Highly-Efficient Universally-Composable Commitments based on the DDH Assumption*

Yehuda Lindell

Department of Computer Science
Bar-Ilang University, Israel
lindell@cs.biu.ac.il

Abstract. Universal composability (a.k.a. UC security) provides very strong security guarantees for protocols that run in complex real-world environments. In particular, security is guaranteed to hold when the protocol is run concurrently many times with other secure and possibly insecure protocols. Commitment schemes are a basic building block in many cryptographic constructions, and as such universally composable commitments are of great importance in constructing UC-secure protocols. In this paper, we construct highly efficient UC-secure commitments from the standard DDH assumption, in the common reference string model. Our commitment stage is non-interactive, has a common reference string with $O(1)$ group elements, and has complexity of $O(1)$ exponentiations for committing to a group element (to be more exact, the effective cost is that of $23\frac{1}{3}$ exponentiations overall, for both the commit and decommit stages). We present a construction that is secure in the presence of static adversaries, and a construction that is secure in the presence of adaptive adversaries with erasures, where the latter construction has an effective additional cost of just $5\frac{1}{3}$ exponentiations.

1 Introduction

Background – universal composability and efficiency. Modern cryptographic protocols are run in complex environments. Many different secure and insecure protocols are executed concurrently, and some protocols may have been designed specifically to attack others [13]. The classic definitions of security that consider stand-alone executions only do not guarantee security in modern real-world setting. Universal composability (or UC security) is a definitional framework that guarantees security even if the protocol is run concurrently with arbitrarily many other secure and insecure protocols, and even if related inputs are used. More specifically, a UC-secure protocol behaves like an ideal execution (where an incorruptible trusted party carries out the computation for the parties) no matter what other protocols are being run by the honest parties at the time.

* This research was supported by the European Research Council as part of the ERC project "LAST", and by the israel science foundation (grant No. 781/07).

The UC-framework models the *real-world* execution environment in a far more realistic way than the classic stand-alone definitions. As such, one would expect the framework to be adopted by practitioners and those interested in implementing cryptographic protocols that could be run in practice. In the setting of key exchange this is indeed the case. For just two examples, the SIGMA family of key exchange protocols that are part of IKE (the standardized Internet key exchange protocol) and the HMQV protocol have been proven secure in the UC-framework [5, 15]. However, beyond key exchange, there seems to have been little interest in UC-security from the applied cryptographic community. (We stress that this is in contrast to the recent growing interest in implementations of general and specific protocols for secure two-party and multiparty computation; see [17, 2, 23, 20, 19] for just a few examples.) There are a number of reasons for this. We believe that one of the primary reasons is the lack of *efficient* UC-secure primitives, the exception being UC-secure oblivious transfer [22]. Given this state of affairs, it is very difficult to construct efficient UC-secure protocols that can be reasonably implemented.

UC commitments. Commitment schemes are one of the most basic building blocks for cryptographic protocols. A commitment scheme is made up of two phases: a *commit phase* in which a committer commits to a value while keeping it hidden, and a *decommit phase* in which the committer reveals the value that it previously committed to. The binding property of a commitment states that the committer is bound to a single value after the commit phase and can only decommit to that value; the hiding property states that the receiver learns nothing about the committed value until it is revealed. As such, a commitment scheme has been intuitively described as a digital envelope containing the committed value: once the envelope has been closed the committer cannot change the value, and until the envelope is opened the receiver cannot learn what is inside. Despite this appealing description, regular commitments do not behave in this way. For just one example, they may be malleable (e.g., it may be possible to generate a commitment to $2x$ from a commitment to x, without knowing x). In contrast, UC-secure commitments are non-malleable, cannot be copied, and are guaranteed to remain secure irrespective of whatever other protocols are run.

Commitment schemes that are secure in the UC-framework were first presented by Canetti and Fischlin in [4]. They also showed that it is impossible to construct UC commitments in the plain model, and thus some setup like a common reference string is required. The commitment schemes of [4] have the property that $O(1)$ asymmetric operations are needed for every bit committed to. Soon after, Damgård and Nielsen [9] presented UC commitments with the property that $O(1)$ exponentiations are sufficient for committing to an entire string (that can be mapped into a group element). This is a significant improvement. However, the Damgård-Nielsen construction suffers from a few drawbacks. First, it requires a common reference string that grows linearly with the number of parties in the system; specifically, one group element is needed for every party. This is a significant obstacle in implementations because it means that it is not possible to publish a single common reference string that can then be used

by arbitrary parties who wish to run the protocol. Second, the Damgård-Nielsen constructions are based on the N-residuosity and p-subgroup assumptions. These are less established assumptions than RSA and DDH, for example. Furthermore the N-residuosity assumption, which has become accepted since its introduction in [21], suffers from a significant computational overhead. This is due to the fact that exponentiations are modulo N^2 (at least) and thus a modulus N of size 1536 – which is needed for reasonable security – results in exponentiations modulo a number of length *3072 bits*. In contrast, basic discrete log exponentiations can be run in Elliptic curves of size 224 or 256 bits and are significantly faster. In cryptographic protocols where many UC commitments are needed (see below for an example), this can be a real obstacle.

Our results. We present a conceptually simple and efficient protocol for UC-secure commitments in the common reference string model that is based on the DDH assumption. Our protocol requires $O(1)$ regular group exponentiations and has a common reference string with $O(1)$ group elements for any number of parties. In addition, we have a version that provides security in the presence of adaptive adversaries with erasures that has only slightly additional cost. A comparison of our result with the construction of Damgård-Nielsen, which is the previous best known, yields the following:

- *Assumptions:* We rely on the standard DDH assumption, while Damgård-Nielsen rely on the N-residuosity or p-subgroup assumptions.
- *Common reference string (CRS):* Our common reference string contains a description of the discrete log group, its order and 7 group elements, and can be used by *any number of parties*. Thus, a single CRS can be published for all to use. In contrast, Damgård-Nielsen need a CRS with a single (ring or group) element for every party in the system.
- *Efficiency:* Our protocol has a non-interactive commitment phase with just 5 exponentiations to be computed by the committer. The decommit phase is interactive and requires both parties overall to compute 21 exponentiations. Using optimizations for computing multi-exponentiations of the form $g^r \cdot h^s$ the overall cost in both phases is $23\frac{1}{3}$ regular DDH exponentiations. In contrast Damgård-Nielsen have an interactive commitment phase with 10 large modulus exponentiations and a non-interactive decommit phase requiring 4 exponentiations.[1] Based on experiments, we estimate that our commitment scheme is approximately 25–30 times faster than the scheme of Damgård-Nielsen. (This estimate is not based on an implementation of the schemes, but rather a comparison of the time taken to compute 14 exponentiations $\mathrm{mod} N^2$ with a modulus N of size 2048 bits versus $23\frac{1}{3}$ Elliptic curve "exponentiations" over a field of size 256 bits, using the Crypto++ library [25]. When using a modulus N of size 1536 bits versus a curve over a field of size 224 bits, our scheme is approximately 20 times faster.)

[1] The question of whether there is more cost in the commitment or decommitment phase is significant in protocols of the cut-and-choose type where many commitments are sent and only some of them opened. In such cases, it is preferable to use a commitment scheme with a faster commitment phase.

– *Adaptive security:* The Damgård-Nielsen construction is secure only for static corruptions (where the set of corrupted parties is fixed before any commitments are sent), whereas we also have a construction that is secure in the presence of adaptive corruptions with erasures. (In this model, the adversary can choose to corrupt parties over time, but an honest party can follow erase instructions and it is assumed that once data is erased it cannot be retrieved by the adversary if it later corrupts the party.) The additional cost of achieving adaptive security is just $5\frac{1}{3}$ exponentiations, yielding a total of $28\frac{2}{3}$. In this case, however, the majority of the work is in the commitment stage and not in the decommitment stage.

An example – UC zero-knowledge from Sigma-protocols. Since our efficiency improvement over prior work is *concrete* and not asymptotic, we demonstrate its potential significance in implementations. We do this by considering the ramification of our efficiency improvement on constructions of efficient UC-secure zero-knowledge protocols. Many, if not most, useful efficient zero-knowledge protocols are based on Sigma protocols [8]. In the stand-alone case, transformations from Sigma protocols to zero-knowledge and zero-knowledge proofs of knowledge are highly efficient, requiring only a few additional exponentiations; see [10, Section 6.5]. Unfortunately, no such efficient analogue is known for achieving UC zero-knowledge from Sigma protocols. Rather, it is necessary to repeat the Sigma protocol L times in order to achieve a soundness error of 2^{-L}. In addition, 3 UC-commitments are needed for each repetition (but only two are opened); see [12] and [16, App. C] for a description of the transformation. Setting $L = 40$ for a reasonable soundness error, we have that 120 UC commitments are needed for the transformation. Assuming 47 milliseconds for our scheme and 1.35 seconds for Damgård Nielsen (based on estimates using the Crypto++ library), we have that the additional overhead resulting from the UC commitments is 5.6 seconds for our protocol versus 162 seconds for Damgård-Nielsen (the difference is actually even greater since 40 of the 120 commitments are not opened; see Footnote 1). We conclude that in *protocol implementations* the efficiency improvement gained by using our new UC commitment protocol can be definitive.

2 Preliminaries and Definitions

Universal composability [3] is a definition of security that considers a stand-alone execution of a protocol in a special setting involving an environment machine $\mathcal{Z}$, in addition to the honest parties and adversary. As with the classic definition of secure computation, ideal and real models are considered where a trusted party carries out the computation in the ideal model and the real protocol is run in the real model. The environment adaptively chooses the inputs for the honest parties, interacts with the adversary throughout the computation, and receives the honest parties' outputs. Security is formulated by requiring the existence of an ideal-model simulator $\mathcal{S}$ so that no environment $\mathcal{Z}$ can distinguish between the case that it runs with the real adversary $\mathcal{A}$ in the real model and the case that it runs with the ideal-model simulator $\mathcal{S}$ in the ideal model.

In slightly more detail, we denote by $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(n,z)$ the output of the environment $\mathcal{Z}$ with input z after an ideal execution with the ideal adversary (simulator) $\mathcal{S}$ and functionality $\mathcal{F}$, with security parameter n. We will only consider black-box simulators $\mathcal{S}$, and so we denote the simulator by $\mathcal{S}^{\mathcal{A}}$ meaning that it works with the adversary $\mathcal{A}$ attacking the real protocol. Furthermore, we denote by $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)$ the output of environment $\mathcal{Z}$ with input z after a real execution of the protocol π with adversary $\mathcal{A}$, with security parameter n. Our protocols are in the common reference string (CRS) model. Formally, this means that the protocol π is run in a hybrid model where the parties have access to an ideal functionality $\mathcal{F}_{\text{CRS}}$ that chooses a CRS according to the prescribed distribution and hands it to any party that requests it. We denote an execution of π in such a model by $\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)$. Informally, a protocol π UC-realizes a functionality $\mathcal{F}$ in the $\mathcal{F}_{\text{CRS}}$ hybrid model if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every non-uniform probabilistic polynomial-time environment $\mathcal{Z}$ and every probabilistic polynomial-time adversary $\mathcal{A}$, it holds that

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(n,z)\right\}_{n\in\mathbb{N};z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathbb{N};z\in\{0,1\}^*}.$$

The importance of this definition is a composition theorem that states that any protocol that is universally composable is secure when run concurrently with many other arbitrary protocols; see [3] for definitions and details.

UC commitments. The multi-commitment ideal functionality $\mathcal{F}_{\text{MCOM}}$, which is the functionality that we UC realize in this paper, is formally defined in Figure 1.

FIGURE 1 (Functionality $\mathcal{F}_{\text{MCOM}}$)

$\mathcal{F}_{\text{MCOM}}$ proceeds as follows, running with parties $P_1,\ldots,P_m$, a parameter 1^n, and an adversary $\mathcal{S}$:

- **Commit phase:** Upon receiving a message $(\mathsf{commit}, sid, ssid, P_i, P_j, x)$ from P_i where $x \in \{0,1\}^{n-\log^2 n}$, record the tuple $(ssid, P_i, P_j, x)$ and send the messages $(\mathsf{receipt}, sid, ssid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any future commit messages with the same $ssid$ from P_i to P_j.
- **Reveal phase:** Upon receiving a message $(\mathsf{reveal}, sid, ssid)$ from P_i: If a tuple $(ssid, P_i, P_j, x)$ was previously recorded, then send the message $(\mathsf{reveal}, sid, ssid, P_i, P_j, x)$ to P_j and $\mathcal{S}$. Otherwise, ignore.

The ideal commitment functionality

For technical reasons, the length of the committed value x is $n - \log^2 n$. It is defined in this way because our commitment involves encrypting x together with the session identifiers $sid, ssid$ and the parties' identities (i,j). Now, the encryption that we use is of a single group element that is of length n, and so the combined length of $x, sid, ssid, i, j$ must be n. We therefore define each identifier and identity to be of size $\frac{\log^2 n}{4}$; this means that each comes from a superpolynomial domain and so there are enough to ensure that the session identifiers do not repeat and each party has a unique identity. Thus, taking x of size $n - \log^2 n$ we have that the string $(x, sid, ssid, i, j)$ is of length n.

3 Efficient UC Commitments

3.1 Protocol Idea and Overview

Before describing the idea behind our construction, recall that a UC-secure commitment must be both *extractable* (meaning that a simulator can extract the value that a corrupted party commits to) and *equivocal* (meaning that a simulator can generate commitments that can be opened to any value), without the simulator rewinding the adversary. In addition, the adversary must not be able to generate commitments that are related to commitments generated by honest parties; thus, the commitment must be essentially non-malleable. Our protocol is in the common reference string (CRS) model; this is justified by the fact that UC commitments cannot be achieved in the plain model [4].

The high-level idea behind our construction is as follows. The committer commits to a string by encrypting it with a CCA2-secure encryption scheme E^{cca}, using a public-key pk_1 that is found in the common reference string (CRS). Observe that this enables extraction because when simulating a protocol that is in the CRS model, the simulator is allowed to choose the CRS itself. Thus, it can choose the public key so that it knows the corresponding private decryption key. This enables it to decrypt and obtain the committed value. Next, in order to decommit, it is clearly not possible to reveal the value and randomness used to encrypt, because encryptions are perfectly binding and so it is not possible to equivocate. Thus, in order to decommit, the committer instead sends the committed value and then *proves* in zero knowledge that this is indeed the correct value. At first sight, this approach may seem futile because in the UC setting it seems no easier to construct UC zero-knowledge than UC commitments. Nevertheless, we observe that the proof need not be a full fledged UC zero-knowledge protocol, and in particular there is no need to extract the witness from the proof. Rather, the only property that we need is that it be possible to simulate without rewinding. This is due to the fact that the extraction of the committed value already took place in the commit stage and this proof is just to ensure that corrupted parties decommit to the same value that they committed to. Thus, only soundness is necessary. (Of course, the ability for a simulator to equivocate is due to its ability to run a zero-knowledge simulator and essentially lie about the value committed to.) The proof that we use is based on a Sigma protocol and we make it zero knowledge (without rewinding) by having the verifier first commit to its challenge and then run the Sigma protocol with the verifier decommitting. In order to have a straight-line simulator we make this commitment from the verifier be an encryption of the challenge under a different public key pk_2 in the CRS. As above, in the simulation the simulator can choose the public-key so that it knows the corresponding private key, enabling it to extract the challenge from the verifier. Once it has extracted the challenge, it can run the simulator for the Sigma protocol which is perfect and straight line once given the verifier challenge. Although intuitively appealing, this is problematic because soundness of this transformation from a Sigma protocol to a zero-knowledge proof can only be proven if the commitment is *perfectly hiding*. But this then clashes with the

requirement to have the commitment be extractable. We solve this efficiently by using a *dual mode* cryptosystem E^{dual}, as introduced by [22],[2] although we only need a simpler version. Such a cryptosystem has a regular key generation algorithm and an alternative one, and has the property that it behaves as a regular public-key encryption scheme when a regular key is generated, but perfectly hides the encrypted value when an alternative key is generated. Furthermore, the regular and alternative keys are indistinguishable. As we will see in the proof, this suffices for proving soundness, because at the point where soundness is needed we no longer need to be able to extract the verifier's challenge and thus can replace the key in the common reference string by an alternative one. Note that a regular key for E^{dual} is used in a real protocol execution, and the ability to generate an alternative key is used within the proof of security. (Note also that we cannot use this method for the actual UC commitment because we need to *simultaneously* extract and equivocate.)

The above yields the following template for UC commitments:

PROTOCOL 1 (UC-commitment template)

Common reference string: (pk_1, pk_2) where pk_1 is the public-key of a CCA2-secure encryption scheme, and pk_2 is the public-key of a dual mode cryptosystem, as described above.

The commit phase:

1. The committer commits to x by encrypting it under pk_1 and sending the ciphertext $c = E^{cca}_{pk_1}(x; r)$ to the receiver (i.e., it encrypts x using random coins r).
   (Actually, x is encrypted together with a unique session identifier and the identities of the parties, but we ignore these details here.)

The decommitment phase:

1. The committer sends x to the receiver (without revealing r)
2. Let (α, ε, z) denote the message of a Sigma protocol for proving that c is an encryption of x (using witness r).
   (a) The receiver sends $c' = E^{dual}_{pk_2}(\varepsilon; r')$
   (b) The committer sends α
   (c) The receiver decommits to ε by sending ε and r'
   (d) The committer checks that $c' = E^{dual}_{pk_2}(\varepsilon; r')$ and if yes, computes the reply z for the Sigma protocol, based on (α, ε)
   (e) The receiver outputs x as the decommitted value if and only if (α, ε, z) is an accepting Sigma-protocol transcript

Before proceeding, we explain why the value x is committed to by encrypting it under an encryption scheme that is secure under adaptive chosen-ciphertext attacks (CCA2 secure). Specifically, we have already discussed why some notion of non-malleability is needed, but CCA2-security is stronger than NM-CPA

[2] We use the formulation as it appears in [22], although the idea of having alternative keys that provide perfect hiding or regular encryption goes back earlier. Two examples of where similar notions were defined are in [11, 14]. In fact, our construction of dual encryption is exactly the same as the ambiguous commitment used in [11].

(non-malleability under chosen plaintext attacks). In order to understand why we nevertheless need CCA2 security, recall that a simulator must equivocate. Specifically, in the simulation in the ideal model, the simulator receives commitment receipts that contain no information about the committed value. However, in the real world, the adversary receives encryptions of the actual committed value. Thus, whenever it receives a commitment receipt, the simulator encrypts 0 and hands it to the real-world adversary. Later, when the commitment is opened and the simulator learns that it was to a value x, it cheats in the Sigma protocol and "proves" that the encryption of 0 was actually an encryption of x. In order to prove that encrypting 0 (as the simulator does) and encrypting x (as an honest party does) makes no difference, it is necessary to reduce this to the security of the encryption scheme. In such a reduction, an adversary attacking the encryption scheme simulates the UC commitment execution such that if it received encryptions of 0 then the result should be the same as the ideal simulation, and if it received encryptions of real values x then the result should be the same as a real execution with honest parties and the real adversary. To be more exact, this reduction is carried out by running the simulator for the UC commitment scheme and using challenge ciphertexts obtained in the encryption game instead of the simulator generating commitments itself. Of course, in this reduction the simulator does not choose the CCA2-secure public key to place in the CRS but rather places the public-key that it receives as part of the encryption distinguishing game. However, as we have already discussed, the simulator must also be able to *extract* committed values generated by the adversary by decrypting, at the same time as we carry out this reduction. This brings us to the crux of the problem which is that it can only carry out this decryption because it knows the private key, and so it cannot decrypt when proving the reduction. This problem is solved by using CCA2-secure encryption because now in the distinguishing game the adversary is allowed to ask for decryptions of ciphertexts, and so the simulator can decrypt the commitments from the adversary, as required.

Efficient implementations. It remains to describe how all of the elements of the protocol can be efficiently implemented. First, we use the Cramer-Shoup (CS) encryption scheme [7] as the CCA2-secure encryption scheme. This scheme is defined as follows:

- **CS key generation:** Let $(\mathbb{G}, q, g_1, g_2)$ be such that $\mathbb{G}$ is a group of order q and g_1, g_2 are two distinct generators. Choose $x_1, x_2, y_1, y_2, z \in_R \mathbb{Z}_q$ at random and compute $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$ and $h = g_1^z$. The public key is $(\mathbb{G}, q, g_1, g_2, c, d, h)$ and the secret key is (x_1, x_2, y_1, y_2, z).
- **CS encryption:** Let $m \in G$. Then, in order to encrypt m, choose a random $r \in_R \mathbb{Z}_q$, compute $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r \cdot m$, $\omega = H(u_1, u_2, e)$ where H is a collision-resistant hash function, and $v = (c \cdot d^\omega)^r$. The ciphertext is (u_1, u_2, e, v).
- **CS decryption:** Compute $\omega = H(u_1, u_2, e)$. If $u_1^{x_1} \cdot u_2^{x_2} \cdot (u_1^{y_1} \cdot u_2^{y_2})^\omega = v$, then output $m = e/(u_1^z)$.

The crucial observation that we make is that in order to verify that a ciphertext (u_1, u_2, e, v) is a valid encryption of a message m, it suffices to prove that there

exists a value $r \in \mathbb{Z}_q$ such that

$$u_1 = g_1^r, \quad u_2 = g_2^r, \quad \frac{e}{m} = h^r, \quad \text{and} \quad v = (cd^\omega)^r.$$

Furthermore, since ω can be computed publicly from the public-key and ciphertext, all the values except for r are public. Thus, we have that in order to prove that a ciphertext encrypts some given value m, we just need to run a proof that 4 values have the same discrete log with respect to their respective bases. Highly efficient Sigma protocols exist for this task (this is the same as proving that a tuple is of the Diffie-Hellman form). Thus, the CCA2-secure encryption scheme together with the required proof can both be implemented very efficiently.

It remains to show how a dual-model encryption scheme can be efficiently implemented. We essentially use the construction of [22], but we need only their basic cryptosystem and not their full dual-mode one. Specifically, we need the ability to construct a fake public-key that is indistinguishable from a regular one, so that if encryption is carried out under this key, then the encrypted value is perfectly hidden. Such an encryption scheme can be constructed at double the cost of El Gamal as follows:

- **Dual regular key generation:** Let $(\mathbb{G}, q, g_1, g_2)$ be as above. Choose $\rho \in_R \mathbb{Z}_q$ and compute $h_1 = g_1^\rho$ and $h_2 = g_2^\rho$. The public key is $(\mathbb{G}, q, g_1, g_2, h_1, h_2)$, and the private key is ρ.
- **Dual alternative key generation:** As above, except choose $\rho_1, \rho_2 \in_R \mathbb{Z}_q$ with $\rho_1 \neq \rho_2$ and compute $h_1 = g_1^{\rho_1}$ and $h_2 = g_2^{\rho_2}$.
- **Dual encryption:** To encrypt $m \in G$, choose random $R, S \in \mathbb{Z}_q$ and compute $u = g_1^R \cdot g_2^S$ and $v = h_1^R \cdot h_2^S \cdot m$. The ciphertext is $c = (u, v)$.
- **Dual decryption:** To decrypt (u, v), compute $m = v/u^\rho$.

In order to see that this scheme has the desired properties, observe that decryption works as in El Gamal, an alternative key is indistinguishable from a real one by the DDH assumption, and encryption under an alternative key is perfectly hiding since when $\rho_1 \neq \rho_2$ the values u and v are independent.

Naively, the cost of encryption is 4 exponentiations which is twice that of El Gamal. However, using the method of simultaneous multiple exponentiations in [18, Sec. 14.6], we have that u and v can be computed at the equivalent cost of $1\frac{1}{3}$ exponentiations each. Thus, the cost is $2\frac{2}{3}$ exponentiations.

3.2 The Actual Protocol

The full specification of our commitment scheme appears in Protocol 2. The proof carried out in the decommitment phase is based on a Sigma protocol for Diffie-Hellman tuples. Regarding completeness of this proof, observe that if P_i is honest, then $g_1^z = g_1^{s+\varepsilon r} = g_1^s \cdot (g_1^r)^\varepsilon = \alpha \cdot u_1^\varepsilon$, $g_2^z = g_2^{s+\varepsilon r} = g_2^s \cdot (g_2^r)^\varepsilon = \beta \cdot u_2^\varepsilon$, $h^z = h^{s+\varepsilon r} = h^s \cdot (h^r)^\varepsilon = \gamma \cdot \left(\frac{e}{m}\right)^\varepsilon$, and

$$(cd^\omega)^z = (cd^\omega)^{s+\varepsilon r} = (cd^\omega)^s \cdot ((cd^\omega)^r)^\varepsilon = (cd^\omega)^s \cdot (c^r d^{r\omega})^\varepsilon = \delta \cdot v^\varepsilon.$$

PROTOCOL 2 (UC-Secure Commitment Protocol)

Common reference string: $(\mathbb{G}, q, g_1, g_2, c, d, h, h_1, h_2)$ where $\mathbb{G}$ is a group of order q with generators g_1, g_2, and $c, d, h \in_R \mathbb{G}$ are random elements of $\mathbb{G}$, and $h_1 = g_1^\rho$, $h_2 = g_2^\rho$ for a random $\rho \in_R \mathbb{Z}_q$. (Note that $(\mathbb{G}, q, g_1, g_2, c, d, h)$ is a Cramer-Shoup public key, and $(\mathbb{G}, q, g_1, g_2, h_1, h_2)$ is the regular public key of a dual-mode encryption scheme.)
Let $G(y)$ be a mapping of a string $y \in \{0,1\}^n$ to $\mathbb{G}$, and assume that G^{-1} is also efficiently computable.

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

1. P_i computes $m = G(x, sid, ssid, i, j)$. (The identities i, j can be mapped to $\{0,1\}^{\log^2 n/4}$ and so overall $(x, sid, ssid, i, j)$ is an n-bit string.)
2. P_i chooses a random $r \in_R \mathbb{Z}_q$, computes $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r \cdot m$, $\omega = H(u_1, u_2, e)$ and $v = c^r \cdot d^{r\omega}$, where H is a collision-resistant hash function (formally, the key for the hash function can appear in the CRS; we ignore this for simplicity).
3. P_i sets $c = (u_1, u_2, e, v)$, and sends $(sid, ssid, c)$ to P_j.
4. P_j stores $(sid, ssid, P_i, P_j, c)$ and outputs $(\mathsf{receipt}, sid, ssid, P_i, P_j)$. P_j ignores any later commitment messages with the same $(sid, ssid)$ from P_i.

The decommit phase:

1. Upon input $(\mathsf{reveal}, sid, ssid, P_i, P_j)$, party P_i sends $(sid, ssid, x)$ to P_j
2. P_j computes $m = G(x, sid, ssid, i, j)$
3. *Proof of committed value:* P_i proves to P_j that m is the encrypted value. This is equivalent to P_i proving that there exists a value r such that
$$u_1 = g_1^r, \quad u_2 = g_2^r, \quad \frac{e}{m} = h^r, \quad \text{and} \quad v = (cd^\omega)^r$$
The proof is carried out as follows:
   (a) P_j sends $(sid, ssid, c')$ to P_i, where $c' = (g_1^R \cdot g_2^S, h_1^R \cdot h_2^S \cdot G(\varepsilon))$ is a commitment to a random challenge $\varepsilon \in_R \{0,1\}^n$, and $R, S \in_R \mathbb{Z}_q$.
   (b) P_i computes $\alpha = g_1^s$, $\beta = g_2^s$, $\gamma = h^s$ and $\delta = (cd^\omega)^s$, and sends $(sid, ssid, \alpha, \beta, \gamma, \delta)$ to P_j.
   (c) P_j sends the decommitment $(sid, ssid, R, S, \varepsilon)$ to the challenge to P_i.
   (d) P_i verifies that $c' = (g_1^R \cdot g_2^S, h_1^R \cdot h_2^S \cdot G(\varepsilon))$. If no, P_i aborts. Otherwise, P_i computes $z = s + \varepsilon r$ and sends $(sid, ssid, z)$ to P_j.
   (e) P_j outputs $(\mathsf{reveal}, sid, ssid, P_i, P_j, x)$ if and only if
$$g_1^z = \alpha \cdot u_1^\varepsilon, \quad g_2^z = \beta \cdot u_2^\varepsilon, \quad h^z = \gamma \cdot \left(\frac{e}{m}\right)^\varepsilon, \quad \text{and} \quad (cd^\omega)^z = \delta \cdot v^\varepsilon$$

Concrete efficiency: The cost of the protocol in the number of exponentiations (all other operations are insignificant) is as follows:

1. P_i computes 5 exponentiations in order to generate the commitment, and 8 exponentiations in the decommit phase (note that 4 of these exponentiations are in order to verify the challenge ε from P_j, and since cd^ω was already computed in the commit stage only a single exponentiation is needed for δ).
2. P_j computes 0 exponentiations in the commit phase, and 13 exponentiations in the decommit phase.

Overall, the parties compute 26 exponentiations. Observe that P_i can preprocess all but 6 of its exponentiations. This is because it can compute $g_1^r, g_2^r, h^r, c^r, d^r$ and $g_1^s, g_2^s, h^s, c^s, d^s$ before m and thus ω is known. Once $(x, sid, ssid, P_i, P_j)$ is given and thus m can be computed, P_i just needs to compute $(d^r)^\omega$ to finish the commitment and $(d^s)^\omega$ to finish the first message of the decommit stage. Finally, it needs 4 more exponentiation to verify the ε sent by P_j Likewise, P_j can preprocess 4 of its exponentiations by generating c' ahead of time. We conclude that the protocol requires 26 exponentiations overall, but using preprocessing the committer P_i needs to compute only 6 exponentiations and the receiver P_j needs to compute only 9 exponentiations. In addition, the computations $g_1^R \cdot g_2^S$ and $h_1^R \cdot h_2^S$ needed for computing and verifying the encryption of ε can be computed at the cost of $1\frac{1}{3}$ exponentiations each, using the optimization appearing in [18, Sec. 14.6]. Thus, the effective number of exponentiations can be reduced to $23\frac{1}{3}$.

3.3 Proof of Security

Theorem 1 *Assuming that the DDH assumption holds in the group $\mathbb{G}$, Protocol 2 UC-securely realizes the $\mathcal{F}_{\text{MCOM}}$ functionality in the $\mathcal{F}_{\text{CRS}}$-hybrid model, in the presence of static adversaries.*

Proof: The intuition behind the proof of security already appears in Section 3.1. We therefore proceed directly to the description of the simulator and the proof of security.

The simulator $\mathcal{S}$:

- **Initialization step:** $\mathcal{S}$ chooses a public-key/private-key pair for the Cramer-Shoup cryptosystem; let $(\mathbb{G}, q, g_1, g_2, c, d, h)$ be the public-key. In addition, $\mathcal{S}$ chooses a random ρ and computes $h_1 = g_1^\rho$ and $h_2 = g_2^\rho$. $\mathcal{S}$ sets the CRS to be $(\mathbb{G}, q, g_1, g_2, c, d, h, h_1, h_2)$.
- **Simulating the communication with $\mathcal{Z}$:** Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{Z}$) and vice versa.
- **Simulating the commit stage when the committer P_i is corrupted and the receiver P_j is honest:** Upon receiving $(sid, ssid, c)$ from $\mathcal{A}$ as it intends to send from P_i to P_j, the simulator $\mathcal{S}$ uses its knowledge of the Cramer-Shoup secret key to decrypt c. Let $m = G(x, sid', ssid', i', j')$ be the result. If $(sid', ssid', i', j') \neq (sid, ssid, i, j)$ or the decryption is invalid, then $\mathcal{S}$ sends a dummy commitment $(\mathsf{commit}, sid, ssid, P_i, P_j, 0)$ to $\mathcal{F}_{\text{MCOM}}$. Otherwise, $\mathcal{S}$ sends $(\mathsf{commit}, sid, ssid, P_i, P_j, x)$ to $\mathcal{F}_{\text{MCOM}}$.
- **Simulating the decommit stage when P_i is corrupted and P_j is honest:** $\mathcal{S}$ runs the honest strategy of P_j with $\mathcal{A}$ controlling P_i. If P_j would output $(\mathsf{reveal}, sid, ssid, P_i, P_j, x)$, then $\mathcal{S}$ sends $(\mathsf{reveal}, sid, ssid, P_i, P_j)$ to $\mathcal{F}_{\text{MCOM}}$. Otherwise, it does nothing.
- **Simulating the commit stage when P_i is honest and P_j is corrupted:** Upon receiving $(\mathsf{receipt}, sid, ssid, P_i, P_j)$ from $\mathcal{F}_{\text{MCOM}}$, the simulator $\mathcal{S}$ computes a Cramer-Shoup encryption c of 0, and hands $(sid, ssid, c)$ to $\mathcal{A}$, as it expects to receive from P_i.

– **Simulating the decommit stage when P_i is honest and P_j is corrupted:** Upon receiving $(\mathsf{reveal}, sid, ssid, P_i, P_j, x)$ from $\mathcal{F}_{\text{MCOM}}$, $\mathcal{S}$ works as follows:
  1. $\mathcal{S}$ hands $(sid, ssid, x)$ to $\mathcal{A}$, as it expects to receive from P_i.
  2. $\mathcal{S}$ receives c' from $\mathcal{A}$ and uses its knowledge of the discrete log ρ of h_1, h_2 (in the CRS) in order to decrypt the encryption c' of $G(\varepsilon)$ and obtain ε.
  3. Let $c = (u_1, u_2, e, v)$ be as computed by $\mathcal{S}$ in the commit stage. $\mathcal{S}$ chooses a random $z \in_R \mathbb{Z}_q$ and computes $\alpha = g_1^z/u_1^\varepsilon$, $\beta = g_2^z/u_2^\varepsilon$, $\gamma = h^z/(e/m)^\varepsilon$ and $\delta = (cd^\omega)^z/v^\varepsilon$, and hands $(\alpha, \beta, \gamma, \delta)$ to $\mathcal{A}$.
  4. $\mathcal{S}$ receives (R', S', ε') from $\mathcal{A}$. If $c' \neq (g_1^{R'} \cdot g_2^{S'}, h_1^{R'} \cdot h_2^{S'} \cdot G(\varepsilon'))$ then $\mathcal{S}$ simulates P_i aborting the decommitment. Otherwise, $\varepsilon' = \varepsilon$ (this must be the case because when the regular public-key of the dual encryption scheme is used the encryption is perfectly binding), and $\mathcal{S}$ hands z to $\mathcal{A}$.

  Simulation in the cases that both P_i and P_j are honest is straightforward. This is due to the fact that when both parties are honest, the simulator can choose the value ε itself and generate a valid proof for any value needed.

Analysis of the simulation: Denoting Protocol 2 by π and recalling that it runs in the $\mathcal{F}_{\text{CRS}}$-hybrid model, we need to prove that for every $\mathcal{A}$ and every $\mathcal{Z}$,

$$\left\{\text{IDEAL}_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(n, z)\right\}_{n \in \mathbb{N}; z \in \{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}_{\pi, \mathcal{A}, \mathcal{Z}}(n, z)\right\}_{n \in \mathbb{N}; z \in \{0,1\}^*}.$$

We prove this via a series of hybrid games.

Hybrid game HYB-GAME^1: In this game, the ideal functionality gives the simulator $\mathcal{S}_1$ the value x committed to by an honest P_i together with the regular $(\mathsf{receipt}, sid, ssid, P_i, P_j)$ message. $\mathcal{S}_1$ works in exactly the same way as $\mathcal{S}$ except that when simulating the commit stage when P_i is honest and P_j is corrupted, it computes c as an encryption of $m = G(x, sid, ssid, i, j)$ as an honest P_i would. Otherwise, it behaves exactly as $\mathcal{S}$ in the simulation. In order to show that the output of $\mathcal{Z}$ in HYB-GAME^1 is indistinguishable from its output in IDEAL, we need to reduce the difference to the security of the encryption scheme. However, $\mathcal{S}$ and $\mathcal{S}_1$ need to decrypt in the simulation of the commit stage when the committer P_i is corrupted and P_j is honest (see the simulator description). $\mathcal{S}$ and $\mathcal{S}_1$ can carry out this decryption because they know the Cramer-Shoup secret-key. But, this means that security cannot be reduced to this scheme. We solve this problem by using the fact that the Cramer-Shoup encryption scheme is CCA2-secure. Thus, $\mathcal{S}$ and $\mathcal{S}_1$ can decrypt by using their decryption oracle. We use the LR-formulation of CCA2-security [1]. In this formulation a bit b is randomly chosen and the adversary can ask for many *encryption challenges*. Each query consists of a pair (m_0, m_1) and the adversary receives back an encryption of m_b (always with the same b). The aim of the adversary is to guess the bit b. Of course, given that this is a CCA2 game, the adversary can ask for a decryption of any ciphertext that was not received as an encryption of one of the pairs.

Formally, we construct a CCA2 adversary $\mathcal{A}_{\text{CS}}$ attacking the Cramer-Shoup scheme as follows. Let $(\mathbb{G}, q, g_1, g_2, c, d, h)$ be the public-key given to $\mathcal{A}_{\text{CS}}$. Adversary $\mathcal{A}_{\text{CS}}$ chooses $\rho \in_R \mathbb{Z}_q$, computes $h_1 = g_1^\rho$ and $h_2 = g_2^\rho$, and sets the CRS to be

$(\mathbb{G}, q, g_1, g_2, c, d, h, h_1, h_2)$. Then $\mathcal{A}_{\text{CS}}$ simulates an execution of $\text{IDEAL}_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}$ with the following differences:

1. Whenever an honest P_i commits to a value x, instead of $\mathcal{S}$ encrypting 0 (or $\mathcal{S}_1$ encrypting x), $\mathcal{A}_{\text{CS}}$ generates the encryption in the ciphertext by asking for an encryption challenge of the pair $(0, G(x, sid, ssid, i, j))$. The ciphertext c received back is sent as the commitment. (Note that $\mathcal{A}_{\text{CS}}$ knows x because it runs $\mathcal{Z}$ and so knows the inputs handed to the honest parties.)
2. Whenever a corrupted P_i sends a commitment value $(sid, ssid, c)$ and the simulator needs to decrypt c, $\mathcal{A}_{\text{CS}}$ queries its decryption oracle with c. If c was received as a ciphertext challenge then $\mathcal{A}_{\text{CS}}$ has the simulator send a dummy commitment $(\mathsf{commit}, sid, ssid, P_i, P_j, 0)$ to $\mathcal{F}_{\text{MCOM}}$ as in the case that $(sid', ssid', i', j') \neq (sid, ssid, i, j)$ in the simulation. Since c was received as a ciphertext challenge, indeed it holds that $(sid', ssid', i', j') \neq (sid, ssid, i, j)$ and so this is the same.

Finally, $\mathcal{A}_{\text{CS}}$ outputs whatever $\mathcal{Z}$ outputs.

Now, if $b = 0$ in the CCA2 game, then all of the commitments c generated when the committer P_i is honest are to 0. Thus, the simulation is exactly like $\mathcal{S}$ and the output of $\mathcal{A}_{\text{CS}}$ is exactly that of $\text{IDEAL}_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(n, z)$. (Note that all other instructions are carried out identically to $\mathcal{S}$.) In contrast, if $b = 1$, then the commitments generated are to the correct values x and so the simulation is exactly like $\mathcal{S}_1$. Thus, the output of $\mathcal{A}_{\text{CS}}$ is exactly that of $\text{HYB-GAME}^1_{\mathcal{S}_1^{\mathcal{A}}, \mathcal{Z}}(n, z)$. We conclude that

$$\left\{\text{HYB-GAME}^1_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}_1^{\mathcal{A}}, \mathcal{Z}}(n, z)\right\}_{n;z} \stackrel{\text{c}}{\equiv} \left\{\text{IDEAL}_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(n, z)\right\}_{n;z},$$

by the fact that the Cramer-Shoup encryption scheme is CCA2-secure.

Hybrid game HYB-GAME^2: In this game, the simulator $\mathcal{S}_2$ works in exactly the same way as $\mathcal{S}_1$, except that when simulating the decommitment phase when P_i is honest and P_j is corrupted, it computes the messages $(\alpha, \beta, \gamma, \delta)$ and z in the proof exactly as an honest P_i would. It can do this because the commitment c sent in the commitment phase is to the correct value $m = G(x, sid, ssid, i, j)$ and so it can play the honest prover. The output distribution of this game is *identical* to HYB-GAME^1 by the perfect simulation property of the proof of the decommitment phase. This proof is based on a standard Sigma protocol that a tuple is a Diffie-Hellman tuple and it is straightforward to verify that the distributions are identical. We therefore have that: We conclude that

$$\left\{\text{HYB-GAME}^2_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}_2^{\mathcal{A}}, \mathcal{Z}}(n, z)\right\}_{n;z} \equiv \left\{\text{HYB-GAME}^1_{\mathcal{F}_{\text{MCOM}}, \mathcal{S}_1^{\mathcal{A}}, \mathcal{Z}}(n, z)\right\}_{n;z}.$$

Completing the proof: It remains to show that the output of $\mathcal{Z}$ after an execution of π in the $\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}$ model is indistinguishable from its output after the HYB-GAME^2 game. First, observe that the commitment and decommitment messages in the case of an honest committer P_i are identical in both HYB-GAME^2 and a real protocol execution in the $\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}$ model. Thus, the only difference between the output of $\mathcal{Z}$ in both cases can be due to the value x output by an honest receiver P_j after a decommit from a corrupted sender P_i. This is due to the

fact that in HYB-GAME^2, the value x output by an honest P_j is the value sent by $\mathcal{S}_2$ to $\mathcal{F}_{\text{MCOM}}$ after decrypting the associated ciphertext in the commit stage using the Cramer-Shoup secret-key. In contrast, in HYBRID$^{\mathcal{F}_{\text{CRS}}}$ the value x output by an honest party is that sent by $\mathcal{A}$ in the first step of the decommitment stage (as long as the proof passes). These values can only be different if $\mathcal{A}$ can convince an honest P_j to output x in the decommitment phase, even though the encrypted value c sent in the commitment phase is not to $m = G(x, sid, ssid, i, j)$. Thus, this difference reduces to the *soundness* of the proof in the decommitment phase. Recall that by the special soundness property of Sigma protocols, in the case that c is *not* an encryption of $m = G(x, sid, ssid, i, j)$, for every first message $(\alpha, \beta, \gamma, \delta)$ there is only a *single* ε for which there exists a convincing answer z.

It is tempting to conclude that since the encryption of ε is semantically secure, the adversary cannot cheat in the Sigma protocol. However, this requires a reduction and such a reduction cannot be carried out because the adversary does not "reveal" to us whether it succeeds in the proof until we decrypt ε. Thus, one cannot reduce the ability of the adversary to cheat to the hiding of ε (in such a reduction, one cannot reveal ε together with the randomness used to encrypt). However, it *is* possible to replace the values h_1, h_2 where $h_1 = g_1^{\rho}$ and $h_2 = g_2^{\rho}$ with values $h_1 = g_1^{\rho_1}$ and $h_2 = g_2^{\rho_2}$ for $\rho_1, \rho_2 \in_R \mathbb{Z}_q$. In such a case, as we have discussed, the encryption c' perfectly hides the value ε. Furthermore, there is no need to ever decrypt c' here (the simulator $\mathcal{S}_2$ in HYB-GAME^2 does not decrypt these values). Thus, there is no problem replacing h_1, h_2 in this way. Finally, recall that the alternative key h_1, h_2 is indistinguishable from the regular one. Thus, defining HYB-GAME^3 to be the same as HYB-GAME^2 except that the keys h_1, h_2 are different as described, and letting $\mathcal{S}_3 = \mathcal{S}_2$ (except again for how h_1, h_2 are chosen), we have

$$\left\{\text{HYB-GAME}^3_{\mathcal{F}_{\text{MCOM}},\mathcal{S}_3^{\mathcal{A}},\mathcal{Z}}(n,z)\right\}_{n;z} \stackrel{\text{c}}{\equiv} \left\{\text{HYB-GAME}^2_{\mathcal{F}_{\text{MCOM}},\mathcal{S}_2^{\mathcal{A}},\mathcal{Z}}(n,z)\right\}_{n;z}.$$

We are now ready to conclude the proof. Since the encryption c' perfectly hides the challenge ε, the probability that $\mathcal{A}$ successfully proves an incorrect statement in the decommitment stage is at most 2^{-n} (recall that there is exactly one ε that it can answer). Thus, the value sent by $\mathcal{S}_3$ to $\mathcal{F}_{\text{MCOM}}$ is the same value as that output by an honest P_j, except with negligible probability. The only other difference is that in HYB-GAME^3 an alternative public-key for the dual mode cryptosystem is used, whereas in HYBRID a regular one is used. Recalling that these keys are computationally indistinguishable, we conclude that

$$\left\{\text{HYBRID}^{\mathcal{F}_{\text{CRS}}}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathbb{N};z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{HYB-GAME}^3_{\mathcal{F}_{\text{MCOM}},\mathcal{S}_2^{\mathcal{A}},\mathcal{Z}}(n,z)\right\}_{n\in\mathbb{N};z\in\{0,1\}^*}.$$

Combining all of the above, we have that the output of $\mathcal{Z}$ with $\mathcal{A}$ after an execution of π in the $\mathcal{F}_{\text{CRS}}$-hybrid model is computationally indistinguishable from its output after an execution with $\mathcal{S}^{\mathcal{A}}$ and $\mathcal{F}_{\text{MCOM}}$ in the IDEAL model, and so Protocol 2 is UC-secure in the presence of static adversaries, as required. ■

4 Adaptive Adversaries with Erasures

4.1 Background and Outline of Solution

In the setting of adaptive corruptions, the adversary can corrupt parties throughout the computation. Upon corruption, it receives the local state of the parties, including randomness it has used and so on. In the model with erasures, a protocol can instruct a party to erase some of its state (e.g., old keys), and in such a case the adversary does not obtain the erased state upon corruption. Adaptive corruptions accurately models the realistic setting where parties can be "hacked" during a computation. As such, it is desirable to have protocols that are secure in this model.

This model introduces significant difficulties when proving security. Specifically, observe that Protocol 2 is not secure in the presence of adaptive adversaries, even with erasures, because the committer must store the randomness r used to commit to x in order to run the decommitment stage. Now, in our simulation, the simulator commits to 0, even when the commitment is really to x. However, upon corruption in the real world, the adversary obtains r and x such that c is encryption of x using randomness r. In the simulation, such randomness can never be produced because c is an encryption of 0 and not of x (there does not exist an r' that can explain c as an encryption of $x \neq 0$).

Achieving adaptive security. Our protocol can be modified so that it achieves adaptive security with erasures, with little additional cost. Interestingly, the only modifications necessary are a change in the order of operations and 1 additional Pedersen commitment. In order to see this, recall that the problem with achieving adaptive security is that the committer cannot erase r before sending c in the commit phase, because then it will not be able to prove the proof in the decommit phase. However, it is possible for the parties to run most of the proof already in the commit phase, *before* the commitment is even sent (actually, the ciphertext c is committed to equivocally, but not yet revealed). That is, the committer and receiver run the zero-knowledge protocol before c is sent, without the committer sending the last message z. In addition, the committer commits to its first message $(\alpha, \beta, \gamma, \delta)$ of the protocol instead of sending it in the clear. (Thus, the receiver sends a commitment to ε; the committer sends a commitment to $(\alpha, \beta, \gamma, \delta)$; the receiver decommits revealing ε; finally, the committer prepares z based on ε without sending it.) Following this preamble, the committer erases all of its randomness, except for that needed to decommit to the first message of the zero-knowledge protocol, and only then reveals c. This completes the commit phase. The decommit phase simply consists of the committer sending the decommitment to $(\alpha, \beta, \gamma, \delta)$ and the message z (which has already been prepared), and the receiver verifies the decommitment and that $((\alpha, \beta, \gamma, \delta), \varepsilon, z)$ constitutes an accepting transcript.

Observe that before the committer sends c, nothing has actually been revealed; the committer only sent a commitment to $(\alpha, \beta, \gamma, \delta)$. Thus, this does not affect the hiding property of the original commitment scheme. Furthermore, the committer erases all secret state before sending c, and in particular erases the

random coins used to generate c. Thus adaptive corruptions make no difference because the committer has no secret state once c is sent, and has revealed no information before c is sent. In actuality, in order to achieve this property that all messages sent before c are independent of x, we have to have the committer commit to the first message of the proof using a *perfectly hiding* commitment scheme. Furthermore, it needs to be adaptively secure in that upon corruption, the prover can open it to anything that it wishes. Fortunately, this can be easily achieved by using a Pedersen commitment $Com(x) = g^r \cdot \hat{h}^x$ with a value $\hat{h}$ that appears in the CRS.[3] (Note that given the discrete log $\hat{\rho}$ of $\hat{h}$ it is possible to decommit to any value desired. Specifically, commit by computing $c = g^a$ for a known a. Now, given x we wish to find r such that $c = g^a = g^r \cdot \hat{h}^x$. Given that $\hat{h} = g^{\hat{\rho}}$ this means that we need to find r such that $g^a = g^{r+\hat{\rho}x}$, or equivalently r such that $a = r + \hat{\rho}x \bmod q$. Thus, just take $r = a - \hat{\rho}x \bmod q$.) We remark that although the above works, it introduces an additional difficulty because the soundness of the Sigma protocol now also rests on the hardness of finding the discrete log of $\hat{h}$. This requires an additional reduction; see the proof for details. See Protocol 3 for the outline of the modified protocol.

PROTOCOL 3 (UC-commitment template for adaptive security)

Common reference string: $(pk_1, pk_2, \mathbb{G}, q, g, \hat{h})$ where pk_1 is the public-key of a CCA2-secure encryption scheme, pk_2 is the public-key of a dual mode cryptosystem, and $(\mathbb{G}, q, g, \hat{h})$ are parameters for the Pedersen commitment scheme.

The commit phase:

1. The committer computes a commitment to x as $c = E^{cca}_{pk_1}(x; r)$, and sends a Pedersen commitment of c to the receiver, using $(g, \hat{h})$.
2. The receiver sends $c' = E^{dual}_{pk_2}(\varepsilon; r')$; its commitment to the first message of the proof.
3. The committer sends a Pedersen commitment to the first prover message α to the receiver, computed from the ciphertext c (observe that c has not yet been revealed).
4. The receiver decommits to ε by sending ε and r'
5. The committer checks that $c' = E^{dual}_{pk_2}(\varepsilon; r')$ and if yes, it computes the reply z for the Sigma protocol, based on (α, ε).
6. The committer now *erases* r and the randomness used to generate α and z, stores α, z and the randomness used to generate the Pedersen commitments, and finally sends c and its decommitment to the receiver.

The decommitment phase:

1. The committer sends x, (α, z), and the randomness used to generate the Pedersen commitment to α to the receiver.
2. The receiver outputs x as the decommitted value if and only if the Pedersen commitment was to α and (α, ε, z) is an accepting Sigma-protocol transcript.

[3] We note that such a commitment is not extractable but we do not need it to be.

4.2 The Adaptive Protocol

The scheme that is adaptively secure with erasures appears in Protocol 4.

PROTOCOL 4 (UC-Secure Commitment – Adaptive with Erasures)

Common reference string: $(\mathbb{G}, q, g_1, g_2, c, d, h, h_1, h_2, \hat{h})$ where all parameters are as in Protocol 2, and $(\mathbb{G}, q, g_1, \hat{h})$ are parameters for Pedersen commitments.)

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

1. P_i computes $m = G(x, sid, ssid, i, j)$. (The identities i, j can be mapped to $\{0,1\}^{\log^2 n/4}$ and so overall $(x, sid, ssid, i, j)$ is an n-bit string.)
2. P_i chooses a random $r \in_R \mathbb{Z}_q$, computes $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r \cdot m$, $\omega = H(u_1, u_2, e)$ and $v = c^r \cdot d^{r\omega}$, where H is a collision-resistant hash function (formally, the key for the hash function can appear in the CRS; we ignore this for simplicity). P_i sets $c = (u_1, u_2, e, v)$.
3. P_i chooses $\kappa_1 \in_R \mathcal{Z}_q$, computes $c^1_{ped} = g_1^{\kappa_1} \cdot \hat{h}^{H(c)}$, and sends c^1_{ped} to P_j.
4. P_j sends $c' = (g_1^R \cdot g_2^S, h_1^R \cdot h_2^S \cdot G(\varepsilon))$ to P_i, where $\varepsilon \in_R \{0,1\}^n$ and $R, S \in_R \mathbb{Z}_q$.
5. P_i computes $\alpha = g_1^s$, $\beta = g_2^s$, $\gamma = h^s$ and $\delta = (cd^\omega)^s$, and computes a Pedersen commitment $c^2_{ped} = g_1^{\kappa_2} \cdot \hat{h}^{H(\alpha,\beta,\gamma,\delta)}$, where $\kappa_2 \in_R \mathbb{Z}_q$. P_i sends c^2_{ped} to P_j.
6. P_j sends (R, S, ε) to P_i.
7. P_i verifies that $c' = (g_1^R \cdot g_2^S, h_1^R \cdot h_2^S \cdot G(\varepsilon))$. If no, it aborts. Otherwise, P_i computes $z = s + \varepsilon r$.
8. P_i erases r and s, and stores $(x, \alpha, \beta, \gamma, \delta, \kappa_2, z)$. P_i sends (κ_1, c) to P_j.
9. P_j verifies that $c^1_{ped} = g_1^{\kappa_1} \cdot \hat{h}^{H(c)}$. If yes, it stores $(sid, ssid, P_i, P_j, c, \varepsilon, c^2_{ped})$ and outputs $(\mathsf{receipt}, sid, ssid, P_i, P_j)$. P_j ignores any later commitment messages with the same $(sid, ssid)$ from P_i.

The decommit phase:

1. Upon input $(\mathsf{reveal}, sid, ssid, P_i, P_j)$, P_i sends $(x, \alpha, \beta, \gamma, \delta, \kappa_2, z)$ to P_j.
2. P_j sets $m = G(x, sid, ssid, i, j)$ and outputs $(\mathsf{reveal}, sid, ssid, P_i, P_j, x)$ if and only if

$$c^2_{ped} = g_1^{\kappa_2} \cdot \hat{h}^{H(\alpha,\beta,\gamma,\delta)},\ g_1^z = \alpha \cdot u_1^\varepsilon,\ g_2^z = \beta \cdot u_2^\varepsilon,\ h^z = \gamma \cdot \left(\frac{e}{m}\right)^\varepsilon, (cd^\omega)^z = \delta \cdot v^\varepsilon$$

We note one difference between the actual protocol and the intuitive explanation above, regarding the Pedersen commitments. We use these commitments to commit to group elements in $\mathbb{G}$. However, the input of a Pedersen commitment is in $\mathbb{Z}_q$ and not $\mathbb{G}$. One solution to this is to break the elements up into pieces and separately commit to each piece. We use a different solution which is to compute $Com(m) = g^r \cdot \hat{h}^{H(m)}$ where H is a collision-resistant hash function. The commitment is still perfectly hiding and can be opened to any value. The only difference is that the binding property relies now both on the hardness of the discrete log problem (as in the standard case) and on the collision resistance of H. By convention, in Protocol 4, all messages are sent together with $(sid, ssid)$.

Efficiency. The complexity of Protocol 4 is the same as the static version (Protocol 2) plus two additional Pedersen commitment that must be computed and verified. Naively, this costs an additional 8 exponentiations overall. However, again using the multiexponentiation optimization, these can be computed at the effective cost of $5\frac{1}{3}$ exponentiations. Thus, we have an overall cost of $28\frac{2}{3}$ exponentiations.

4.3 Proof of Security

Theorem 2 *Assuming that the DDH assumption holds in the group $\mathbb{G}$, Protocol 4 UC-securely realizes the $\mathcal{F}_{\text{MCOM}}$ functionality in the $\mathcal{F}_{\text{CRS}}$-hybrid model, in the presence of adaptive adversaries with erasures.*

Proof: The proof of security is very similar to the static case, with the addition of how to deal with adaptive corruptions. We remark that we follow the convention where the only part of the commitment message not seen by the ideal adversary is the commitment value [6]. Thus, when an honest P_i sends a message to the $\mathcal{F}_{\text{MCOM}}$ functionality, the adaptive ideal adversary knows what type of message it is and who the intended recipient is.

Due to lack of space in this extended abstract, we sketch the main difference between the proof here, and the proof in the static case. The main observation is that if a committing party is corrupted *before* the commitment stage is finished, then no meaningful information has been given away. This is due to the use of Pedersen commitments and the fact that the simulator can open them to any way it wishes by choosing $\hat{h}$ so that it knows its discrete log. Furthermore, if a committing party is corrupted *after* the commitment phase is finished, then the randomness used to generate the Cramer-Shoup encryption and the Sigma protocol prover messages has already been erased. Thus, all the simulator has to do is to run the Sigma-protocol simulator using the proof statement based on the commitment value obtained, and this will look exactly like an honestly generated commitment.

Due to the above, the simulation and proof in this case of adaptive corruptions is very similar to the case of static corruptions. However, there is one major difference regarding the last step where we must prove the soundness of the proofs provided by corrupted parties. Specifically, we need to claim that a corrupted party can prove an incorrect statement with probability that is at most negligible. In order to prove this, we first replace the dual mode public key with the alternative one, as in the proof of the case of static corruptions. However, this does not yet suffice because the adversary may be able to prove an incorrect claim by breaking the computational binding of the Pedersen commitments (recall that the last message of the proof is decommitted to only after the challenge ε is revealed). Despite this, we use the fact that the ability to decommit to two different values is equivalent to find the discrete log of $\hat{h}$. Specifically, given c_{ped} together with $(\kappa, m) \neq (\kappa', m')$ such that $c_{ped} = g_1^{\kappa} \cdot \hat{h}^m = g_1^{\kappa'} \cdot \hat{h}^{m'}$, it holds that $\hat{h} = g_1^{(\kappa-\kappa')(m'-m)^{-1}}$ and so the discrete log of $\hat{h}$ is $\frac{\kappa-\kappa'}{m'-m}$ which can be efficiently computed.

We therefore prove soundness as follows. Assume that there exists an environment $\mathcal{Z}$, adversary $\mathcal{A}$, and an input z to $\mathcal{Z}$ such that for infinitely many n's, $\mathcal{A}$ succeeds in proving an incorrect statement with non-negligible probability. In this case, $\mathcal{A}$ will succeed in proving an incorrect statement with non-negligible probability also in HYB-GAME^3. (We remark that it is possible to detect this event because we can decrypt the Cramer-Shoup encryption and see what value was actually encrypted.) Now, let $(\mathbb{G}, q, g_1, \hat{h})$ be parameters for a Pedersen commitment. An adversary $\mathcal{A}_{ped}$ attempting to break the commitment scheme receives the parameters and works as follows. It chooses all of the values in the common reference string like $\mathcal{S}$ (based on $(\mathbb{G}, q, g_1)$) and then simulates the HYB-GAME^3 experiment running $\mathcal{Z}$ with input z and $\mathcal{A}$. If $\mathcal{A}$ proves an incorrect statement, then $\mathcal{A}_{ped}$ *rewinds* the entire execution (including $\mathcal{Z}$) until the point that $\mathcal{A}$ sent c^2_{ped}. $\mathcal{A}_{ped}$ then sends a different decommitment (R', S', ε') to a fresh random ε'. Note that since at this point the public key for the dual-mode cryptosystem is the alternative one, and $\mathcal{A}_{ped}$ knows the discrete logs ρ_1, ρ_2 of h_1, h_2, it can efficiently find (R', S', ε') such that $c' = (g_1^R \cdot g_2^S, h_1^R \cdot h_2^S \cdot G(\varepsilon))$ even though c' was originally generated as an encryption of some $\varepsilon \neq \varepsilon'$. (In order to do this, $\mathcal{A}_{ped}$ also needs to know the discrete logs of g_2 and $G(\varepsilon), G(\varepsilon')$ relative to g_1, but these value can be chosen in that way. See the explanation of the concrete dual-mode cryptosystem at the end of Section 3.1 in order to see what equations $\mathcal{A}_{ped}$ needs to solve.) $\mathcal{A}_{ped}$ then continues the execution until the point that $\mathcal{A}$ decommits to the transcript. If it is accepting, then $\mathcal{A}$ must have opened the Pedersen commitment c^2_{ped} differently (because $\mathcal{A}$ can only answer one ε if the statement is incorrect). In this case $\mathcal{A}_{ped}$ has found the discrete log of $\hat{h}$ and halts. Otherwise, $\mathcal{A}_{ped}$ repeatedly rewinds until $\mathcal{A}$ does provide an accepting transcript. This yields an expected polynomial-time adversary $\mathcal{A}_{ped}$; a strict polynomial-time adversary can be derived by just truncating the execution after enough time. We remark that although we are working in the UC framework, $\mathcal{A}_{ped}$ is allowed to rewind in the reduction because this has nothing to do with the simulation, and we are reducing the difference between HYB-GAME^3 and HYBRID to the hardness of finding the discrete log of $\hat{h}$. ■

Acknowledgements

We thank Ran Canetti and Benny Pinkas for helpful discussions.

References

1. M. Bellare and P. Rogaway. *Introduction to Modern Cryptography, Chapter 7* (course notes), 2007.
2. A. Ben-David, N. Nisan and B. Pinkas. FairplayMP: a System for Secure Multi-Party Computation. In the 15*th ACM CCS*, pages 257–266, 2008.
3. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.
4. R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO 2001*, Springer (LNCS 2139), pages 19–40, 2001.

5. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *CRYPTO 2002*, Springer (LNCS 2442), 143–161, 2002.
6. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002. Full version available at `http://eprint.iacr.org/2002/140`.
7. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO 1998*, Springer (LNCS 1462), pages 13–25, 1998.
8. I. Damgård. On Σ Protocols. `http://www.daimi.au.dk/~ivan/Sigma.pdf`.
9. I. Damgard and J. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In *CRYPTO 2002,* Springer-Verlag (LNCS 2442), pages 581–596, 2002.
10. C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols – Techniques and Constructions*. Springer, October 2010.
11. C. Hazay, J. Katz, C.Y. Koo and Y. Lindell. Concurrently-Secure Blind Signatures without Random Oracles or Setup Assumptions. In the 5*th TCC*, Springer (LNCS 4392), pages 323–341, 2007.
12. C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC 2010*, Springer (LNCS 6056), pages 312–331, 2010. Full version in the *Cryptology ePrint Archive*, report 2009/594.
13. J. Kelsey, B. Schneier and D. Wagner. Protocol Interactions and the Chosen Protocol Attack. In the 5*th International Security Protocols Workshop*, Springer (LNCS 1361), pages 91–104, 1998.
14. G. Kol and M. Naor. Cryptography and Game Theory: Designing Protocols for Exchanging Information. In 5*th TCC*, Springer (LNCS 4948), 320–339, 2008.
15. H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In *CRYPTO 2005*, Springer (LNCS 3621), pages 546–566, 2005.
16. Y. Lindell and B. Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In the *8th TCC*, Springer (LNCS 6597), pages 329-346, 2011.
17. Y. Lindell, B. Pinkas and N.P. Smart. Implementing Two-Party Computation Efficiently with Security Against Malicious Adversaries. In the 6*th SCN*, pages 2–20, 2008.
18. A. Menezes, P. Van Oorschot and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
19. T. Moran and T. Moore. The Phish-Market Protocol: Securely Sharing Attack Data between Competitors. 14*th Financial Cryptography*, Springer (LNCS 6052), pages 222–237, 2010.
20. M. Osadchy, B. Pinkas, A. Jarrous and B. Moskovich. SCiFI - A System for Secure Face Identification. In the 31*st IEEE Symposium on Security and Privacy*, pages 239–254, 2010.
21. P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT '99*, Springer (LNCS 1592), pages 223–238, 1999.
22. C. Peikert, V. Vaikuntanathan and B. Waters. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO'08*, Springer (LNCS 5157), pages 554–571, 2008.
23. B. Pinkas, T. Schneider, N.P. Smart and S.C. Williams. Secure Two-Party Computation Is Practical. In *ASIACRYPT 2009*, Springer (LNCS 5912), pages 250–267, 2009.
24. S. Vanstone. Deployments of Elliptic Curve Cryptography. In the *9th Workshop on Elliptic Curve Cryptography* (ECC), 2005.
25. The Crypto++ Library, `http://www.cryptopp.com`.