literature in the field (see Section 2.2 for details).

*A shorter version of this paper appeared in the proceedings of ProvSec'12 [3].

- We relate our first functionality of distinguishable envelopes ($\mathcal{F}^{DE}_{\texttt{OneSeal}}$), the standard UC-functionality of bit-commitment ($\mathcal{F}^{BC}$) and those of WBC (already existing and newly introduced herein), showing most implication-relations between them.
- We introduce a second distinguishable envelope functionality ($\mathcal{F}^{\texttt{purpoted}DE}_{\texttt{OneSeal}}$), which allows for the corresponding DE-based WBC protocols herein and the ones in [19] to be enjoyed a stronger security notion: be not only UC-secure, but also EUC-secure.

Motivation for Our Formalisation of Tamper-Evident Envelopes.
I. As Moran et al. state in [19], there are many ways to formalise tamper-evident containers, reflecting the different requirements of the possible physical implementations of such devices. The *sole* motivation given in [19] for allowing creator-forgeability is the desiderata of creating more complex, somewhat stronger protocols.

But, when it comes to placing this sort of asymmetric strength on the sender's side, it only makes sense to construct commitment protocols that are, in the standard sense, *computationally hiding* and somewhat binding, i.e., the receiver is powerless and the sender can possibly equivocate his commitments. (By contrast, in [19] are both *partially* hiding and *partially* binding and are then amplified.) In this context, we conjecture that it is not possible to be based only on tamper-evident envelopes à la Moran et al. [19] and construct *hiding* sender-strong bit-commitment protocols, which would further be UC-secure in the same time. To overcome this shortcoming, we have herein slightly modified the original, tamper-evident envelope functionality from [19], preventing the creator from resealing envelopes. Hence, we model *seal-once* distinguishable tamper-evident envelopes (or, envelope allowing *one-seal* only). By contrast, the functionality in [19] formalises a *multi-seal* distinguishable tamper-evident envelope.

II. The previous protocols designed using tamper-evident envelopes à la Moran et al. [19] were only UC-secure and not EUC-secure. We noted that if we relaxed the forging abilities of the envelope-creator in the aforementioned way and we furthermore allow for purported destinator for envelopes the corresponding DE-based protocols obtained both here and in [19] attain EUC-security and not only UC-security.

Our Weak Bit-Commitments: a theoretical viewpoint.
Alongside the UC-framework, sender-strong weak bit-commitments are also interesting by traditional theoretical lines, where they are easier to construct (see Section 3.5). In [4], outside of the UC-framework, Brassard et al. proved that the existence of "chameleon" bit-commitments[1] implies the existence of zero-knowledge (ZK) proofs of knowledge which were MA-protocols (i.e., where the verifier sends independent bits). Moreover, in [2] Beaver proved that in order for the aforementioned ZK proofs of knowledge (PoK) to be provable secure against adaptive adversaries, the chameleon bit-commitments (BC) are not enough, but content-equivocable bit commitments are needed (i.e., the equivocation is possible only if a record of the traffic between the sender and the receiver is available to the sender and not other types of witnesses, like parts of messages). One of our weak BC functionalities, $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$, models this last type of important weak bit-commitments.

Our Weak *Sender-Strong* Bit-Commitments: a practical viewpoint.
Practice also imposes situations 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 [10, 15, 12], 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".

A real-life situation were the committer/sender should be given the chance to "change his mind" is the case in negotiation-based protocols where the receiver is known or thought to be corrupt (e.g, hostage-release cases, reputations [1], anonymous special auctioning [23], etc.).

To sum up, we are motivated to present certain means of attaining different UC and EUC-secure, bit-commitment protocols placing their strength on the sender's side, i.e., *sender-strong (SS).* For this, we use two new lightweight tamper-evident devices justified in the given context.

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

Related Work

We will hereby refer to lines of work using tamper-resistance and tamper-evidence to construct cryptographic protocols, designed mainly in the UC framework [6].

A series of works on designing UC-secure protocols using tamper-resistant building blocks have recently emerged [14, 9, 16, 13, 20]. For example, the formalism by Katz, in [14], opens for the creation and exchange of tamper-proof hardware tokens used in a commitment protocol, which is UC-secure if the tokens are stateful and the DDH assumption holds. In [9], the two-party computation can equally be UC-realized, but the model is relaxed: the tokens are stateless and the assumption is switched to the existence of oblivious transfer protocols in the UC plain model. Similar results are obtained using tamper-resistant devices as building blocks in a model called the trusted agent model [16]. Like in [9] and unlike in [14], Mateus and Vaudenay [16] permit a freer flow of devices from their creator to their users and backwards. Similar protocols are constructed by Moran et al., in [20], 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 also been underlined [20] within this context of using tamper-resistant hardware as UC-setup.

Simpler cryptographic protocols UC-constructed using not tamper-resistant devices, but tamper-evident devices in form of sealed envelopes and sealed locks have been studied in [19, 17, 18]. All the protocols thereby presented place their strength on the receiver's side.

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 [19]. To relate more closely to Moran and Naor's work [19], 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 [17, 19]. 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 [11, 2] 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

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

In particular, 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 $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ ideal functionality, running in the presence of parties $P_1, \ldots, P_n$ and an ideal adversary $\mathcal{I}$ is described in the following.

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 of the differences from the corresponding functionality presented in [19] is that the one 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 herein. However, a more important difference is that the creator of an envelope cannot re-seal it, i.e., he cannot forge the value stored initially inside the envelope. Hence, we use the syntagm "`OneSeal`" to refer to the functionality herein and, sometimes, we use the expression "`MultiSeal`" to designate the tamper-evident envelopes in [19]. This modification is driven by the fact that we could not yet construct a sender-strong, somewhat binding and *not partially hiding, but computationally hiding* bit commitment *that is also simulatable within UC*, using only creator-forgeable/multi-seal tamper-evident envelopes; we have not yet disproved this either.

Our change brings the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality closer to regular commitment than the tamper-evident functionality in [19] 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*.

For consistency, in Appendix B, the reader can find the tamper-evident envelope functionality corresponding to [19]. We denote it $\mathcal{F}^{DE}_{\texttt{MultiSeal}}$.

2.2 Target UC Functionalities of Bit-Commitment

We now describe our target functionalities $\mathcal{F}^{q-\mathrm{WBC}}_{\star}$ that respectively model different weak bit-commitment (WBC) protocols, where $\star \in \{\texttt{EscapeThenMayCheat}, \texttt{LearnAtCommitment}, \texttt{LearnAtOpening}\}$. In this fashion, we can relate the WBCs UC-realized herein both with traditional weak bit-commitments [2] of theoretical importance (e.g., see our $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$), and with weak bit-commitments UC-created in [19] 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 $\mathcal{I}$[2]. 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.

[2]Throughout, the fact that the output is sent to the ideal adversary as well is inherent to the UC framework, i.e., see the UC-notion of "delayed output".

EquivocatoryOpen c. Upon receiving the **EquivocatoryOpen** c command from the sender, with $c \in \{0,1\}$, the functionality verifies that $bit = \square$. Then, the functionality $\mathcal{F}^{q-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ 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 [19], 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).

Next, we give a similar functionality where in turn the possibility of equivocation becomes clear only at the opening phase.

The $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ 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 $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ 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 $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ 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 $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ outputs (**Opened**, $\overline{bit}$) to the receiver and to the ideal adversary. With probability $1-q$, the $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ 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 $\mathcal{F}^{q-\text{WBC}}_{\texttt{LearnAtOpening}}$ 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., i.e., it is at some point during the opening phase that the potential cheater *learns* about his opportunity, similarly to traditional lines in [2]. As aforementioned, note that while the cheating is allowed, it does not necessarily need to happen.

Note that sender-strong 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. Amplifications techniques could then be applied. However, we take the view that once the protocols that we seek are sender-strong, UC-realizing a functionality which is only partially hiding is would contradict the aim for the senders' strength (hence, the "*computationally hiding*" UC functionalities that we have given above).

3 UC (Sender-Strong) Bit-Commitments

Driven by the theoretical and practical motivations presented in the introduction, we are now going to present protocols that UC-implement the weak bit-commitment (WBC) functionalities above, use the herein introduced functionalities of distinguishable envelopes as the UC-setup and place their strength on the senders' sides.

We start with the protocol `Pass&MayCheat` which UC-realizes the $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\texttt{EscapeThenMayCheat}}$ using the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$. We continue with the protocol `CommitEnablesCheat` and `OpenEnablesCheat` which respectively UC-realize the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommitment}}$ and the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtOpening}}$ functionalities, using the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$. 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 [8], i.e., not only UC-security.

3.1 `Pass&MayCheat` — a SS-WBC protocol à la Moran and Naor [19]

The `Pass&MayCheat` Protocol:

The Commitment Phase.

1. 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 random $x \in \{0, 1\}$. Each pair "contains" its own value x. He sends two envelopes, one from each pair, to the receiver R.
   (E.g., The pairs are $pair_1 = (E_1, E_2)$, $pair_2 = (E_3, E_4)$ and S sends, e.g., E_1, E_3 to R).

2. 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 above.).
   Then, R sends them back without opening them.

3. 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.)

4. The receiver verifies that the envelopes received do not have the ids stored already. If they do, 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 to request back one envelope.
   Given the steps of the protocol so far, note that the opened envelope together with the requested one form an initial pair. Also, once the sender has sent this requested envelope, the sender will be left with the other of the initial pairs at his end. These comments equally apply to the equivocatorial commitment phase to follow. 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, e.g., 1, thus requesting back envelope E_1.)

5. 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. He also sends the value $d{=}b \oplus x$, where b is the bit he is committing to and x 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, $d{=}b \oplus x$, where x is in E_3 and/or in E_4).

6. The receiver R opens the last envelope received and checks that the value 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).

The EquivocatoryCommitment Phase.

1. A sender S seals four envelopes and creates two pairs out of them such that one pair contains the set $\{x, x\}$ of values, for a random $x \in \{0, 1\}$ and the other pair contains the values $\{0, 1\}$. He sends two envelopes, one from each pair, to the receiver R.
   (E.g., The pairs are $pair_1 = (E_1, E_2)$, $pair_2 = (E_3, E_4)$ and S sends E_1, E_3 to R).

2. The receiver R stores the identifiers of the envelopes in a register W.
   (I.e., it stores $(1, 3)$).
   Then, R sends them back without opening them.

3. The sender S verifies that the recently returned envelops 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.)

4. The receiver verifies that the envelopes received do not have the ids stored already. If this is not the case, he halts. Otherwise, he opens one of these envelopes, sends back the other one without opening it and requests one envelope with the id stored already in W. 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 requests, e.g., E_1.)

5. 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, together with the value $d{=}b \oplus x$, where b is the bit he is committing to and x 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, $d{=}b \oplus x$, where x is in E_3 and/or in E_4).

6. The receiver R opens the last envelope received and checks that the value 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 Opening Phase.

1. The sender S sends one envelope E_k in the remaining pair, i.e., $E_k \in A$ or $k \in \{i, j\}$.

2. The receiver R checks that E_k is in the set A (by checking the ids). If so, he opens the envelope E_k to find the value b_k hidden inside and then he sets the commitment-bit b' to $d \oplus b_k$. Otherwise, the receiver halts.

The EquivocatoryOpening Phase.

1. 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.

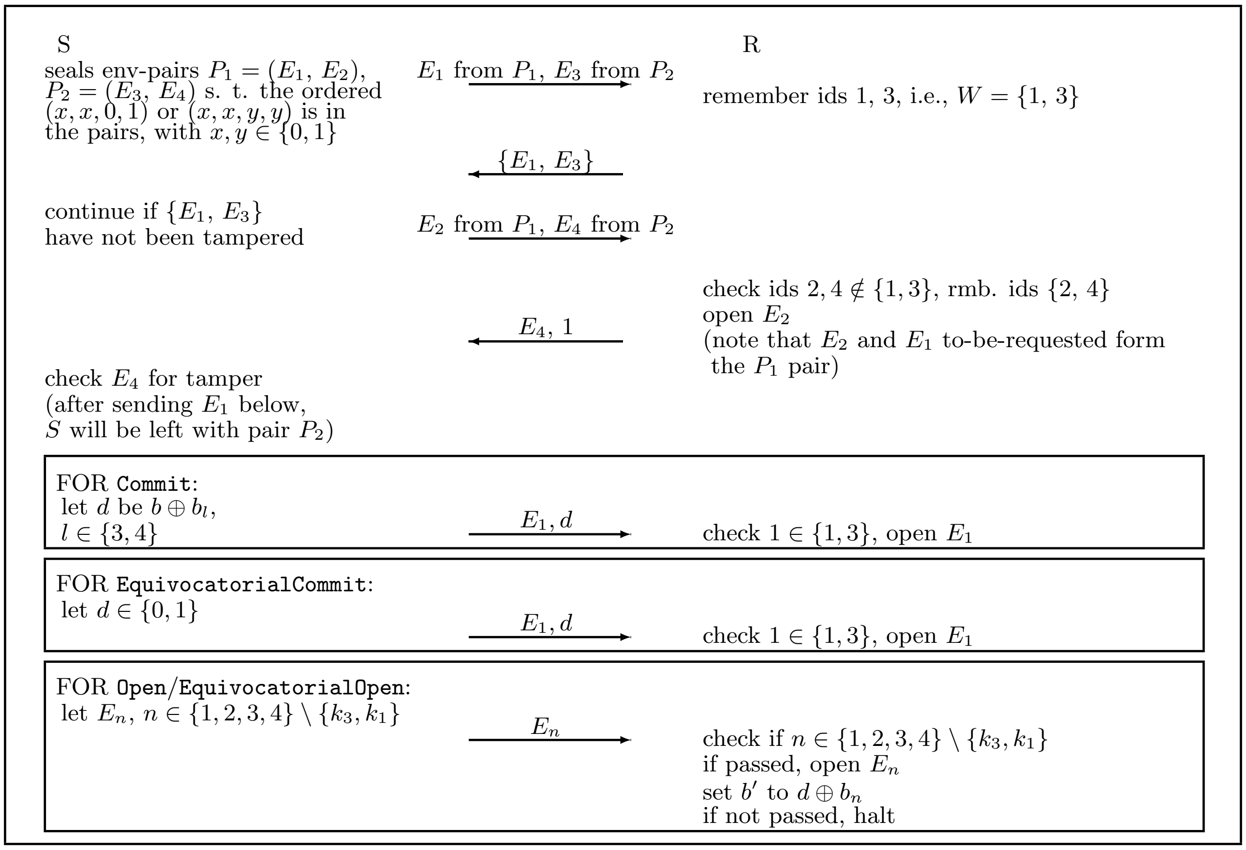
2. The receiver R checks that E_k is in the set A (by checking the ids). If so, he opens the envelope E_k to find the value b_k hidden inside and then he sets the commitment-bit b' to $d \oplus b_k$. Otherwise, the receiver halts.

In the following figure, we give an illustration of the protocol above, in a symmetric way (i.e., E_1 and E_2 could be interchanged in their appearances, etc.).

Explanations on the `Pass&MayCheat` protocol.

Assume first that the sender S creates pairs of envelopes such that each contains the set $\{x, x\}$ of values and that the sender respects the calculation of d for the non-equivocable case. It is clear that at the end of the protocol, the sender has no choice but to open to the correct bit. If, in turn, S does not form d as specified and R does follow the protocol, then S may not be able to open.

Assume now that the sender S creates pairs of envelopes such that one contains the set $\{x, x\}$ of values and the other contains the set $\{0, 1\}$ of values. Depending on the choice of R to open envelopes, the sender may continue the protocol; clearly this is possible 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 such cases, S can clearly open the value d to any bit-value, since he is left with x and its negation $\overline{x}$ in the pair A at his end.


S
R
seals env-pairs $P_1 = (E_1, E_2)$, $P_2 = (E_3, E_4)$ s. t. the ordered $(x, x, 0, 1)$ or (x, x, y, y) is in the pairs, with $x, y \in \{0, 1\}$
E_1 from P_1, E_3 from P_2
remember ids 1, 3, i.e., $W = \{1, 3\}$
$\{E_1, E_3\}$
continue if $\{E_1, E_3\}$ have not been tampered
E_2 from P_1, E_4 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)
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 $b \oplus b_l$, $l \in \{3, 4\}$
E_1, d
check $1 \in \{1, 3\}$, open E_1
FOR EquivocatorialCommit:
let $d \in \{0, 1\}$
E_1, d
check $1 \in \{1, 3\}$, open E_1
FOR Open/EquivocatorialOpen:
let E_n, $n \in \{1, 2, 3, 4\} \setminus \{k_3, k_1\}$
E_n
check if $n \in \{1, 2, 3, 4\} \setminus \{k_3, k_1\}$
if passed, open E_n
set b' to $d \oplus b_n$
if not passed, halt

Note that the receiver R cannot cheat without being caught: i.e., torn envelopes are obvious to the sender and opening by R of more than two envelopes –to try and break the hiding property– is not possible due to the stage-by-stage unsealing enforced by the protocol.

Also, note that the envelopes used within are seal-once envelopes. Thus, the sender S is not able to change the values x stored inside the envelopes, after step 4 of the commitment phase (say, in order to avoid being caught by R).

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 $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ 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 $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$.

   $\mathcal{I}$ 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 $\mathcal{F}^{\frac{1}{2}-\mathrm{WBC}}_{\mathtt{EscapeThenMayCheat}}$ 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 $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality.

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

Lemma: *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 $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$. (No matter what command **EquivocatoryCommit** or **Commit** was sent to $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ by dummy-Alice, $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ will send just **Committed** or $\perp$).

If $\perp$ was sent by $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$, 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 $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$).

Now consider the case that **Committed** was sent by $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$. 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 $\mathcal{A}(Bob)$.

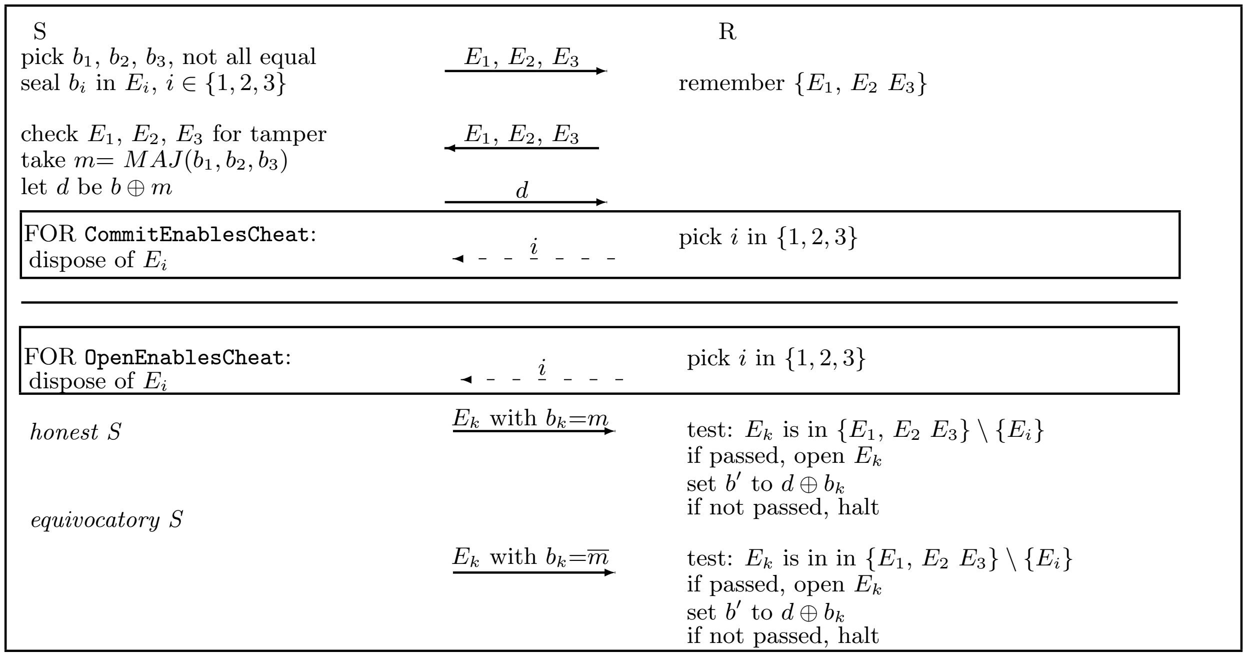
We continue with the simulation of the opening phase. As anticipated above, the ideal adversary $\mathcal{I}$ waits for (**Opened**, b) to be sent by the $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$ functionality.

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 $\mathcal{F}^{\frac{1}{2}-\text{WBC}}_{\texttt{EscapeThenMayCheat}}$. □

3.2 The `CommitEnablesCheat` and `OpenEnablesCheat` Protocols

The `CommitEnablesCheat` Protocol:


S
R
pick b_1, b_2, b_3, not all equal
seal b_i in E_i, $i \in \{1,2,3\}$
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 $b \oplus m$
E_1, E_2, E_3
d
FOR `CommitEnablesCheat`:
dispose of E_i
i
pick i in $\{1,2,3\}$
FOR `OpenEnablesCheat`:
dispose of E_i
i
pick i in $\{1,2,3\}$
honest S
E_k 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
E_k with b_k=$\overline{m}$
test: E_k is in 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

The Commitment Phase. The sender wants to commit to a bit b and proceeds as it follows.

1. 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.

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

3. The sender 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.

4. The receiver sends the identifier i of an envelope that the sender should dispose of, i.e., $i \in \{1,2,3\}$. Let the set S=$\{E_1, E_2, E_3\} \setminus \{E_i\}$ denote the set of remaining envelopes.

5. 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.

1. The non-equivoquing sender sends an envelope E_k such that b_k=m.

2. The equivoquing sender sends an envelope E_k such that $b_k{=}\overline{m}$.
3. The receiver tests that $E_k \in S$ 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[3] 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 figure to follow.

Note once more that, unlike in the `Pass&MayCheat` protocol, in `CommitEnablesCheat` and `OpenEnablesCheat` protocols, the committer can cheat with some probability (i.e., $\frac{2}{3}$), yet this is not influenced by him being caught cheating, but rather by a mere choice of the receiver.

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 [5]. Nevertheless, the existing solutions [5, 11, 21] to problems of the latter kind are receiver-strong, not sender-strong. Also, they are not constructed to be UC-secure, but secure by classical lines, which may be weaker. Moreover, those original constructions involve error-correction codes and/or pseudo-random generators being manipulated by the participants. Thus, those primitives are also beyond our cryptographically lightweight scope. Therefore, to obtain senders' strength, UC-security, simplicity and human operability we have proposed protocols `CommitEnablesCheat` and `OpenEnablesCheat` above.

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

We detail on the `CommitEnablesCheat` protocol above, the explanations on `OpenEnablesCheat` being very similar and immediately following. Let us consider the case where the parties follow the protocol. We can see that if S prepares the envelopes correctly (i.e., they contain a permutation of $\{x, x, \overline{x}\}$, $x \in_U \{0,1\}$) and he adheres to step 2, then at step 3, the value $m{=}x$. No matter what value b_i has (i.e., x or $\overline{x}$), in the set S of remaining envelopes there is always an envelope E_k with the value x inside that opens the commitment correctly. With probability $\frac{2}{3}$, the set S still contains an envelope with value $\overline{x}$. In this last case, S could open his commitment to the flipped bit (i.e., point 2 in the opening phase). By the above, the protocol is complete. One can see that the case where S does not follows the protocol in terms of envelope sealing does not bring him any benefit. In Theorem 3.2, we formalise the above explanations, in the context of the UC framework.

Theorem 3.2 *In a hybrid UC-model, where the setup is the $\mathcal{F}^{DE}_{\mathit{OneSeal}}$ functionality, the `CommitEnablesCheat` and `OpenEnablesCheat` protocols UC-realize the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathit{LearnAtCommitment}}$ and the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathit{LearnAtOpening}}$ 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 $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality. Note that $\mathcal{I}$ intercepts the communication between $\mathcal{A}$ and the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ 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 (**Verified**, id, ok) reply sent by $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$.

$\mathcal{I}$ 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 $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtCommit}}$ functionality. The functionality replies with a

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

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 $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$.

Lemma: *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 $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$.

$\mathcal{I}$ 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 $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$.

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: *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 $\mathcal{F}^{\frac{2}{3}-\text{WBC}}_{\texttt{LearnAtCommit}}$. □

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 $\mathcal{F}_{\star}^{q^k-WBC}$. 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 $\mathcal{F}_{\star}^{q-WBC}$ functionality, we can construct a protocol* `Amplified_✠` *that UC-realizes the $\mathcal{F}^{BC}$ functionality, where*
$\star \in \{$`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., [21]. The proofs therein follow long-established lines, i.e., not the UC framework [4]. 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 3.1 *The protocol* `Amplified_Pass&MayCheat` *UC-realizes the $\mathcal{F}^{q^k-WBC}_{\mathit{EscapeThenMayCheat}}$ 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 $\mathcal{F}^{q-WBC}_{\texttt{EscapeThenMayCheat};\ \ell}$, $\ell \in \{1, \ldots, k\}$. 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 $equiv_j{=}\perp$, otherwise).

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 $jinJ$ 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[5];

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

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

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.

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: *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 3.2 *In a hybrid UC-model, where a linear number of k instances of the $\mathcal{F}^{q-\mathrm{WBC}}_{\mathtt{LearnAtCommitment}}$ 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 $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtCommitment};\,\ell}$, $\ell \in \{1, \ldots, k\}$.

$\mathcal{I}$ flips coins such that for each $\mathcal{F}^{q-\mathrm{WBC}}_{\texttt{LearnAtCommitment};\,l}$, 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 [22], 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 $\mathcal{F}_1^{BC}$ 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: *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 3.3 *: In a hybrid EUC-model, where the setup is the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality, the protocol `CommitEnablesCheat` does not EUC-realizes the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommit}}$ 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). □

3.5 (Stronger) Universally Composable Security

A UC-oriented note is that something as little 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 does impact the UC-simulatability. A protocol only different from `CommitEnablesCheat` in that it inverts the order of events 3 and 4 in the commitment phase does not UC-realize the $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\texttt{LearnAtCommitment}}$ functionality, while it is perfectly hiding and binding with probability $\frac{2}{3}$ in the classical sense. Thus, it seems to be easier to construct q-weak bit-commitments using just a formalisation of distinguishable envelopes, but when sender-strength and UC-security are both sought after subtle difficulties arise ($q \in (0, 1)$).

Another important UC note is that the protocols above (and in fact all weak bit-commitment protocols constructed previously for the receiver-strong case in Moran and Noar's work [19]) are not secure in stronger versions of the UC framework, e.g., GUC (Generalised UC) or EUC (externalised UC) [8]. For a wrap-up on GUC (Generalised UC) or EUC (externalised UC), see the Appendix, Section A.2. To support this claim, it is enough to show that the protocols are not secure in the EUC framework. So, in an EUC model with the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$-setup consider an environment that prepares the envelopes and feeds them to the adversary. It is clear that the ideal adversary cannot "extract" the bit b to commit to and thus he cannot indistinguishably simulate the commitment phase.

As aforementioned, along very similar lines the receiver-strong protocols in previous works [19] are not EUC-secure either. We modify the $\mathcal{F}^{DE}_{\texttt{OneSeal}}$ functionality slightly such that when used as a setup, we attain EUC-security of the protocols herein and those WBC protocols in Moran and Noar's work [19].

$\mathcal{F}^{\mathrm{purpoted}DE}_{\texttt{OneSeal}}$**: 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 envelop 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 main difference between $\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$ and the original $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality is that an envelope is created for a specifically intended holder and this holder is consequently notified with a message of the form (*id*, *creator*). Note that this enhancement is realistic (i.e., if to be used in a protocol, the delivery address of the receiver is to be specified by a manufacturing body). However, note that the functionality does not store or reveal publicly the creators of the envelopes (i.e., that would be a stronger enhancement, akin to signing the tamper-evident devices). With this modification, the holder-to-be knows that a specific envelope has been freshly produced by a specific creator. Intuitively, this prevents the weakness in the proof of Lemma 3.3 from happening, i.e., R cannot accept envelopes that are not (newly) meant for him. Also, the creator is authenticated by the functionality, in the sense that he cannot use envelopes made by others. In a larger sense, this can prevent relay attacks. More formally, the following holds.

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.*

The proof of the above theorem follows from the proofs of Theorem 3.2 and that of Lemma 3.3, 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.

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

Given the results above and those in [19], we have that $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ (or $\mathcal{F}^{\mathtt{purpoted}DE}_{\mathtt{OneSeal}}$) can create receiver-strong and sender-strong weak UC bit-commitments, which in turn can be amplified to obtain regular UC bit-commitments. The $\mathcal{F}^{BC}$ functionality or a flavour of it (see $\mathcal{F}_{COM}$ in [6]) constitutes a sufficient setup to UC-realize a ZK protocol [6]. Under these circumstances, it is definitely interesting to investigate the existent implications between different sort of weak bit-commitment, regular bit-commitment and tamper-evident envelopes, in the UC framework.

Firstly, note that a multiple commitment $\mathcal{F}_{MCOM}$ [6] setup suffices to 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 $q \in (0, 1)$. (We recall the $\mathcal{F}_{MCOM}$ [6] functionality in Section C of the Appendix). In other words, several instances of the regular bit-commitment functionality $\mathcal{F}^{BC}$ suffice to UC-construct a sender-strong weak bit-commitment. In particular, three regular bit-commitment $\mathcal{F}^{BC}$ functionalities (see Section C of the Appendix) UC-construct a $\frac{2}{3}$-WBC which is sender-strong. Let a protocol $\mathcal{P}$ be obtained from protocol `CommitEnablesCheat` where the creation and transmission of the three envelopes is respectively replaced by creation and transmission of three commitments using $\mathcal{F}_{MCOM}$ or using three respective instances of $\mathcal{F}^{BC}$. An analogous fact holds also for $\mathcal{F}^{\frac{2}{3}-\mathrm{WBC}}_{\mathtt{LearnAtOpening}}$, where to construct the protocol $\mathcal{P}$ we use `OpenEnablesCheat` instead of `CommitEnablesCheat`.

Secondly, as a consequence of Theorem 3.3, note that all sender-strong weak BCs UC-imply regular BCs.

Thirdly, it should be answered whether bit-commitment setup suffice to UC-realize the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ distinguishable tamper-evident envelope functionality. We conjecture that question 1 has a negative answer. This is intuitively due to the fact that in bit-commitments it is the sender who opens the commitment and, in an envelope-emulating protocol, it should be the envelope's creator who needs to perform the corresponding opening. But, in order for this to be generally possible, an envelope creator ought to know the current envelope holder, which is not the case in our formalisations (i.e., envelopes can be passed on from holder to holder, without the notification of the creator).

To sum up, amplification proofs considered, we have completed the picture of UC-realisability of different flavours of sender-strong weak BC with tamper-evident envelopes and of their relation to (almost) regular BC and receiver-strong weak BCs by Moran and Naor [19]. To some level, we can say that all weak BCs are equivalent to regular BCs. We leave the EUC or the GUC correspondents of the implications enumerated above as open questions.

5 Conclusions

Answering a variant of the open question in Moran and Naor's work [19] and several practical needs [10, 15, 12], we conclude that simple, sealed envelopes can also create sender-strong (weak) bit-commitments protocols. In the process, we have also discussed the fact that the protocols in [19] are not EUC-secure but only UC-secure. We mainly focused on creating sender-strong bit-commitments with the same level of security. Nevertheless, we showed how to modify the $\mathcal{F}^{DE}_{\mathtt{OneSeal}}$ functionality given in Moran and Naor's work [19] such that we also create (weak) bit-commitment protocols that are EUC-secure. We showed lightweight amplification proofs of our WBC protocols. We lastly discussed some of the implications between UC weak BCs, UC regular BCs and distinguishable tamper-evident UC envelopes. The interest in *weak* BC protocol per se was motivated by both theoretical and practical reasons. The GUC-security of our schemes remains to be discussed.

References

[1] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Tuttle. Collaboration of Untrusting Peers with Changing Interests. In *Proceedings of the 5th ACM conference on Electronic commerce*, EC '04, pages 112–119, New York, NY, USA, 2004. ACM.

[2] 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.

[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*, Lecture Notes in Computer Science, pages 70–87, 2012.

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

[5] 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.

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

[7] 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.

[8] 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/`.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

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

[23] F. Stajano and R. Anderson. The Cocaine Auction Protocol: On the Power of Anonymous Broadcast. In *Proceedings of the 3rd International Workshop on Information Hiding*, IH '99, pages 434–447, London, UK, 2000. Springer-Verlag.

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" [6].

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 [6, 8].

A.2 Enhanced UC frameworks

This short summary on enhanced UC frameworks follows the work by Canetti et al. [8], 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 [7], 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 [8]. To bypass such impossibility results, strengthen UC framework were created [8], 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 [19])

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, **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 (**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.