Analysis and Improvement of Lindell's UC-Secure Commitment Schemes

Olivier Blazy[1], Céline Chevalier[2], David Pointcheval[3], and Damien Vergnaud[3]

[1] Ruhr-Universität Bochum, Germany
[2] Université Panthéon-Assas, Paris, France
[3] ENS, Paris, France*

Abstract. In 2011, Lindell proposed an efficient commitment scheme, with a non-interactive opening algorithm, in the Universal Composability (UC) framework. He recently acknowledged a bug in its security analysis for the adaptive case. We analyze the proof of the original paper and propose a simple patch of the scheme. More interestingly, we then modify it and present a more efficient commitment scheme secure in the UC framework, with the same level of security as Lindell's protocol: adaptive corruptions, with erasures. The security is proven in the standard model (with a Common Reference String) under the classical Decisional Diffie-Hellman assumption. Our proposal is the most efficient UC-secure commitment proposed to date (in terms of computational workload and communication complexity).

1 Introduction

Related Work. The Universal Composability (UC) framework introduced by Canetti [5] is a popular security paradigm. It guarantees that a protocol proven secure in this framework remains secure even if it is run concurrently with arbitrary —even insecure— protocols (whereas classical definitions only guarantee its security in the stand-alone setting). The UC framework enables one to split the design of a complex protocol into that of simpler sub-protocols.

Commitment schemes are one of the most important tools in cryptographic protocols. This is a two-phase protocol between two parties, a committer and a receiver. In the first *commit* phase, the committer gives the receiver an *in silico* analogue of a sealed envelope containing a value m. In the second *opening* phase, the committer reveals m in such a way that the receiver can verify it. As in the sealed envelope analogy, it is required that a committer cannot change the committed value (*i.e.*, he should not be able to open to a value different from the one he committed to), this is called the *binding* property. It is also required that the receiver cannot learn anything about m before the opening phase, this is simply called the *hiding* property.

The security definition for commitment schemes in the UC framework was presented by Canetti and Fischlin [7]. A UC-secure commitment scheme achieves

* ENS, CNRS & INRIA – UMR 8548.

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 534–551, 2013.

the binding and hiding properties under any concurrent composition with arbitrary protocols and it was shown, in [7], that it cannot be securely realized without additional assumptions. The common reference string (CRS) setting is the most widely used assumption when considering commitment schemes. In this setting, all parties have access to public information ideally drawn from some predefined distribution.

From a theoretical viewpoint, UC-secure commitments are an essential building block to construct more complex UC-secure protocols such as zero-knowledge protocols [11] and two-party or multi-party computations [9]. Moreover, a UC-secure commitment scheme provides *equivocability* (*i.e.*, an algorithm that knows a secret related to the CRS can generate commitments that can be opened correctly to any value) and *extractability* (*i.e.*, another algorithm that knows a secret related to the CRS can correctly extract the content of any valid commitment generated by anybody). Therefore, since their introduction, UC-secure commitments have found numerous practical applications in the area of Authenticated Key Exchange, either in Password Authenticated Key Exchange like [1,8,13], or the recent generalization to Language Authenticated Key Exchange [2].

Several UC-secure commitment schemes in the CRS model have been proposed. Canetti and Fischlin [7] and Canetti, Lindell, Ostrovsky, and Sahai [9] proposed inefficient non-interactive schemes from general primitives. On the other hand, Damgård and Nielsen [11], and Camenish and Shoup [4] (among others) presented interactive constructions from several number-theoretic assumptions.

Lindell [15] has recently presented the first very efficient commitment schemes proven in the UC framework. They can be viewed as combinations of Cramer-Shoup encryption schemes and Σ-protocols. He presented two versions, one proven against static adversaries (static corruptions), while the other can also handle adaptive corruptions. These two schemes have commitment lengths of only 4 and 6 group elements respectively, while their total communication complexity amount to 14 and 19 group elements respectively. Their security relies on the classical Decisional Diffie-Hellman assumption in standard cryptographic groups. Fischlin, Libert and Manulis [12] shortly after adapted the scheme secure against static corruptions by removing the interaction in the Σ-protocol using non-interactive Groth-Sahai proofs [14]. This transformation also makes the scheme secure against adaptive corruptions but at the cost of relying on the Decisional Linear assumption in symmetric bilinear groups. It thus requires the use of computationally expensive pairing computations for the receiver and can only be implemented over groups twice[1] as large (rather than the ones that do not admit pairing computations).

Contributions of the Paper. Recently, Lindell edited the ePrint version of his paper [16], to signal a bug in the original proof of the protocol design for adaptive corruptions. While there is no known detail on this bug, we detail on this paper a possible inconsistency on the *binding* property of the scheme. In

[1] It may be possible to adapt the scheme from [12] to asymmetric bilinear groups using the instantiation of Groth-Sahai proofs based on the Strong eXternal Diffie-Hellman assumption but our scheme will nevertheless remain more efficient.

order to avoid the above concern, we propose a simple patch to Lindell's scheme making it secure against adaptive corruptions.

However, our main contribution is on improving both Lindell's commitment schemes [15]. As mentioned above, the committer encrypts the value m (encoded as a group element) using the Cramer-Shoup encryption scheme [10]. In the opening phase, he simply reveals the value m and uses a Σ protocol to give an interactive proof that the message is indeed the one encrypted in the ciphertext. In Lindell's schemes, the challenge in the Σ protocol is sent to the committer using a "dual encryption scheme". Our improvement consists in noting that the receiver can in fact send this challenge directly without having to send it encrypted before. With additional modifications of the schemes, we can present two new protocols secure under the DDH assumption in the UC framework, against static and adaptive corruptions. Both schemes require a smaller bandwidth and less interactions than the original schemes:

- Static corruptions: the scheme requires the communication of 9 group elements and 3 scalars, where Lindell's original proposal requires 10 group elements and 4 scalars. The commit phase is non-interactive and the opening phase needs 3 rounds (instead of 5 in Lindell's scheme).
- Active corruptions: the scheme requires the communication of 10 group elements and 4 scalars, where Lindell's original proposal requires 12 group elements and 6 scalars. The commitment phase needs 3 rounds (instead of 5 in Lindell's scheme) and the opening phase is non-interactive.

Implemented on suitable elliptic curves over 256-bit finite fields, our schemes provide a 128-bit security level with a communication complexity reduced to only 3072 and 3584 bits respectively (see Table 1 for a detailed comparison). The computational workload of the new schemes has also slightly decreased compared to Lindell's original proposal and is significantly better than Fischlin *et al.*'s scheme from [12] since the new schemes do not require any expensive pairing computation and can be implemented in much smaller groups.

Table 1. Efficiency comparison of UC-secure commitment schemes (128-bit security)

Scheme	Communication Complexity (in bits)			Round Complexity		Computation Complexity		Adaptivity
	Commit	Decommit	Total	Commit	Decommit	exp.	pair.	
[15, § 3]	1024	2560	3584	1	5	27	-	✗
[15, § 4]	3072	1536	4608	5	1	36	-	✓
[12, § 3]	2560	8192	10752	1	1	41	69[2]	✓
[12, § 4]	18944	1536	20480	1	1	88	129[2]	✓
Fig. 5	1024	2048	3072	1	3	22	-	✗
Fig. 6	2048	1536	3584	3	1	26	-	✓

[2] These numbers can be reduced using batching techniques [3] but at the cost of additional exponentiations.

Outline of the Paper. We start by reviewing the standard definitions, in Section 2. We then present the original Lindell's commitment schemes in Section 3, followed by an explanation of a possible inconsistency and a simple correction.

Section 4 focuses on improving the original protocols. We will show how to reduce both the number of rounds and the number of elements exchanged, in both schemes. We then provide complete security proofs under the same computational assumption as for the original schemes, namely the DDH assumption.

2 Definitions

2.1 Commitments

A commitment scheme $\mathcal{C}$ is defined by 3 algorithms:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the scheme, implicitly given as input to the other algorithms;
- $\mathsf{Commit}(m; r)$ produces a commitment c on the input message $m \in \mathcal{M}$, using the random coins $r \xleftarrow{\$} \mathcal{R}$, and also outputs the opening information w;
- $\mathsf{Decommit}(c, m; w)$ decommits the commitment c using the opening information w; it outputs the message m, or $\perp$ if the opening check fails.

Such a scheme should be both *binding*, which means that the decommit phase can successfully open to one value only, and *hiding*, which means that the commit phase does not reveal any information about m.

These two properties can be obtained in a perfect, statistical or computational way, according to the power an adversary would need to break them. But essentially, a *perfectly binding* commitment scheme guarantees the uniqueness of the opening phase. This is achieved by an encryption scheme, which on the other hand provides the *computational hiding* property only, under the IND-CPA security. A *perfectly hiding* commitment scheme guarantees the perfect secrecy of m.

Some additional properties are sometimes required. The first is *extractability*, for a *perfectly binding* commitment scheme. The latter admits an indistinguishable Setup phase that also generates a trapdoor allowing message extraction from the commitment. Again, an encryption scheme is an extractable commitment, where the decryption key is the trapdoor that allows extraction. The second one is *equivocability*, for a *perfectly hiding* commitment scheme. The latter admits an indistinguishable Setup phase that generates a trapdoor allowing to open a commitment in any way.

2.2 Universal Composability Framework

The Universal Composability framework was introduced in [5]. The aim of the following is just to give a brief overview to have some common conventions.

In the context of multi-party computation, one wants several users P_i with inputs x_i to be able to compute a specific function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$

without leaking anything except y_i to P_i. One can think about Yao's Millionaires' problem [18]. Instead of following the classical approach which aims at listing exhaustively all the expected properties, Canetti did something else and tried to define how a protocol should ideally work: what are the inputs, and what are the available outputs. For that, he specified two worlds: the real world, where the protocol is run with some possible attacks, and the ideal world where everything would go smoothly, and namely no damage can be done with the protocol. For a good protocol instantiation, it should be impossible to distinguish, for an external player, the real world from the ideal one.

In the *ideal world* there is indeed an incorruptible entity named the *ideal functionality*, to which players can send their inputs privately, and then receive the corresponding outputs without any kind of communication between the players. This way the functionality can be set to be correct, without revealing anything except what is expected. It is thus perfectly secure. A protocol, in the *real world* with real players and thus possibly malicious players, should create executions that look similar to the ones in the previous world. This is to show that the communications between the players should not give more information than the description of the functionality and its outputs.

As a consequence, the formal security proof is performed by showing that for any external entity, that gives inputs to the honest players and gets the outputs but that also controls the adversary, the executions in the two above worlds are indistinguishable. More concretely, in order to prove that a protocol $\mathcal{P}$ realizes an ideal functionality $\mathcal{F}$, we consider an environment $\mathcal{Z}$ which can choose inputs given to all the honest players and receives back the outputs they get, but which also controls an adversary $\mathcal{A}$. Its goal is to distinguish in which case it is: either the real world with concrete interactions between the players and the adversary, or the ideal world in which players simply forward everything to and from the ideal functionality and the adversary interacts with a simulator $\mathcal{S}$ to attack the ideal functionality. We have to build a simulator $\mathcal{S}$ that makes the two views indistinguishable to the environment: since the combination of the adversary and the simulator cannot cause any damage against the ideal functionality, this shows that the adversary cannot cause any damage either against the real protocol.

The main constraint is that the simulator cannot rewind the execution as often done in classical proofs, since it interacts with an adversary under the control of the environment: there is no possible rewind in the real word, it is thus impossible too in the ideal world.

The adversary $\mathcal{A}$ has access to the communication but nothing else, and namely not to the inputs/outputs for the honest players. In case of corruption, it gets complete access to inputs and the internal memory of the honest player, and then gets control of it.

2.3 Ideal Functionality of Commitment

The ideal functionality of commitment is presented on Figure 1. It is borrowed from [6, 15], where a *public delayed output* is an output first sent to the adversary $\mathcal{S}$ that eventually decides if and when the message is actually delivered to

$\mathcal{F}_{\mathsf{mcom}}$ with session identifier sid proceeds as follows, running with parties $P_1, \ldots, P_n$, a parameter 1^k, and an adversary $\mathcal{S}$:

- Commit phase: Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ from P_i where $x \in \{0,1\}^{\mathsf{polylog}k}$, record the tuple $(\mathsf{ssid}, P_i, P_j, x)$ and generate a *public delayed output* $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ to P_j. Ignore further Commit-message with the same $(\mathsf{sid}, \mathsf{ssid})$.
- Reveal/decommit phase: Upon receiving a message of the form $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid})$ from party P_i, if a tuple $(\mathsf{ssid}, P_i, P_j, x)$ was previously recorded, then generate a *public delayed output* $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ to P_j. Ignore further reveal-message with the same $(\mathsf{sid}, \mathsf{ssid})$ from P_i.

Fig. 1. Ideal Functionality $\mathcal{F}_{\mathsf{mcom}}$ of Commitment

the recipient. In case of corruption of the committer, if this is before the $\mathsf{receipt}$-message for the receiver, the adversary chooses the committed value, otherwise it is provided by the ideal functionality, according to the Commit-message.

2.4 Useful Primitives

Hash Function Family. A hash function family $\mathcal{H}$ is a family of functions H_K from $\{0,1\}^*$ onto a fix-length output, either a bitstring or $\mathbb{Z}_p$. Such a family is said to be *collision-resistant* if for any adversary $\mathcal{A}$ on a random function $H_K \xleftarrow{\$} \mathcal{H}$, it is hard to find a collision. More precisely, this means that $\Pr[H_K \xleftarrow{\$} \mathcal{H}, (m_0, m_1) \leftarrow \mathcal{A}(H_K) : H_K(m_0) = H_K(m_1)]$ should be small.

Pedersen Commitment. The Pedersen commitment [17] is an *equivocable* commitment:

- $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with two independent generators g and ζ;
- $\mathsf{Commit}(m; r)$, for a message $m \xleftarrow{\$} \mathbb{Z}_p$ and random coins $r \xleftarrow{\$} \mathbb{Z}_p$, produces a commitment $c = \mathsf{Ped}(m, r) = g^m \zeta^r$, while r is the opening information;
- $\mathsf{Decommit}(c, m; r)$ outputs m and r, which opens c into m, and allows the validity test $c \stackrel{?}{=} g^m \zeta^r$.

This commitment is computationally binding under the discrete logarithm assumption: two different openings (m, r) and (m', r') for a commitment c, lead to the discrete logarithm of ζ in basis g. On the other hand, with this discrete logarithm value as additional information from the setup, one can equivocate any dummy commitment, when the input and opening values are known.

Cramer-Shoup Encryption. The Cramer-Shoup encryption scheme [10] is an $\mathsf{IND\text{-}CCA}$ version of the ElGamal encryption. By merging the Setup and KeyGen algorithm into a unique Setup algorithm, we make it into an extractable commitment scheme CS, where dk is the extraction key, and r is the witness for the opening.

- $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p.
- $\mathsf{KeyGen}(\mathsf{param})$ generates $(g_1, g_2) \xleftarrow{\$} \mathbb{G}^2$, $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \xleftarrow{\$} \mathbb{Z}_p^5$, and sets $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. It also chooses a Collision-Resistant hash function H_K in a hash family $\mathcal{H}$ (or simply second-preimage resistant). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, H_K)$.
- $\mathsf{Encrypt}(\mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = \mathsf{CS}(M; r) = (\mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^\omega)^r)$, where v is computed afterwards with $\omega = H_K(\mathbf{u}, e)$.
- $\mathsf{Decrypt}(\mathsf{dk}, C)$: one first computes $\omega = H_K(\mathbf{u}, e)$ and checks whether the equality $u_1^{x_1+\omega y_1} \cdot u_2^{x_2+\omega y_2} = v$ holds or not. If the equality holds, one computes $M = e/(u_1^z)$ and outputs M. Otherwise, one outputs $\perp$.

The IND-CCA security can be proven under the DDH assumption and the fact the hash function used is collision-resistant or simple second-preimage resistant. This also leads to a non-malleable commitment scheme, that is additionally extractable when the Setup outputs the decryption key dk.

3 Lindell's Commitment Protocols

We now have all the tools to review the two original Lindell's commitment schemes [15]. The first variant can be found on Figure 2. It only prevents static corruptions: the adversary can decide to run the protocol on behalf of a player, with its inputs, from the beginning, but cannot corrupt anybody when the execution has started. The second variant prevents adaptive corruptions with erasures.

3.1 Description of the Scheme for Adaptive Corruptions

It is presented on Figure 3. The main difference from the static case is to move some proof from the decommit phase to the commit phase.

3.2 Discussion

Adaptive Corruptions. Lindell has proven both schemes secure under the DDH assumption, the former in details but a sketch of proof only for the latter. And actually, as noted by Lindell in the last version of [16], the security against adaptive corruptions might eventually not be guaranteed.

He indeed proves that no adversary can choose a message x' beforehand, and do a valid commit/decommit sequence to x' where the simulator extraction, at the end of the commit phase, would output an x different from x'. However this is not enough as an adversary could still do a valid commit/decommit sequence to x' where the simulator extraction at the end of the commit phase would output an x different from x'. The difference between the two experiments is how much the adversary controls the value x': in the former x' has to be chosen beforehand, while in the latter x' is any value different from x.

We describe, on top of Figure 4, such a situation in which the adversary $\mathcal{A}$ plays as P_i, and makes the simulator extract the value x, while in fact committing

We have a CRS, consisting of $(p, \mathbb{G}, g_1, g_2, c, d, h, h_1, h_2)$, where $\mathbb{G}$ is a group of order p with generators g_1, g_2; $c, d, h \in \mathbb{G}$ are random elements in $\mathbb{G}$ and $h_1 = {g_1}^{\rho}$ and $h_2 = {g_2}^{\rho}$ for a random $\rho \in \mathbb{Z}_p$.
Intuitively, $(p, \mathbb{G}, g_1, g_2, c, d, h)$ is a Cramer-Shoup encryption key and $(p, \mathbb{G}, g_1, g_2, h_1, h_2)$ is the CRS of a dual-mode encryption scheme.
Let $G : \{0,1\}^n \to \mathbb{G}$ be an efficiently computable and invertible mapping of a binary string to the group.

The commit phase. Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$, where $x \in \{0,1\}^{n-\log^2(n)}$ and $\mathsf{sid}, \mathsf{ssid} \in \{0,1\}^{\log^2(n)/4}$, party P_i works as follows:

1. P_i computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
2. P_i picks $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $C = \mathsf{CS}(m; r)$, we will note ω the hash of the first three terms.
3. P_i sends $(\mathsf{sid}, \mathsf{ssid}, C)$ to P_j.
4. P_j stores $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, C)$ and outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
   P_j ignores any later commitment messages with the same $(\mathsf{sid}, \mathsf{ssid})$ from P_i.

The decommit phase. Upon receiving a message $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, P_i works as follows:

1. P_i sends $(\mathsf{sid}, \mathsf{ssid}, x)$ to P_j.
2. P_j computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$
3. (a) P_j picks $R, S \xleftarrow{\$} \mathbb{Z}_p$, a random challenge $\varepsilon \xleftarrow{\$} \{0,1\}^n$.
   He then sends $c' = ({g_1}^R {g_2}^S, {h_1}^r {h_2}^S G(\varepsilon))$ to P_i.
   (b) P_i picks $s \xleftarrow{\$} \mathbb{Z}_p$ and computes $(\alpha, \beta, \gamma, \delta) = ({g_1}^s, {g_2}^s, h^s, (cd^{\omega})^s)$.
   He then sends $(\mathsf{sid}, \mathsf{ssid}, \alpha, \beta, \gamma, \delta)$ to P_j.
   (c) P_j now opens c' by sending $(\mathsf{sid}, \mathsf{ssid}, R, S, \varepsilon)$ to P_i.
   (d) P_i checks if this is consistent with c' otherwise he aborts.
   P_i now computes $z = s + \varepsilon r$ and sends $(\mathsf{sid}, \mathsf{ssid}, z)$ to P_j.
   (e) P_j outputs $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ if and only if

$$g_1^z = \alpha {u_1}^{\varepsilon}, {g_2}^z = \beta {u_2}^{\varepsilon}, h^z = \gamma (e/m)^{\varepsilon}, (cd^{\omega})^z = \delta v^{\varepsilon}$$

Fig. 2. Lindell's Commitment Protocol, UC-Secure against Static Adversaries

(or actually opening) to another value x', but that is too late when the simulator discover the mistake. For the sake of clarity, we only mention the differences between this situation and the real protocol presented on Figure 3.

Any extraction done on C at the end of the commit phase would lead the simulator to believe to a commit to x, however the valid decommit outputs x'. Note however that this attack does not succeed very often since one needs, for a random ε, that $G^{-1}(mD^{1/\varepsilon})$ exists and can be parsed as $(x', \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.

Static Corruptions. We stress that this possible inconsistency comes from the move forward of the proof in the commit phase, even before the message x is strongly committed. The first protocol does not suffer from this issue.

We have a CRS, consisting of $(p, \mathbb{G}, g_1, g_2, c, d, h, h_1, h_2, \zeta, H_K)$, where $\mathbb{G}$ is a group of order p with generators g_1, g_2; $c, d, h \in \mathbb{G}$ are random elements in $\mathbb{G}$ and $h_1 = {g_1}^{\rho}$ and $h_2 = {g_2}^{\rho}$ for a random $\rho \in \mathbb{Z}_p$; H_K is randomly drawn from a collision-resistant hash function family $\mathcal{H}$.
Intuitively, the tuple $(p, \mathbb{G}, g_1, g_2, c, d, h, H_K)$ is a Cramer-Shoup encryption key, the tuple $(p, \mathbb{G}, g_1, g_2, h_1, h_2)$ is the CRS of a dual-mode encryption scheme, and the tuple $(p, \mathbb{G}, g, \zeta) =$ is the CRS of a Pedersen commitment scheme.
Let $G : \{0,1\}^n \to \mathbb{G}$ be an efficiently computable and invertible mapping of a binary string to the group.

The commit phase. Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$, where $x \in \{0,1\}^{n-\log^2(n)}$ and $\mathsf{sid}, \mathsf{ssid} \in \{0,1\}^{\log^2(n)/4}$, party P_i works as follows:

1. P_i computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
2. P_i picks $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $C = \mathsf{CS}(m; r)$, we will note ω the hash of the first three terms.
3. P_i picks $k_1 \xleftarrow{\$} \mathbb{Z}_p$, computes $c_p^1 = \mathsf{Ped}(H_K(C); k_1)$ and sends it to P_j.
4. P_j picks $R, S \xleftarrow{\$} \mathbb{Z}_p$, $\varepsilon \xleftarrow{\$} \{0,1\}^n$ and sends $c' = ({g_1}^R {g_2}^S, {h_1}^R {h_2}^S G(\varepsilon))$ to P_i.
5. P_i picks $s, k_2 \xleftarrow{\$} \mathbb{Z}_p$ and computes $(\alpha, \beta, \gamma, \delta) = ({g_1}^s, {g_2}^s, h^s, (cd^{\omega})^s)$.
   He then computes and sends $c_p^2 = \mathsf{Ped}(H_K(\alpha, \beta, \gamma, \delta); k_2)$ to P_j.
6. P_j now opens c' by sending (R, S, ε) to P_i.
7. P_i checks if this is consistent with c' otherwise he aborts.
8. P_i now computes $z = s + \varepsilon r$, and erases r, s.
   He also opens c_p^1 by sending C, k_1 to P_j.
9. P_j verifies the consistency of c_p^1.
   If yes, he stores $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, c, \varepsilon, c_p^2)$ and outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
   He ignores any later commitment messages with the same $(\mathsf{sid}, \mathsf{ssid})$ from P_i.

The decommit phase. Upon receiving a message $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, P_i works as follows:

1. P_i sends $(x, \alpha, \beta, \gamma, \delta, k_2, z)$ to P_j.
2. P_j computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, and outputs $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ if and only if c_p^2 is consistent and:

$$ {g_1}^z = \alpha {u_1}^{\varepsilon}, {g_2}^z = \beta {u_2}^{\varepsilon}, h^z = \gamma (e/m)^{\varepsilon}, (cd^{\omega})^z = \delta v^{\varepsilon} $$

Fig. 3. Lindell's Commitment Protocol, UC-Secure against Adaptive Adversaries

3.3 A Simple Patch

In order to avoid the above concern, a simple patch consists in committing $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ in the second Pedersen commitment c_p^2. This leads to the simple change in the protocol presented on the bottom part of Figure 4, where x is now strongly committed before the proof, and then the previous issue does not occur anymore.

4 Our Optimization of the Commitments Protocols

We now focus on much more efficient protocols, with the above modification, and additional ones. We kept the original notations, but as done in [2], we can note

The commit phase. Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ where $x \in \{0,1\}^{n-\log^2(n)}$ and $\mathsf{sid}, \mathsf{ssid} \in \{0,1\}^{\log^2(n)/4}$, $\mathcal{A}$ works as follows:

5. $\mathcal{A}$ picks $s, k_2 \xleftarrow{\$} \mathbb{Z}_p$, $\underline{D \xleftarrow{\$} \mathbb{G}}$ and computes $(\alpha, \beta, \gamma, \delta) = ({g_1}^s, {g_2}^s, h^s \underline{D}, (cd^\omega)^s)$. $\mathcal{A}$ then computes and sends $c_p^2 = \mathsf{Ped}(H_K(\alpha, \beta, \gamma, \delta); k_2)$ to P_j.
8. $\mathcal{A}$ <u>checks</u> if $G^{-1}(mD^{1/\varepsilon})$ exists and can be parsed as $(x', \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ for a random x'. If so, $\mathcal{A}$ now computes $z = s + \varepsilon r$, and erases r, s. It also opens c_p^1 by sending C, k_1 to P_j.

The decommit phase. Upon receiving a message $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, $\mathcal{A}$ works as follows:

1. $\mathcal{A}$ sends $(\underline{x'}, \alpha, \beta, \gamma, \delta, k_2, z)$ to P_j.
2. P_j computes $\underline{m'} = G(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \underline{x})$, and outputs $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, \underline{x'})$ if and only if c_p^2 is consistent and:

$$g_1^z = \alpha {u_1}^\varepsilon, {g_2}^z = \beta {u_2}^\varepsilon, h^z = \underline{h^s (e' D^{1/\varepsilon})^\varepsilon} = \gamma (e/\underline{m'})^\varepsilon, (cd^\omega)^z = \delta v^\varepsilon$$

The commit phase.

5. P_i picks $s, k_2 \xleftarrow{\$} \mathbb{Z}_p$ and computes $(\alpha, \beta, \gamma, \delta) = ({g_1}^s, {g_2}^s, h^s, (cd^\omega)^s)$. He then computes and sends $c_p^2 = \mathsf{Ped}(\underline{m}, H_K(\alpha, \beta, \gamma, \delta); k_2)$ to P_j.

Fig. 4. Inconsistent Extraction/Opening and Simple Patch w.r.t. Figure 3

that C is actually a Cramer-Shoup encryption of m, and $(\alpha, \beta, \gamma, \delta)$ is a partial Cramer-Shoup encryption of 1 with the same ω as in the first ciphertext: the double Cramer-Shoup encryption of (m, m') was denoted by $\mathsf{DCS}(m, m'; r, s) = (C_1, C_2)$, where

- C_1 is a real Cramer-Shoup encryption $C_1 = \mathsf{CS}(m; r)$ of m for a random $r \xleftarrow{\$} \mathbb{Z}_p$: $C_1 = (\mathbf{u}_1 = ({g_1}^r, {g_2}^r), e_1 = m \cdot h^r, v_1 = (cd^\omega)^r)$, where v_1 is computed afterwards with $\omega = H_K(\mathbf{u}_1, e_1)$;
- C_2 is a partial Cramer-Shoup encryption $C_2 = \mathsf{PCS}(m'; \omega, s)$ of m' for a random $s \xleftarrow{\$} \mathbb{Z}_p$ with the above ω value: $C_2 = (\mathbf{u}_2 = ({g_1}^s, {g_2}^s), e_2 = m' \cdot h^s, v_2 = (cd^\omega)^s)$, where v_2 is computed directly with the above $\omega = H_K(\mathbf{u}_1, e_1)$.

In addition, when ω is fixed, we have an homomorphic property: if $(C_1, C_2) = \mathsf{DCS}(m, m'; r, s)$, with a common ω, the component-wise product $C_1 \times C_2 = \mathsf{PCS}(m \times m'; \omega, r + s)$. In particular, we can see the last tuple $(\alpha {u_1}^\varepsilon, \beta {u_2}^\varepsilon, \gamma e^\varepsilon, \delta v^\varepsilon)$ as $C_2 \times C_1^\varepsilon$. It should thus be $\mathsf{PCS}(m^\varepsilon; \omega, \varepsilon r + s) = \mathsf{PCS}(m^\varepsilon; \omega, z)$, which is the final check. We now use these new notations in the following.

4.1 Improvement of the Static Protocol

The improvement presented below consists in noting that the receiver can directly send the value ε in the decommit phase, without having to send a commitment first. To allow this, we simply ask the sender to send a Pedersen commitment of $C_2 = (\alpha, \beta, \gamma, \delta)$ prior to receiving ε. This reduces the number

We have a CRS, consisting of $(p, \mathbb{G}, g, g_1, g_2, c, d, h, \zeta, H_K)$, where $\mathbb{G}$ is a group of order p with generators g, ζ, g_1, g_2; $c, d, h \in \mathbb{G}$ are random elements in $\mathbb{G}$; H_K is randomly drawn from a collision-resistant hash function family $\mathcal{H}$.
Intuitively $(p, \mathbb{G}, g_1, g_2, c, d, h, H_K)$ is a Cramer-Shoup public key and $(p, \mathbb{G}, g, \zeta)$ is a CRS for a Pedersen commitment.
Let $G : \{0,1\}^n \to \mathbb{G}$ be an efficiently computable and invertible mapping of a binary string to the group, as before.
The commit phase. Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ where $x \in \{0,1\}^{n-\log^2(n)}$ and $\mathsf{sid}, \mathsf{ssid} \in \{0,1\}^{\log^2(n)/4}$, party P_i works as follows:

1. P_i computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
2. P_i picks $r, s \xleftarrow{\$} \mathbb{Z}_p$ and computes $(C_1, C_2) = \mathsf{DCS}(m, 1; r, s)$.
   We note $C_2 = (\alpha, \beta, \gamma, \delta)$.
3. P_i sends $(\mathsf{sid}, \mathsf{ssid}, C_1)$ to P_j.
4. P_j stores $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, C_1)$ and outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
   He ignores any later commitment messages with the same $(\mathsf{sid}, \mathsf{ssid})$ from P_i.

The decommit phase. Upon receiving a message $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, P_i works as follows:

1. P_i picks $k_2 \xleftarrow{\$} \mathbb{Z}_p$, computes $c_p^2 = \mathsf{Ped}(H_K(m, C_2, \mathsf{sid}, \mathsf{ssid}, P_i, P_j); k_2)$.
   He then sends $(\mathsf{sid}, \mathsf{ssid}, x, c_p^2)$ to P_j.
2. P_j computes $m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, picks $\varepsilon \xleftarrow{\$} \mathbb{Z}_p$ and sends it to P_i.
3. P_i now computes $z = s + \varepsilon r$ and sends $(\mathsf{sid}, \mathsf{ssid}, C_2, k_2, z)$ to P_j.
4. P_j outputs $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ if and only if c_p^2 is consistent and
$$g_1^z = \alpha u_1{}^{\varepsilon}, g_2{}^z = \beta u_2{}^{\varepsilon}, h^z = \gamma(e/m)^{\varepsilon}, (cd^{\omega})^z = \delta v^{\varepsilon}$$

Fig. 5. Our New Commitment Protocol UC-Secure against Static Adversaries

of flows of the decommit phase (from 5 downto 3) and the number of elements sent by the receiver, (from 2 group elements and 3 scalars down to only 1 scalar, the challenge), simply increasing the number of elements sent by the sender by 1 group element and 1 scalar (the Pedersen commitment).

4.2 Sketch of Proof of the Static Protocol

For lack of space, we do not give here the full proof of the protocol. One may note that it is very similar to the one given in [15]. The only change lies in the decommit phase, where we make the receiver directly send his challenge value ε rather than encrypting it first. But this change is made possible by the sender sending a Pedersen commitment c_p^2 of C_2 before having seen ε, as in the commit phase of the adaptive version of our protocol.

The proof can thus be easily adapted from the one given for our adaptive protocol (see Section 4.4). The only difference is that in the static version, the sender does not commit to his value C_1, so that the simulator cannot change its mind on the value it gave inside this ciphertext later on. But one can note that in the proof of the adaptive protocol, this commitment c_p^1 has to be equivocated only in case of adaptive corruptions (if the latter occur before the adversary has

sent ε). This yields to the same simulator as in the adaptive case (see Section 4.5) with the following modifications, when P_i is honest only:

COMMIT STAGE: Exactly as in the adaptive case except there is no corruption to deal with.

DECOMMIT STAGE: Upon receiving the information that the decommitment has been performed on x, with $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ from $\mathcal{F}_{\mathsf{mcom}}$, $\mathcal{S}$ first chooses a random z and computes the ciphertext $C_3 = \mathsf{PCS}(m; \omega, z)$. It then chooses a random k_2, a random C_2, computes the associated Pedersen commitment c_p^2 and simulates the first flow of the decommit phase to P_j. Upon receiving ε from P_j, it then adapts $C_2 = C_3/{C_1}^{\varepsilon}$ and uses the trapdoor for the Pedersen commitment to produce a new value k_2 corresponding to the new value C_2. It then simulates the third flow of the decommit phase to P_j.

4.3 Improvement of the Adaptive Protocol

As for the static version of the protocol, the main improvement presented on Figure 6 below consists in noting that the receiver can directly send the value ε, without having to send an encryption before. To allow this, we simply ask the sender to send his two Pedersen commitments prior to receiving ε.

This reduces, in the commit phase, the number of rounds (from 5 downto 3) and the number of elements sent by the receiver (from 2 group elements and 3 scalars down to only 1 scalar, the challenge). Contrary to the static version, there is no additional cost. This is illustrated in Section 5, which sums up the differences between Lindell's protocol and ours, in the same setting: UC-security against adaptive corruption with erasures.

In addition, in order to slightly increase the message space from $n - \log_2(n)$ to n, we move the sensitive prefix $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ into the second Pedersen.

Eventually, in order to definitely exclude the security concerns presented in Section 3.2, we include the value m to the second Pedersen to prevent the adversary from trying to open his commitment to another value.

4.4 Security Proof

We now provide a full proof, with a sequence of games, that the above protocol emulates the ideal functionality against adaptive corruptions with erasures. This sequence starts from the real game, where the adversary interacts with real players, and ends with the ideal game, where we have built a simulator that makes the interface between the ideal functionality and the adversary.

As already explained, we denote by $C_3 = C_2{C_1}^{\varepsilon}$, the tuple involved in the last check. It should be a partial encryption of m under randomness $z = s + \varepsilon r$: $C_3 = \mathsf{PCS}(m; \omega, z)$ where ω is the hash value of the first three terms of C_1.

Game $\mathbf{G_0}$: This is the real game, in which every flow from the honest players is generated correctly by the simulator which knows the input x sent by the

We have a CRS, consisting of $(p, \mathbb{G}, g, g_1, g_2, c, d, h, \zeta, H_K)$, where $\mathbb{G}$ is a group of order p with generators g, ζ, g_1, g_2; $c, d, h \in \mathbb{G}$ are random elements in $\mathbb{G}$; H_K is randomly drawn from a collision-resistant hash function family $\mathcal{H}$.
Intuitively $(p, \mathbb{G}, g_1, g_2, c, d, h, H_K)$ is a Cramer-Shoup public key and $(p, \mathbb{G}, g, \zeta)$ is a CRS for a Pedersen commitment.
Let $G : \{0,1\}^n \to \mathbb{G}$ be an efficiently computable and invertible mapping of a binary string to the group, as before.

The commit phase. Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$, party P_i works as follows, where $x \in \{0,1\}^n$ and $\mathsf{sid}, \mathsf{ssid} \in \{0,1\}^{\log^2(n)/4}$:

1. P_i computes $m = G(x)$.
2. P_i picks $r, s \xleftarrow{\$} \mathbb{Z}_p$ and computes $(C_1, C_2) = \mathsf{DCS}(m, 1; r, s)$.
   We note $C_2 = (\alpha, \beta, \gamma, \delta)$.
3. P_i picks $k_1, k_2 \xleftarrow{\$} \mathbb{Z}_p$.
   He computes $c_p^1 = \mathsf{Ped}(H_K(C_1); k_1)$, $c_p^2 = \mathsf{Ped}(H_K(m, C_2, \mathsf{sid}, \mathsf{ssid}, P_i, P_j); k_2)$.
   He sends (c_p^1, c_p^2) to P_j.
4. P_j picks $\varepsilon \xleftarrow{\$} \mathbb{Z}_p$ and sends it to P_i.
5. P_i now computes $z = s + \varepsilon r$, and erases r, s.
   He also opens c_p^1 by sending (C_1, k_1) to P_j.
6. P_j verifies the consistency of c_p^1.
   If yes, he stores $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, C_1, \varepsilon, c_p^2)$ and outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.
   He ignores any later commitment messages with the same $(\mathsf{sid}, \mathsf{ssid})$ from P_i.

The decommit phase. Upon receiving a message $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$, P_i works as follows:

1. P_i sends (x, C_2, k_2, z) to P_j.
2. P_j computes $m = G(x)$, and outputs $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ if and only if c_p^2 is consistent and:
$$g_1^z = \alpha {u_1}^\varepsilon, {g_2}^z = \beta {u_2}^\varepsilon, h^z = \gamma(e/m)^\varepsilon, (cd^\omega)^z = \delta v^\varepsilon$$

Fig. 6. Our New Commitment Protocol UC-Secure against Adaptive Adversaries

environment to the sender. There is no use of the ideal functionality for the moment.

Game $\mathbf{G}_1$: In this game, we focus on the simulation of an honest receiver interacting with a corrupted sender. Executions with an honest sender are still simulated as before, using the input x. The simulator will generate the CRS in such a way it knows the Cramer-Shoup decryption key, but ζ is a discrete logarithm challenge.

Upon receiving the values (c_p^1, c_p^2) from the adversary, the simulator simply chooses a challenge ε at random and sends it to the adversary, as P_j would do with P_i. After receiving the values (C_1, k_1), the simulator checks the consistency of the Pedersen commitment c_p^1 and aborts in case of failure. It then uses the Cramer-Shoup decryption key to recover the value m' sent by the adversary, and computes $x' = G^{-1}(m')$. In case of invalid ciphertext, one sets $x' = \perp$ (an element not in the domain of G). It stores $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, C_1, \varepsilon, c_p^2)$ and $(x', \mathsf{sid}, P_i, P_j)$ (this will correspond later to the Commit query to the ideal

functionality, in the ideal game). Upon receiving the values (x, C_2, k_2, z), the simulator does as P_j would do in checking the commitment c_p^2 and that $C_3 = \mathsf{PCS}(m^\varepsilon; \omega, z)$, but accepts x' as the opening for the commitment.

The only difference with the previous game is that P_i will accept x', as decrypted from $C_1 = \mathsf{CS}(m' = G(x'); r)$, for the decommitment instead of the value x output at the decommitment time, which leads to $m = G(x)$ that matches with $C_3 = \mathsf{PCS}(m^\varepsilon; \omega, z)$, but that is also contained in c_p^2 together with C_2. We will show that under the binding property of the Pedersen commitment, one necessarily has $x' = x$, and thus there is no difference.

Let us assume that $x' \neq x$ in at least one of such executions: for the first one, we rewind the adversary up to the step 4., and send a new random challenge ε'. Then the adversary should send the same C_1, otherwise one extracts the discrete logarithm of ζ in basis g or a collision for H_K, and the same pair (m, C_2) in the decommit phase for the same reason, but possibly a different z'. Then, the final checks guarantee that $C_3 = \mathsf{PCS}(m^\varepsilon; \omega, z)$ in the first execution and $C_3' = \mathsf{PCS}(m^{\varepsilon'}; \omega, z')$ in the second execution. From the homomorphic property: C_2 encrypts $(m/m')^\varepsilon$ in the first execution, but $(m/m')^{\varepsilon'}$ in the second execution, which are thus equal. Since $\varepsilon' \neq \varepsilon$, this implies that $m' = m$. For the same reason, one can note that if C_1 is not a valid ciphertext, C_3 cannot be valid either (for the fixed ω). We stress that the rewind here is just for the proof of indistinguishability of the two games, but not in the simulation.

In case of corruption of the receiver, one can note that he has no secret.

Game $\mathbf{G_2}$: In this game, we start modifying the simulation of an honest sender, still knowing his input x. For the honest verifier against a corrupted sender, we still have to know the Cramer-Shoup decryption key to run the same simulation as in the previous game. But we now need to know the discrete logarithm for equivocating the Pedersen commitment.

This game is almost the same as the previous one excepted the way the double Cramer-Shoup ciphertext is generated: $(C_1, C_2) = \mathsf{DCS}(m, n; r, s)$, for a random n instead of 1. The rest of the commit phase is unchanged.

At the decommit phase, $\mathcal{S}$ chooses random coins z and computes $C_3 = \mathsf{PCS}(m; \omega, z)$, and then "repairs" $C_2 = C_3/{C_1}^\varepsilon$, and k_2 for being able to open c_p^2 to this new value.

Thanks to the homomorphic property, the repaired C_2 is a correct ciphertext of 1, and the equivocation of the Pedersen commitment guarantees a correct opening. This game is thus perfectly indistinguishable from the previous one.

In case a corruption of P_i occurs before the decommit phase, the simulator anticipates the equivocation of c_p^2.

Game $\mathbf{G_3}$: One can note that in the previous game, r is not used anymore to compute z. One could thus ignore it, unless $\mathcal{P}_i$ gets corrupted before ε has been sent, since we should be able to give it. But in such a case, one can compute again C_1 knowing r and equivocate c_p^1. We then alter again the way the double Cramer-Shoup ciphertext is generated: $(C_1, C_2) = \mathsf{DCS}(m', n; r, s)$, for random m' and n. Everything remains unchanged.

The unique change is thus the ciphertext C_1 that encrypts a random m' instead of m. One can run the IND-CCA security game, in an hybrid way, to show this game is indistinguishable from the previous one. To this aim, one has to show that the random coins r are not needed to be known, and that the challenge ciphertexts are never asked for decryption (where the decryption key here is replaced by an access to the decryption oracle, hence the IND-CCA security game). The former point has been discussed above. For the latter, we have shown that the value actually encrypted in C_1 by the corrupted sender is the value sent at the decommit phase, which would even break the one-wayness of the encryption. Hence, if such a replay happens, one knows that the decommit phase will fail.

In case of corruption of P_i before receiving ε, Pedersen commitments only have been sent, and they can thus be equivocated with correct values (given by either the ideal functionality or the adversary). In case of corruption of P_i after having received ε, one does has before, anticipating the equivocation of c_p^2.

Game $\mathbf{G}_4$: This is the ideal game, in which the simulator works as described below: when P_i is corrupted, one uses the decryption of C_1 to send the Commit query to the ideal functionality, when P_i is honest one can wait for the receipt and reveal confirmations from the adversary to conclude the simulation of the real flows.

4.5 Description of the Simulator

Setup. The simulator generates the parameters, knowing the Cramer-Shoup decryption key and the Pedersen equivocation trapdoor.

When P_i is Honest.

Commit stage: Upon receiving the information that a commitment has been performed, with $(\mathsf{receipt}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ from $\mathcal{F}_{\mathsf{mcom}}$, $\mathcal{S}$ computes $(C_1, C_2) = \mathsf{DCS}(m', n; r, s)$, for random m' and n but then follows as P_i would do. If P_j is honest too, one just has to send a random ε.

In case of corruption of P_i before receiving ε, one can equivocate c_p^1, otherwise one equivocates c_p^2, as explained below, in both cases using the value given either by the ideal functionality or the adversary, according to the time of the corruption.

Decommit stage: Upon receiving the information that the decommitment has been performed on x, with $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$ from $\mathcal{F}_{\mathsf{mcom}}$, $\mathcal{S}$ exploits the equivocability of the Pedersen commitment: it first chooses a random z and computes the ciphertext $C_3 = \mathsf{PCS}(m = G(x); \omega, z)$. It then adapts $C_2 = C_3/{C_1}^{\varepsilon}$ and uses the trapdoor for the Pedersen commitment to produce a new value k_2 corresponding to the new value C_2. It then simulates the decommit phase to P_j.

When P_i is Corrupted and P_j is Honest.

COMMIT STAGE: Upon receiving (C_1, k_1) from the adversary, $\mathcal{S}$ decrypts the Cramer-Shoup ciphertext C_1 and extracts x from G. If the decryption is invalid, $\mathcal{S}$ sends $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, \perp)$ to $\mathcal{F}_{\mathsf{mcom}}$. Otherwise, $\mathcal{S}$ sends $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, x)$.

DECOMMIT STAGE: $\mathcal{S}$ acts as a regular honest user P_j from the incoming message of $\mathcal{A}$ on behalf of P_i. In case of validity, send the query $(\mathsf{reveal}, \mathsf{sid}, \mathsf{ssid})$.

5 Conclusion

As a conclusion, let us graphically present a comparison of the two protocols.

5.1 The Original Lindell's Protocol for Adaptive Adversaries

The commit phase

$$
\begin{array}{rl}
m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j) & \\
r \xleftarrow{\$} \mathbb{Z}_p, k_1 \xleftarrow{\$} \mathbb{Z}_p, C = \mathsf{CS}(m; r) & \\
c_p^1 = \mathsf{Ped}(H_K(C); k_1) & \xrightarrow{c_p^1} R, S \xleftarrow{\$} \mathbb{Z}_p, \varepsilon \xleftarrow{\$} \{0,1\}^n \\
s \xleftarrow{\$} \mathbb{Z}_p, k_2 \xleftarrow{\$} \mathbb{Z}_p & \xleftarrow{c'} c' = ({g_1}^R {g_2}^S, {h_1}^R {h_2}^S G(\varepsilon)) \\
(\alpha, \beta, \gamma, \delta) = ({g_1}^s, {g_2}^s, h^s, (cd^\omega)^s) & \\
c_p^2 = \mathsf{Ped}(H_K(\alpha, \beta, \gamma, \delta)); k_2) & \xrightarrow{c_p^2} \\
\text{Aborts if } c' \text{ inconsistent} & \xleftarrow{R, S, \varepsilon} \\
z = s + \varepsilon r, \text{ erases } r, s & \xrightarrow{k_1, C} \text{Aborts if } c_p^1 \text{ inconsistent}
\end{array}
$$

The decommit phase

$$
\begin{array}{rl}
\xrightarrow{(x, \alpha, \beta, \gamma, \delta, k_2, z)} & m = G(x, \mathsf{sid}, \mathsf{ssid}, P_i, P_j) \text{ checks } c_p^2 \text{ and whether} \\
& g_1^z = \alpha {u_1}^\varepsilon, {g_2}^z = \beta {u_2}^\varepsilon, h^z = \gamma (e/m)^\varepsilon, (cd^\omega)^z = \delta v^\varepsilon
\end{array}
$$

5.2 Our Protocol

The commit phase

$$
\begin{array}{rl}
m = G(x), r \xleftarrow{\$} \mathbb{Z}_p, s \xleftarrow{\$} \mathbb{Z}_p & \\
(C_1, C_2) = \mathsf{DCS}(m; 1; r, s), k_1, k_2 \xleftarrow{\$} \mathbb{Z}_p & \\
c_p^1 = \mathsf{Ped}(H_K(C_1); k_1) & \\
c_p^2 = \mathsf{Ped}(H_K(C_2, m, \mathsf{sid}, \mathsf{ssid}, P_i, P_j); k_2) & \xrightarrow{c_p^1, c_p^2} \\
& \xleftarrow{\varepsilon} \varepsilon \xleftarrow{\$} \mathbb{Z}_p \\
z = s + \varepsilon r, \text{ erases } r, s & \xrightarrow{k_1, C_1} \text{Aborts if } c_p^1 \text{ inconsistent}
\end{array}
$$

The decommit phase

$$
\begin{array}{rl}
\xrightarrow{(x, C_2, k_2, z)} & m = G(x), \text{ checks } c_p^2 \text{ and whether} \\
& g_1^z = \alpha {u_1}^\varepsilon, {g_2}^z = \beta {u_2}^\varepsilon, h^z = \gamma (e/m)^\varepsilon, (cd^\omega)^z = \delta v^\varepsilon
\end{array}
$$

Acknowledgments. We thank Yehuda Lindell for his fruitful comments. The first author was funded by a Sofja Kovalevskaja Award of the Alexander von Humboldt Foundation and the German Federal Ministry for Education and Research. This work was supported in part by the French ANR-12-INSE-0014 SIMPATIC Project and in part by the European Commission through the FP7-ICT-2011-EU-Brazil Program under Contract 288349 SecFuNet.

References

1. Abdalla, M., Chevalier, C., Pointcheval, D.: Smooth projective hashing for conditionally extractable commitments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 671–689. Springer, Heidelberg (2009)
2. Ben Hamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: Efficient UC-secure authenticated key-exchange for algebraic languages. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 272–291. Springer, Heidelberg (2013)
3. Blazy, O., Fuchsbauer, G., Izabachène, M., Jambert, A., Sibert, H., Vergnaud, D.: Batch Groth–Sahai. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 218–235. Springer, Heidelberg (2010)
4. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (October 2001)
6. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. On the Cryptology ePrint Archive, Report 2000/067 (December 2005)
7. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
8. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005)
9. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC, pp. 494–503. ACM Press (May 2002)
10. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
11. Damgård, I., Nielsen, J.B.: Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 581–596. Springer, Heidelberg (2002)
12. Fischlin, M., Libert, B., Manulis, M.: Non-interactive and re-usable universally composable string commitments with adaptive security. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 468–485. Springer, Heidelberg (2011)
13. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer, Heidelberg (2003)

14. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
15. Lindell, Y.: Highly-efficient universally-composable commitments based on the DDH assumption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 446–466. Springer, Heidelberg (2011)
16. Lindell, Y.: Highly-efficient universally-composable commitments based on the DDH assumption. Cryptology ePrint Archive, Report 2011/180 (2011)
17. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
18. Yao, A.C.: Theory and applications of trapdoor functions. In: 23rd FOCS, pp. 80–91. IEEE Computer Society Press (November 1982)