Committing Encryption and Publicly-Verifiable SignCryption

Yitchak Gertner[*] Amir Herzberg[↑]

Department of Computer Science, Bar-Ilan University

16/12/2003

Abstract

Encryption is often conceived as a *committing* process, in the sense that the ciphertext may serve as a commitment to the plaintext. But this does not follow from the standard definitions of secure encryption.

We define and construct symmetric and asymmetric *committing encryption* schemes, enabling publicly verifiable non-repudiation. Committing encryption eliminates key-spoofing attacks and has also the robustness to be signed afterwards. Our constructions are very efficient and practical. In particular, we show that most popular asymmetric encryption schemes, e.g. RSA, are committing encryption schemes; we also have an (efficient) construction given an arbitrary asymmetric encryption scheme. Our construction of symmetric committing encryption retains the efficiency of the symmetric encryption for real-time operations, although it uses few public key signatures in the setup phase.

Finally, we investigate how to achieve both confidentiality and non-repudiation, and present a *publicly verifiable signcryption* scheme. Contrary to previous signcryption schemes, which are *not* publicly verifiable, our publicly verifiable signcryption supports non-repudiation. We construct a simple and efficient publicly verifiable signcryption scheme based on a new composition method which we call "*commit-encrypt-then-sign*" (*CEtS*) that preserves security properties of both committing encryption and digital signature schemes.

Keywords: Encryption, Commitment, Key-spoofing attack, Committing Encryption, Signcryption, Non-repudiation, Signatures.

1 Introduction

Encryption is often conceived as a *committing* process, in the sense that the ciphertext may serve as a commitment to the plaintext. But without appropriate security properties, the encryption scheme is vulnerable to key-spoofing attacks. Key-spoofing attacks [AN95]

[*] Email: gertner@ieee.org

[↑]Department of Computer Science, Bar-Ilan University, Ramat-Gan 529000, ISRAEL. Email: amir@herzberg.name

derive different message *m*' from the original ciphertext *c* by changing the (decryption) key. Key-spoofing can cause failures of non-repudiation protocols such as of [ZG96a].

We introduce symmetric and asymmetric *committing encryption* schemes. Committing encryption schemes combine the security properties of commitment and encryption schemes such that they ensure that the decrypted message is indeed the original message. Committing encryption schemes allow the sender (encrypt side) to prove to a third party that the recipient (decrypt side) is able to open the ciphertext and retrieve the original plaintext, and allow the recipient to prove to a third party that this is the original plaintext that was sent by the sender. Having these properties, committing encryption schemes eliminate key-spoofing attacks and can be safely signed.

Many existing crypto protocols use encryption and implicitly assume commitment properties. Is this secure, for typical, standard cryptosystem? We show that indeed, most asymmetric cryptosystems such as (padded) RSA are also committing encryption. In fact, any public-key cryptosystem where decryption recovers the random bits used during encryption is also committing encryption. However, we did not find a similar requirement to ensure that symmetric encryption schemes are also committing. In fact, it is easy to show secure encryption schemes where key-spoofing is easy, e.g. one-time pad.

In this work we formally study the notion of committing encryption. We provide a formal definition of committing encryption schemes for both symmetric and asymmetric setting. The definition provides for better understanding the security requirements and the constraints under which an encryption can provide also commitment along with confidentiality.

The definition is essential also for cryptanalysts. Given an encryption algorithm that may be used for commitment, cryptanalysts can analyze and check according to the committing encryption definition whether the encryption algorithm can be used to provide also commitment along with confidentiality or not. This is especially relevant to symmetric encryption; we suggest that this should be an additional criteria in the cryptoanalytical evaluation of proposed symmetric encryption schemes.

The common intuition to use encryption for commitment is getting formalized in this work by proposing a simple method of constructing asymmetric committing encryption scheme that is based on a 'regular' asymmetric encryption scheme that its decryption function restores the random value used by the encryption function. Such an encryption scheme is the popular and most commonly used RSA encryption scheme with EME (Encoding Method for Encryption) operations i.e., with padding or with hashing e.g. using OAEP (Optimal Asymmetric Encryption Padding) [BR94, PKCS1v2.0, PKCS1v2.1]. We further refer to such a scheme as *randomness-recovering asymmetric encryption*.

We present a new composition method which we call "*commit-then-encrypt*". This composition involves several cryptographic tools such as asymmetric encryption, commitment and digital signature schemes. Even though, it is still fairly efficient in the run time. We prove that this simple and efficient composition constructs a secure asymmetric committing encryption scheme. This construction may be used as a mechanism to transform any asymmetric encryption (even if the decryption function does not restore the random value used by the encryption function) to committing encryption.

Another important construction is the construction of symmetric committing encryption scheme. We show that given any symmetric encryption scheme, although by itself it is not a committing encryption, adding a committing key setup protocol results in a secure symmetric committing encryption scheme. We propose a simple method of constructing symmetric committing encryption scheme. This construction involves several cryptographic tools such

as symmetric encryption, commitment and digital signature schemes. Even though, in the run time, its efficiency remains the same as the efficiency of the chosen symmetric encryption. We prove that this simple and efficient construction is a secure symmetric committing encryption scheme.

Finally, we present a *publicly verifiable signcryption* scheme. Contrary to Zheng [Zh97] and An, Dodis and Rabin [ADR02] signcryption schemes which are *not* publicly verifiable, our publicly verifiable signcryption supports non-repudiation. We construct a simple and efficient publicly verifiable signcryption scheme based on a new composition method which we call "*commit-encrypt-then-sign*" (*CEtS*) that preserves security properties of both ingredients committing encryption and digital signature schemes including the non-repudiation feature of the signature scheme.

NOTATIONS

If x, y denote bit strings, $|x|$ denotes the bit length of string x, and $x||y$ denotes the concatenation of strings x and y. Let $\{0, 1\}^*$ be the space of (finite) binary strings. If S is a set then $a \xleftarrow{R} S$ denotes the experiment of selecting a point uniformly from S and assigning a this value.

If $A(\cdot, \cdot, \ldots)$ is any probabilistic algorithm then $a \xleftarrow{R} A(x_1, x_2, ...)$ denotes the experiment of running A on inputs x_1, x_2, … and letting a be the outcome, the probability being over the coins of A obtained internally by A. If the coins of A are given explicitly to A, we denote by r the input of a random value where $r \xleftarrow{R} \{0,1\}^*$ and we write $a \longleftarrow A(x_1, x_2, ..., r)$.

An *adversary* denoted $\mathcal{A}$, is a probabilistic algorithm that may access one or more oracles. An *oracle* is a probabilistic algorithm. If O is an oracle then $\mathcal{A}^{O(X)|_{\forall x \in X, p(x,y,...)}}$ denotes an adversary $\mathcal{A}$ that has access to an oracle O with a restriction that the predicate, $p(x, y, \ldots)$, is true for all the queries $x \in X$, where X is the set of the queries x that were given by $\mathcal{A}$ as inputs to O in one invocation of $\mathcal{A}$.

When S, T, … denotes probability spaces, Pr [$p(x, y, \ldots) \mid x \xleftarrow{R} S, y \xleftarrow{R} T, \ldots$] denotes the probability that the predicate, $p(x, y, \ldots)$, is true after the experiments, $x \xleftarrow{R} S, y \xleftarrow{R} T, \ldots$, are executed in that order.

A function $\mu(k)$ is called *negligible* if for any strictly positive polynomial $p(k)$ there exist k_0 such that for all $k > k_0$ we have $\mu(k) < 1/p(k)$. We often write $negl(k)$ to indicate some negligible function of k, without giving an explicit name.

In this work we assume only *polytime* algorithms i.e. algorithms that their running time is polynomially bounded in the length of their inputs. We let 1^k denote the string of 1's of length k. When a probabilistic algorithm A is given 1^k as an input, this suggests that A is allowed to work in time polynomial in k. Note that if an adversary $\mathcal{A}$ is polytime and an oracle O is also polytime then the composition $\mathcal{A}^{O(\cdot)}$ is also polytime.

2 Definition of Committing Encryption

We define a symmetric committing encryption scheme, which uses a secret key shared by the sender and recipient and an asymmetric committing encryption scheme, which uses a public-key known to the sender and a corresponding private-key known only to the recipient.

SYNTAX OF SYMMETRIC COMMITTING ENCRYPTION SCHEMES

The symmetric committing encryption scheme adds to the regular symmetric encryption scheme (1) a public key, which is used for commitment purposes and (2) a public verification function.

The verification function confirms that message m is the original plaintext of committing ciphertext c, and that recipient of committing ciphertext c is able to open it i.e., the message m was encrypted with the original encryption key.

A *symmetric committing encryption scheme* $\mathcal{CE}_{sym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) consists of four algorithms:

- The randomized key generation algorithm *KeyGenSetup* takes as input a security parameter $k \in N$ and returns a pair of keys (CK, K). CK is the public commitment key (possibly empty, but usually consisting of public parameters for the commitment), and K is the encryption and decryption key, which is kept secret; we write $(CK, K) \xleftarrow{R} KeyGenSetup(1^k)$.
- The committing encryption algorithm *CmtEnc* takes as input the keys (CK, K), a message m from the associated message space $\mathcal{M}$ and a random value r, and returns committing ciphertext c; we write $c \leftarrow CmtEnc_{CK,K}(m, r)$.
- The deterministic de-committing decryption algorithm *DcmtDec* takes as input the keys (CK, K), the committing ciphertext c, and returns either a pair (m, *hint*), where $m \in \mathcal{M}$ is the corresponding plaintext and *hint* is a value to help the public verification of the committing encryption, or the symbol $\perp$ denoting failure; we write $(m, hint) \leftarrow DcmtDec_{CK,K}(c)$.
- The deterministic verification algorithm *Ver* takes as input the public key CK, the committing ciphertext c, a message m from the associated message space $\mathcal{M}$, and a *hint*, and returns an answer a which is either *succeed* in case the message m is the plaintext of c, or *fail* otherwise; we write $a \leftarrow Ver_{CK}(c, m, hint)$.

We require, for any $m \in \mathcal{M}$, any keys (CK, K), and any random value r,

- $DcmtDec_{CK,K}(CmtEnc_{CK,K}(m, r)) = (m, hint)$, and
- $Ver_{CK}(CmtEnc_{CK,K}(m, r), DcmtDec_{CK,K}(CmtEnc_{CK,K}(m, r))) = succeed$.

SYNTAX OF ASYMMETRIC COMMITTING ENCRYPTION SCHEMES

The asymmetric committing encryption scheme adds to the regular asymmetric encryption scheme the public verification function. The public encryption key is now used for commitment purposes also.

An *asymmetric committing encryption scheme* $\mathcal{CE}_{asym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) consists of four algorithms.

- The randomized key generation algorithm *KeyGenSetup* takes as input a security parameter $k \in N$ and outputs a pair of keys (CEK, DK). CEK is the public committing encryption key, and DK is the decryption key, which is kept secret; we write $(CEK, DK) \xleftarrow{R} KeyGenSetup(1^k)$.

- The committing encryption algorithm *CmtEnc* takes as input the public key *CEK*, a message *m* from the associated message space $\mathcal{M}$ and a random value *r*, and returns committing ciphertext *c*; we write $c \leftarrow CmtEnc_{CEK}(m, r)$.
- The deterministic de-committing decryption algorithm *DcmtDec* takes as input the keys (*CEK*, *DK*), the committing ciphertext *c*, and returns either a pair (*m*, *hint*), where $m \in \mathcal{M}$ is the corresponding plaintext and *hint* is a value to help the public verification of the committing encryption, or the symbol $\perp$ denoting failure; we write $(m, hint) \leftarrow DcmtDec_{CEK,DK}(c)$.
- The deterministic verification algorithm *Ver* takes as input the public key *CEK*, the committing ciphertext *c*, a message *m* from the associated message space $\mathcal{M}$, and a *hint*, and returns an answer *a* which is either *succeed* in case the message *m* is the plaintext of *c*, or *fail* otherwise; we write $a \leftarrow Ver_{CEK}(c, m, hint)$.

We require, for any $m \in \mathcal{M}$, any keys (*CEK*, *DK*) and any random value *r*,

- $DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r)) = (m, hint)$, and
- $Ver_{CEK}(CmtEnc_{CEK}(m, r), DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r))) = succeed$.

SECURITY OF COMMITTING ENCRYPTION

For brevity and since the definitions of the security notions are very similar for symmetric and asymmetric committing encryptions we define them simultaneously for both symmetric and asymmetric committing encryptions.

Both schemes consist of four algorithms each. It is clear that the sender runs *CmtEnc* algorithm and the recipient runs *DcmtDec* algorithm; anyone can run *Ver*. In the asymmetric scheme the recipient runs *KeyGenSetup* because only the recipient knows the secret decryption key *DK*. In the symmetric scheme, although it is symmetric, since the sender is also the committing party, we again require the recipient to run *KeyGenSetup* and choose the keys. This eliminates the possibility that the sender selects *CK* in such a way that it is possible for the committing party to find collisions i.e., to commit itself to message *m* with *c* and *hint*, but reveal to another message *m*' with *c* and *hint*'.

We consider three security goals:

- Indistinguishability of ciphertexts (IND)
- Binding
- Recoverability

Indistinguishability of ciphertexts (IND) Intuitively we want that given a committing ciphertext *c* an adversary cannot gain significant information about the message content. We assume an adversary that runs in two stages. During the *find* stage, the adversary endeavors to come up with a pair of equal-length messages, x_0 and x_1, whose committing encryptions it wants to try to tell apart. It also retains some state information *s* that it may want to preserve to help it later. In the *guess* stage, it is given a random committing ciphertext *y* for one of the plaintexts x_0, x_1, together with the state information *s*. The adversary "wins" if it correctly identifies which plaintext goes with *y*. The committing encryption scheme is "good" if "reasonable" adversaries cannot win significantly more than half the time.

We consider the following types of attack:

- Chosen Plaintext Attack (CPA)

- Chosen Ciphertext Attack (CCA)

Chosen Plaintext Attack (CPA) For the symmetric committing encryption, under the chosen plaintext attack, the adversary $\mathcal{A}_{\text{cpa}}$ is given access to the committing encryption oracle $CmtEnc_{CK,K}(\cdot)$ to commit-encrypt arbitrary messages of its choice. While for the asymmetric committing encryption, the adversary $\mathcal{A}_{\text{cpa}}$ is not given any extra capabilities other than using the public committing encryption key *CEK*. We denote this notion of security CE-IND-CPA.

Definition 1.1, 1.2 [CE_{sym}-IND-CPA-security, CE_{asym}-IND-CPA-security]

Let $\mathcal{CE}_{sym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be a symmetric committing encryption scheme. Let $\mathcal{CE}_{asym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be an asymmetric committing encryption scheme. Let $b \xleftarrow{R} \{0,1\}$ and $k \in N$. Let $\mathcal{A}_{\text{cpa}}$ be an adversary that runs in two stages, *find* and *guess*, and for symmetric committing encryption has access to the oracle $CmtEnc_{CK,K}(\cdot)$. We consider the following experiments:

Experiment $Exp^{\text{ind-cpa-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cpa}}}(k)$	Experiment $Exp^{\text{ind-cpa-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cpa}}}(k)$
$(CK, K) \xleftarrow{R} KeyGenSetup(1^k)$	$(CEK, DK) \xleftarrow{R} KeyGenSetup(1^k)$
$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}_{\text{cpa}}^{CmtEnc_{CK,K}(\cdot)}(CK, 1^k, find)$	$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}_{\text{cpa}}(CEK, find)$
$y \leftarrow CmtEnc_{CK,K}(x_b, r)$	$y \leftarrow CmtEnc_{CEK}(x_b, r)$
$b' \xleftarrow{R} \mathcal{A}_{\text{cpa}}^{CmtEnc_{CK,K}(\cdot)}(CK, 1^k, guess, y, s)$	$b' \xleftarrow{R} \mathcal{A}_{\text{cpa}}(CEK, guess, y, s)$
Return *b*'	Return *b*'

The schemes $\mathcal{CE}_{sym}$, $\mathcal{CE}_{asym}$ are said to be CE-IND-CPA-secure if (1.1) and (1.2) holds respectively:

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cpa-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cpa}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{1.1}$$

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cpa-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cpa}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{1.2}$$

Chosen Ciphertext Attack (CCA) For the symmetric committing encryption, under the chosen ciphertext attack, the adversary $\mathcal{A}_{\text{cca}}$ is given access to the committing encryption oracle $CmtEnc_{CK,K}(\cdot)$ to commit encrypt arbitrary messages of its choice and to the de-committing decryption oracle $DcmtDec_{CK,K}(\cdot)$ to de-commit decrypt arbitrary committing ciphertexts of its choice.

For the asymmetric committing encryption, under the chosen ciphertext attack, the adversary $\mathcal{A}_{\text{cca}}$ is given access to the de-committing decryption oracle $DcmtDec_{CEK,DK}(\cdot)$ to de-commit decrypt arbitrary committing ciphertexts of its choice.

Several types of CCA attacks were defined for asymmetric encryption schemes: CCA1 [NY95], CCA2 [RS91], and gCCA2 [ADR02]. Similar definitions apply to symmetric and asymmetric committing encryption schemes. We denote these notions of security CE-IND-CCA1, CE-IND-CCA2, and CE-IND-gCCA2.

Definition 2.1, 2.2 [CE_{sym}-IND-CCA1-security, CE_{asym}-IND-CCA1-security]

Let $\mathcal{CE}_{sym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be a symmetric committing encryption scheme. Let $\mathcal{CE}_{asym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be an asymmetric

committing encryption scheme. Let $b \xleftarrow{R} \{0,1\}$ and $k \in N$. Let $\mathcal{A}_{\text{cca1}}$ be an adversary that runs in two stages, *find* and *guess*, and for symmetric committing encryption has access to the oracles $CmtEnc_{CK,K}(\cdot)$ and $DcmtDec_{CK,K}(\cdot)$ in the *find* stage and to the oracle $CmtEnc_{CK,K}(\cdot)$ in the *guess* stage. And for asymmetric committing encryption, has access to the oracle $DcmtDec_{CEK,DK}(\cdot)$ in the *find* stage. We consider the following experiments:

Experiment $Exp^{\text{ind-cca1-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cca1}}}(k)$

$(CK, K) \xleftarrow{R} KeyGenSetup(1^k)$

$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}_{\text{cca1}}^{CmtEnc_{CK,K}(\cdot), DcmtDec_{CK,K}(\cdot)}(CK, 1^k, find)$

$y \leftarrow CmtEnc_{CK,K}(x_b, r)$

$b' \xleftarrow{R} \mathcal{A}_{\text{cca1}}^{CmtEnc_{CK,K}(\cdot)}(CK, 1^k, guess, y, s)$

Return b'

Experiment $Exp^{\text{ind-cca1-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cca1}}}(k)$

$(CEK, DK) \xleftarrow{R} KeyGenSetup(1^k)$

$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}_{\text{cca1}}^{DcmtDec_{CEK,DK}(\cdot)}(CEK, find)$

$y \leftarrow CmtEnc_{CEK}(x_b, r)$

$b' \xleftarrow{R} \mathcal{A}_{\text{cca1}}(CEK, guess, y, s)$

Return b'

The schemes $\mathcal{CE}_{sym}$, $\mathcal{CE}_{asym}$ are said to be CE-IND-CCA1-secure if (2.1) and (2.2) holds respectively:

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cca1-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cca1}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{2.1}$$

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cca1-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cca1}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{2.2}$$

Definition 3.1, 3.2 [CE_{sym}-IND-CCA2-security, CE_{sym}-IND-gCCA2-security, CE_{asym}-IND-CCA2-security CE_{asym}-IND-gCCA2-security]

Let $\mathcal{CE}_{sym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be a symmetric committing encryption scheme. Let $\mathcal{CE}_{asym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be an asymmetric committing encryption scheme. Let $b \xleftarrow{R} \{0,1\}$ and $k \in N$. Let $\mathcal{A}_{\text{cca2}}$ be an adversary that runs in two stages, *find* and *guess*, and for symmetric committing encryption has access to the oracles $CmtEnc_{CK,K}(\cdot)$ and $DcmtDec_{CK,K}(\cdot)$. And for asymmetric committing encryption, has access to the oracle $DcmtDec_{CEK,DK}(\cdot)$.

Let $\mathcal{R}(y, y') = \text{true} \Rightarrow y = y'$ for CCA2. For gCCA2, for symmetric committing encryption $\mathcal{R}(y, y') = \text{true} \Rightarrow DcmtDec_{CK,K}(y) = DcmtDec_{CK,K}(y')$, and for asymmetric committing encryption, $\mathcal{R}(y, y') = \text{true} \Rightarrow DcmtDec_{CEK,DK}(y) = DcmtDec_{CEK,DK}(y')$. $\mathcal{A}_{\text{cca2}}$, in the *guess* stage, is not allowed to ask oracle $DcmtDec_{CK,K}(\cdot)$ or $DcmtDec_{CEK,DK}(\cdot)$ a query y' s.t. $\mathcal{R}(y, y') = \text{true}$. We consider the following experiments:

Experiment $Exp^{\text{ind-cca2-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cca2}}}(k)$

$(CK, K) \xleftarrow{R} KeyGenSetup(1^k)$

$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}^{CmtEnc_{CK,K}(\cdot), DcmtDec_{CK,K}(\cdot)}_{\text{cca2}}(CK, 1^k, find)$

$y \leftarrow CmtEnc_{CK,K}(x_b, r)$

$b' \xleftarrow{R} \mathcal{A}^{CmtEnc_{CK,K}(\cdot), DcmtDec_{CK,K}(Z)|_{\forall z \in Z, \mathcal{R}(y,z)=false}}_{\text{cca2}}(CK, 1^k, guess, y, s)$

Return b'

Experiment $Exp^{\text{ind-cca2-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cca2}}}(k)$

$(CEK, DK) \xleftarrow{R} KeyGenSetup(1^k)$

$(x_0, x_1, s) \xleftarrow{R} \mathcal{A}^{DcmtDec_{CEK,DK}(\cdot)}_{\text{cca2}}(CEK, find)$

$y \leftarrow CmtEnc_{CEK}(x_b, r)$

$b' \xleftarrow{R} \mathcal{A}^{DcmtDec_{CEK,DK}(Z)|_{\forall z \in Z, \mathcal{R}(y,z)=false}}_{\text{cca2}}(CEK, guess, y, s)$

Return b'

The schemes $\mathcal{CE}_{sym}$, $\mathcal{CE}_{asym}$ are said to be CE-IND-CCA2-secure if (3.1) and (3.2) holds respectively:

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cca2-}b}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{cca2}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{3.1}$$

$$\Pr\left[b' = b \mid b \xleftarrow{R} \{0,1\}, b' \xleftarrow{R} Exp^{\text{ind-cca2-}b}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{cca2}}}(k)\right] \leq \frac{1}{2} + negl(k) \tag{3.2}$$

Binding Intuitively we want that an adversary $\mathcal{A}_{\text{bnd}}$ cannot find a committing ciphertext c, which can be publicly verified for two different messages m, and m' using possibly two different hints *hint*, and *hint*'.

Having the knowledge of (CK, K) (for the symmetric committing encryption) or CEK, (for the asymmetric committing encryption), it is computationally hard for an adversary $\mathcal{A}_{\text{bnd}}$ to come up with c, (m, *hint*), (m', *hint*'), $m' \neq m$ such that giving (c, m, *hint*) and (c, m', *hint*') as input to the verification algorithm *Ver* results in *succeed* (such a triple c, (m, *hint*), (m', *hint*') is said to cause a collision). That is, $\mathcal{A}_{\text{bnd}}$ cannot find a value c, which it can open in two different ways. We denote this notion of security CE-BIND.

Definition 4.1, 4.2 [CE_{sym}-BIND-security, CE_{asym}-BIND-security]

Let $\mathcal{CE}_{sym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be a symmetric committing encryption scheme. Let $\mathcal{CE}_{asym}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be an asymmetric committing encryption scheme. Let $k \in N$ and $\mathcal{A}_{\text{bnd}}$ be an adversary that runs in *find* stage. Let *collision*(c, (m, *hint*), (m', *hint*')) = true $\Leftrightarrow m \neq m' \wedge Ver_{CK}(c, m, hint) = succeed \wedge Ver_{CK}(c, m', hint') = succeed$ for symmetric committing encryption, and for asymmetric committing encryption, *collision*(c, (m, *hint*), (m', *hint*')) = true $\Leftrightarrow m \neq m' \wedge Ver_{CEK}(c, m, hint) = succeed \wedge Ver_{CEK}(c, m', hint') = succeed$. We consider the following experiments:

Experiment $Exp^{\text{bind}}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{bnd}}}(k)$ $(CK,K) \xleftarrow{R} KeyGenSetup(1^k)$ $(c,(m,hint),(m',hint')) \xleftarrow{R} \mathcal{A}_{\text{bnd}}(CK,K,find)$ Return (c, (m, *hint*), (m', *hint*'))
Experiment $Exp^{\text{bind}}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{bnd}}}(k)$ $(CEK,DK) \xleftarrow{R} KeyGenSetup(1^k)$ $(c,(m,hint),(m',hint')) \xleftarrow{R} \mathcal{A}_{\text{bnd}}(CEK,find)$ Return (c, (m, *hint*), (m', *hint*'))

The schemes $\mathcal{CE}_{sym}$, $\mathcal{CE}_{asym}$ are said to be CE-BIND-secure if (4.1) and (4.2) holds respectively:

$$\Pr\left[\begin{array}{l} collision(c,(m,hint),(m',hint')) \\ = true \end{array} \middle| (c,(m,hint),(m',hint')) \xleftarrow{R} Exp^{\text{bind}}_{\mathcal{CE}_{sym},\mathcal{A}_{\text{bnd}}}(k)\right] \leq negl(k) \quad (4.1)$$

$$\Pr\left[\begin{array}{l} collision(c,(m,hint),(m',hint')) \\ = true \end{array} \middle| (c,(m,hint),(m',hint')) \xleftarrow{R} Exp^{\text{bind}}_{\mathcal{CE}_{asym},\mathcal{A}_{\text{bnd}}}(k)\right] \leq negl(k) \quad (4.2)$$

Recoverability Intuitively we want that an adversary $\mathcal{A}_{\text{rcvr}}$ cannot produce committing ciphertext c in such a way that it cannot be de-committed decrypted by the other side nevertheless the adversary itself can produce m and *hint* such that $Ver(c, m, hint) = succeed$. In other words, changing secret key (i.e. using another secret key than the agreed upon one) is not possible.

Indeed, this security goal might be trivial for asymmetric committing encryption since *Ver* uses the public committing encryption key, which is bound to the private decryption key. But for symmetric committing encryption this kind of attack is still possible unless we have some kind of bindings between the public committing key and the secret encryption/decryption key. For the uniformity of the definition, we require this security goal for both symmetric and asymmetric committing encryptions. We denote this notion of security CE-RECOVER.

Definition 5.1, 5.2 [CE_{sym}-RECOVER-security, CE_{asym}-RECOVER-security]

Formally, having the knowledge of (CK, K) for symmetric committing encryption, it is computationally hard for an adversary $\mathcal{A}_{\text{rcvr}}$ to come up with c, m, and *hint* such that $DcmtDec_{CK,K}(c) = \perp$ but $Ver_{CK}(c, m, hint) = succeed$.

$$\Pr\left[\begin{array}{c} DcmtDec_{CK,K}(c) = \perp, \\ Ver_{CK}(c,m,hint) = succeed \end{array} \middle| \begin{array}{l} (CK,K) \xleftarrow{R} KeyGenSetup(1^k), \\ (c,m,hint) \xleftarrow{R} \mathcal{A}_{\text{rcvr}}(CK,K,find) \end{array}\right] \leq negl(k) \qquad (5.1)$$

And for asymmetric committing encryption, having the knowledge of CEK, it is computationally hard for an adversary $\mathcal{A}_{\text{rcvr}}$ to come up with c, m, and *hint* such that $DcmtDec_{CEK,DK}(c) = \perp$ but $Ver_{CEK}(c, m, hint) = succeed$.

$$\Pr\left[\begin{array}{c} DcmtDec_{CEK,DK}(c) = \perp, \\ Ver_{CEK}(c,m,hint) = succeed \end{array} \middle| \begin{array}{l} (CEK,DK) \xleftarrow{R} KeyGenSetup(1^k), \\ (c,m,hint) \xleftarrow{R} \mathcal{A}_{\text{rcvr}}(CEK, find) \end{array}\right] \leq negl(k) \qquad (5.2)$$

3 Committing Encryption from Randomness-Recovering Asymmetric Encryption

In this section we propose a simple method for constructing an asymmetric committing encryption scheme. This construction formalizes the common intuition to use encryption to provide also commitment along with confidentiality. Our construction is based on a random recovery asymmetric encryption scheme where decryption restores the random value used for encryption e.g. RSA with padding e.g. using OAEP (Optimal Asymmetric Encryption Padding) [BR94, PKCS1v2.0, PKCS1v2.1]. Having the randomness-recovering property enables the public verification algorithm *Ver* to publicly verify that (a) the message *m* is the original message and (b) the recipient of the committing ciphertext *c* is able to open it and restore the message *m* and the correct *hint*.

CONSTRUCTION OF ASYMMETRIC COMMITTING ENCRYPTION

Let $\mathcal{E}$ = (*EncKeyGen*, *Enc*, *Dec*) be a randomness-recovering asymmetric encryption scheme. Define an asymmetric committing encryption scheme $\mathcal{CE}_{RR}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) as follows:

- *KeyGenSetup*(1^k) runs $(EK,DK) \xleftarrow{R} EncKeyGen(1^k)$, sets *CEK* = *EK*, and outputs (*CEK*, *DK*)
- $CmtEnc_{CEK}(m, r)$ simply outputs $c \leftarrow Enc_{EK}(m, r)$
- $DcmtDec_{CEK,DK}(c)$ runs $(m, r) \leftarrow Dec_{DK}(c)$, sets *hint* = *r*, and outputs (*m*, *hint*)
- $Ver_{CEK}(c, m, hint)$ parse *r* = *hint*, computes $c' \leftarrow Enc_{EK}(m, r)$, and outputs *succeed* if *c* = *c*' or *fail* otherwise

Theorem 1 Assume that $\mathcal{E}$ satisfies the syntactic properties of a randomness-recovering asymmetric encryption scheme. Let $\mathcal{CE}_{RR}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$ as defined above. Then we have:

(1) $\mathcal{CE}_{RR}$ satisfies the syntactic properties of an asymmetric committing encryption scheme.

(2) $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2, E-IND-CPA, E-IND-CCA1) -secure $\Leftrightarrow$ $\mathcal{CE}_{RR}$ is CE-IND-CCA2 (CE-IND-gCCA2, CE-IND-CPA, CE-IND-CCA1) -secure.

(3) $\mathcal{CE}_{RR}$ is CE-BIND-secure.

(4) $\mathcal{CE}_{RR}$ is CE-RECOVER-secure.

The proof for this theorem is given in appendix A.

4 CtE: Composing Asymmetric Encryption and Commitment

We now present a new composition that we call "*commit-then-encrypt*". This composition involves several cryptographic tools such as asymmetric encryption, commitment and digital signature schemes, yet it is fairly efficient in the run time. We prove that this simple and efficient composition constructs a secure asymmetric committing encryption scheme. This

construction may be used as a mechanism to transform any asymmetric encryption (even if the decryption function does not restore the random value used by the encryption function) to committing encryption.

COMMIT-THEN-ENCRYPT (*CtE*) COMPOSITION

As the name suggests we compose encryption and commitment schemes. In *CmtEnc*, the committing encryption function, a message *m* first passes a commitment stage which produces $c_c(m)$ and $d(m)$. Then, in the second stage, the de-commitment $d(m)$ is encrypted to produce ciphertext c_e. Together, (c_c, c_e) form the committing ciphertext *c*. The process of *DcmtDec*, the de-committing decryption function, is exactly the reverse of *CmtEnc* process. *DcmtDec* outputs the message *m* and the de-commitment $d(m)$ as *hint*.

In *Ver*, the verification function, it seems that we have a problem. We require the public verification algorithm *Ver* to ensure not only that

(a) message *m* is the original plaintext of committing ciphertext *c*, but also that

(b) recipient of committing ciphertext *c* is able to open it with *DcmtDec* i.e., the message *m* was encrypted with the original encryption key.

In the *CtE* composition, the public committing encryption key *CEK* is composed of two sub-keys (*CK*, *EK*) such that there is no relation between them. Thus, key-spoofing attack is still possible.

Indeed, if the recipient gives the input parameters for *Ver*, we can assume that the recipient of committing ciphertext *c* was able to open it with *DcmtDec* i.e., the message *m* was encrypted with the original encryption key. Thus, it is sufficient for *Ver* to ensure only (a) i.e., that message *m* is the original plaintext of committing ciphertext *c*. But if the sender gives the input parameters for *Ver*, key-spoofing attack is still possible and thus we require *Ver* to ensure both (a) and (b).

In order to fix this problem we use a digital signature scheme to distinguish that the receiver gave the input parameters for *Ver*. For that, we require *KeyGenSetup* to generate also signing keys (*SK*, *VK*) for the recipient. The signing key *SK* is kept secret and is used only by the recipient to sign the de-commitment $d(m)$ given by the recipient to *Ver* algorithm as *hint*. *Ver* algorithm uses the public verification key *VK* to validate the signature of *hint*. If *Ver* succeeds to verify the signature, it retrieves the signed de-commitment $d(m)$ from *hint* and uses *Reveal*(c_c, *d*) to retrieve the original message *m*.

If *Ver* fails to verify the signature of *hint* then the sender gave the input parameters for *Ver*. The sender will set the parameter *hint* to the random value *r* used to encrypt message *m*. *Ver* algorithm will use this to repeat the encryption process and ensure both (a) and (b).

Therefore, *Ver* algorithm now runs differently depending whether the recipient or the sender gave the input parameters. *Ver* algorithm determines, independently, who gave the input parameters, using the *hint* parameter. The *hint* parameter that *Ver* algorithm receives is different from the sender and the recipient. If *Ver* algorithm succeeds to verify the signature of *hint* then the recipient gave the input parameters, otherwise the sender gave them.

Note that the recipient does not need to sign the de-commitment $d(m)$ unless the recipient needs to use *Ver*, and usually this is not the case. So implementation can divide *DcmtDec* into two steps. The first step (retrieving de-commitment $d(m)$ by decrypting c_e and then *Reveal*(c_c, *d*) to retrieve the original message *m*) is very efficient and is done always and in *real time*. The second step (signing the de-commitment $d(m)$) is done only *offline* or in exception cases

when the recipient needs to use *Ver*. Thus, *CmtEnc* and *DcmtDec* are both very efficient in the usual case.

CONSTRUCTION OF CtE COMPOSITION

Let $\mathcal{E}$ = (*EncKeyGen*, *Enc*, *Dec*) be an asymmetric encryption scheme, $\mathcal{C}$ = (*KeySetup*, *Commit*, *Decommit*, *Reveal*) be a commitment scheme, and $\mathcal{S}$ = (*SigKeyGen*, *Sign*, *Msg*, *SigVer*) be a digital signature scheme. Define an asymmetric committing encryption scheme $\mathcal{CE}_{CtE}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) as follows:

- *KeyGenSetup*(1^k) runs $(EK, DK) \xleftarrow{R} EncKeyGen(1^k)$, $(SK, VK) \xleftarrow{R} SigKeyGen(1^k)$, $CK \xleftarrow{R} KeySetup(1^k)$, sets *CEK* = (*CK*, *EK*, *VK*), *DK*' = (*DK*, *SK*), and outputs (*CEK*, *DK*')
- $CmtEnc_{CEK}(m, r_c \| r_e)$ runs $c_c \leftarrow Commit_{CK}(m, r_c)$, $d \leftarrow Decommit_{CK}(m, r_c)$, $c_e \leftarrow Enc_{EK}(d, r_e)$, and outputs $c = (c_c, c_e)$
- $DcmtDec_{CEK,DK'}(c)$ parse $(c_c, c_e) = c$, runs $d \leftarrow Dec_{DK}(c_e)$, $m \leftarrow Reveal_{CK}(c_c, d)$, $s \xleftarrow{R} Sign_{SK}(d)$, sets *hint* = *s*, and outputs (*m*, *hint*)
- $Ver_{CEK}(c, m, hint)$ parse $s = hint$, runs $a \leftarrow SigVer_{VK}(s)$ if $a = succeed$ (input parameters from the recipient) then parse $(c_c, c_e) = c$, runs $d \leftarrow Msg_{VK}(s)$, $m' \leftarrow Reveal_{CK}(c_c, d)$, and outputs *succeed* if $m = m'$ or *fail* otherwise. If $a = fail$ (input parameters from the sender) parse $r_c \| r_e = hint$, computes $c' \leftarrow CmtEnc_{CEK}(m, r_c \| r_e)$, and outputs *succeed* if $c = c'$ or *fail* otherwise.

Theorem 2 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then we have:

(1) $\mathcal{CE}_{CtE}$ satisfies the syntactic properties of an asymmetric committing encryption scheme.

(2) $\mathcal{C}$ satisfies the binding property $\Leftrightarrow$ $\mathcal{CE}_{CtE}$ is CE-BIND-secure.

(3) $\mathcal{S}$ is UF-NMA-secure $\Rightarrow$ $\mathcal{CE}_{CtE}$ is CE-RECOVER-secure.

The proof for this theorem is given in appendix A.

Theorem 3 Assume that $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2, E-IND-CPA, E-IND-CCA1) -secure, $\mathcal{C}$ and $\mathcal{S}$ satisfy the syntactic properties of a commitment scheme and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then we have:

(1) $\mathcal{CE}_{CtE}$ is CE-IND-CCA2 (CE-IND-gCCA2, CE-IND-CPA, CE-IND-CCA1) -secure $\Rightarrow$ $\mathcal{C}$ satisfies the hiding property.

(2) $\mathcal{C}$ satisfies the hiding and binding properties $\Rightarrow$ $\mathcal{CE}_{CtE}$ is CE-IND-CCA2 (CE-IND-gCCA2, CE-IND-CPA, CE-IND-CCA1) -secure.

The formal proof for this theorem is given in appendix A. Partial proof for this theorem (only for CE-IND-gCCA2-security) is given by [ADR02] in Appendix D. We expand their proof by proving CE-IND-CCA2-security also.

Intuitively, the hiding property is necessary since c_c is given "in the clear", and is sufficient since $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2, E-IND-CPA, E-IND-CCA1) -secure and there is at most one valid value c_c corresponding to every d.

5 Symmetric Committing Encryption with Key Setup Protocol

In this section we propose a simple method of constructing symmetric committing encryption scheme. This construction involves several cryptographic tools such as symmetric encryption, commitment and digital signature schemes, yet its runtime efficiency remains the same as of the symmetric encryption.

SYMMETRIC COMMITTING ENCRYPTION WITH KEY SETUP PROTOCOL

As the name suggests we simply use any symmetric encryption scheme. In order to construct symmetric committing encryption, we have to add the ability to publicly prove that (a) message m is the original plaintext of committing ciphertext c, and also that (b) recipient of committing ciphertext c is able to open it with *DcmtDec* i.e., the message m was encrypted with the original encryption key.

We can have both requirements publicly verifiable if *Ver*, the verification function, is able to run *Dec*, the decryption function of the symmetric encryption scheme. For that sake we need to expose the decryption key K that is kept secret by both communicating sides.

There are several problems with this suggestion. First, exposing the secret key K will expose also all the messages sent by both communicating sides. Second, since there is no public proof that the exposed secret key K is the original key, key-spoofing attack is still possible.

For the first problem, our suggestion is to produce n secret keys and to use each key K_i, $i = 1 .. n$, only once and only for one message. Now exposing one secret key K_i, for the verification of message m_i does not expose all other messages sent between the communicating sides. WLOG, we will assume that the index i can be determined from the message m (e.g., is part of it), and write $i = Index(m)$ to denote the index of message m.

For the second problem, our suggestion is to use commitment and digital signature schemes. Each side commits itself to the secret keys K_i, $i = 1 .. n$, during *KeyGenSetup*. The commitment $c(K_i)$ is publicly exposed. The de-commitment $d(K_i)$ is signed by each side and the signed de-commitment is kept secret, shared by both sides.

Now each side exposes its own public committing parameters. Denote user $P = R$ (for the recipient) and S (for the sender). User P exposes its public commitment key CK_P, the commitment $c(K_i)$, $i = 1 .. n$, denoted c_{iP}, and its public verification key VK_P. We combine all these parameters to be part of user P public committing key $CK'_P = (CK_P, c_{1P}, \ldots, c_{nP}, VK_P)$, and the public committing key is $CK = (CK'_R, CK'_S)$.

Each side has also some secret information that is shared with the other side. Both communicating sides share the encryption/decryption keys K_i, $i = 1 .. n$, and the signed de-commitment s_{iR} and s_{iS}, $i = 1 .. n$. The shared information is kept secret by both communicating sides. We combine all these parameters to be part of the secret encryption/decryption key $K = (K_1, \ldots, K_n, s_{1R}, \ldots, s_{nR}, s_{1S}, \ldots, s_{nS})$.

Note that since we do not need any more the signing keys of each party, we do not require *KeyGenSetup* to output them.

When user P comes to run *Ver*, we expect the *hint* parameter given to *Ver* to be the sender and the receiver signatures on the de-commitment to some K_i, i.e. $s_{iS}||s_{iR}$. The *Ver* algorithm

validates each signature, retrieves the de-commitment $d(K_i)$ from s_{iP} and uses $Reveal(c_{iP}, d_{iP})$ to retrieve the original key K_i. Now *Ver* is able to run *Dec*, the decryption function of the symmetric encryption scheme.

Note that the use of the signature and commitment is only at offline. During run time we use only the symmetric encryption scheme. Thus, the efficiency of this construction is the same as the selected symmetric encryption.

Note that we cannot use the original definition of symmetric committing encryption scheme. We must slightly modify it and add algorithms for key setup protocol as follows:

A *symmetric committing encryption scheme with key setup protocol* $\mathcal{CE}_{sym}$ = (*KeyGen*, *KeyCommitSetup*, *KeyCommit*, *CmtEnc*, *DcmtDec*, *Ver*) consists of six algorithms:

- The randomized key generation algorithm *KeyGen* takes as input a security parameter $k \in N$ and returns a key $K' \in KEYS$; we write $K' \xleftarrow{R} KeyGen(1^k)$.
- The randomized key commitment setup algorithm *KeyCommitSetup* takes as input a security parameter $k \in N$ and returns a public commitment key *CK*' (possibly empty, but usually consisting of public parameters for the commitment); we write $CK' \xleftarrow{R} KeyCommitSetup(1^k)$.
- The randomized key commitment algorithm *KeyCommit* takes as input a security parameter $k \in N$, a commitment key *CK*', and a key *K*'. It returns a triplet (*c*, *s*, *VK*) where *c* is *c*(*K*') i.e., a commitment to *K*' under the commitment key *CK*', *s* is the de-commitment *d*(*K*') signed with an internal generated signing key *SK*, and *VK* is the validation key for validating *s*. *c* and *VK* are public information, and *s* is kept secret (shared information); we write $(c, s, VK) \xleftarrow{R} KeyCommit(1^k, CK', K')$.

We put the public information to be *CK* = (*CK*', *c*, *VK*) and the shared information to be *K* = (*K*', *s*). The rest of the algorithms *CmtEnc*, *DcmtDec*, and *Ver* are the same as defined in the original definition of symmetric committing encryption scheme.

SECURITY OF SYMMETRIC COMMITTING ENCRYPTION SCHEMES WITH KEY SETUP PROTOCOL

The symmetric committing encryption scheme with key setup protocol consists of six algorithms. As in the original definition of symmetric committing encryption scheme, it is clear that the sender runs *CmtEnc* algorithm and the recipient runs *DcmtDec* algorithm. We also note that anyone can run *Ver* algorithm all alone, offline, and without any interaction with any of the communicating sides.

As opposed to the original definition of symmetric committing encryption scheme, for *KeyGen* algorithm, since it is a symmetric scheme, either the sender or the recipient can run *KeyGen* and choose the shared key.

We require both sides to run *KeyCommitSetup* and *KeyCommit* algorithms as follows: the recipient runs *KeyCommitSetup* to generate CK_R, and then the sender runs *KeyCommit* and uses CK_R as the input parameter *CK*'. The sender runs *KeyCommitSetup* to generate CK_S, and then the recipient runs *KeyCommit* and uses CK_S as the input parameter *CK*'.

Denoting the outputs of the sender's running of *KeyCommit* as (c_S, s_S, VK_S) and the outputs of the recipient's running of *KeyCommit* as (c_R, s_R, VK_R), we combine the outputs and put the public information to be $CK = (CK_R, c_R, VK_R, CK_S, c_S, VK_S)$ and the shared information to be $K = (K', s_R, s_S)$.

The rest of changes in the security notions definitions are trivial and follow from the syntax definition and thus omitted.

We only note that for the CE-BIND-security (Definition 4.1) we require also that the colliding triplet (c, (m, *hint*), (m', *hint*')) be produced s.t. m and m' have the same index i.

CONSTRUCTION OF SYMMETRIC COMMITTING ENCRYPTION

Let $\mathcal{E}$ = (*EncKeyGen*, *Enc*, *Dec*) be a symmetric encryption scheme, $\mathcal{C}$ = (*KeySetup*, *Commit*, *Decommit*, *Reveal*) be a commitment scheme, and $\mathcal{S}$ = (*SigKeyGen*, *Sign*, *Msg*, *SigVer*) be a digital signature scheme. Define a symmetric committing encryption scheme $\mathcal{CE}_{KSP}$ = (*KeyGen*, *KeyCommitSetup*, *KeyCommit*, *CmtEnc*, *DcmtDec*, *Ver*) as follows:

- *KeyGen*(1^k) for $i = 1 .. n$ times runs $K_i \xleftarrow{R} EncKeyGen(1^k)$, set $K' = (K_1, \ldots, K_n)$, and outputs K'
- *KeyCommitSetup*(1^k) simply outputs $CK' \xleftarrow{R} KeySetup(1^k)$

Denote the outputs of the sender's running of *KeyCommitSetup* as CK_S and the outputs of the recipient's running of *KeyCommitSetup* as CK_R.

- *KeyCommit*(1^k, CK', K') runs $(SK,VK) \xleftarrow{R} SigKeyGen(1^k)$, parse $(K_1, \ldots, K_n) = K'$, for $i = 1 .. n$ times runs (1) $c_i \leftarrow Commit_{CK'}(K_i, r)$, (2) $d_i \leftarrow Decommit_{CK'}(K_i, r)$, (3) $s_i \xleftarrow{R} Sign_{SK}(d_i)$, and outputs $(c_1, \ldots, c_n, s_1, \ldots, s_n, VK)$

Denote the outputs of the sender's running of *KeyCommit* as $(c_{1S}, \ldots, c_{nS}, s_{1S}, \ldots, s_{nS}, VK_S)$ and the outputs of the recipient's running of *KeyCommit* as $(c_{1R}, \ldots, c_{nR}, s_{1R}, \ldots, s_{nR}, VK_R)$.

We put the public information to be $CK = (CK_R, c_{1R}, \ldots, c_{nR}, VK_R, CK_S, c_{1S}, \ldots, c_{nS}, VK_S)$ and the shared information to be $K = (K_1, \ldots, K_n, s_{1R}, \ldots, s_{nR}, s_{1S}, \ldots, s_{nS})$.

- $CmtEnc_{CK,K}(m, r)$ retrieve $i = Index(m)$, and outputs $c \leftarrow Enc_{Ki}(m, r)$
- $DcmtDec_{CK,K}(c)$ retrieve $i = Index(m)$, runs $m \leftarrow Dec_{Ki}(c)$, sets $hint = s_{iS}||s_{iR}$, and outputs (m, *hint*)
- $Ver_{CK}(c, m, hint)$ parse $s_{iS}||s_{iR} = hint$, for each s_{iP} runs $a \leftarrow SigVer_{VKp}(s_{iP})$, if $a = fail$ return *fail* and stop. Otherwise, runs $d_{iP} \leftarrow Msg_{VKp}(s_{iP})$, for s_{iS} run $K_{iS} \xleftarrow{} Reveal_{CK_R}(c_{iS}, d_{iS})$, for s_{iR} run $K_{iR} \xleftarrow{} Reveal_{CK_S}(c_{iR}, d_{iR})$. If $K_{iS} \neq K_{iR}$ return *fail* and stop. Otherwise, runs $m' \leftarrow Dec_{Ki}(c)$, and outputs *succeed* if $m = m'$ or *fail* otherwise

Theorem 4 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of a symmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{KSP}$ be a symmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then we have:

(1) $\mathcal{CE}_{KSP}$ satisfies the syntactic properties of a symmetric committing encryption scheme.

(2) $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2, E-IND-CPA, E-IND-CCA1) -secure $\Leftrightarrow$ $\mathcal{CE}_{KSP}$ is CE-IND-CCA2 (CE-IND-gCCA2, CE-IND-CPA, CE-IND-CCA1) -secure.

(3) $\mathcal{C}$ satisfies the binding property $\Rightarrow$ $\mathcal{CE}_{KSP}$ is CE-BIND-secure.

(4) $\mathcal{C}$ satisfies the binding property $\Rightarrow$ $\mathcal{CE}_{KSP}$ is CE-RECOVER-secure.

The proof for this theorem is given in appendix A.

6 Commit-Encrypt-then-Sign (*CEtS*) Publicly Verifiable Signcryption

In this section we present a new scheme which we call *publicly verifiable signcryption* scheme. Publicly verifiable signcryption schemes capture the security properties of committing encryption and signature schemes and provide public verification to anyone and without the need for interaction with any side. Publicly verifiable signcryption schemes do not suffer from key-spoofing attacks and support non-repudiation.

We also present in this section a simple and efficient construction of *commit-encrypt-then-sign* (*CEtS*) composition that preserves security properties of both ingredients committing encryption and signature including the non-repudiation feature of the signature scheme.

PUBLICLY VERIFIABLE SIGNCRYPTION

An, Dodis and Rabin [ADR02] defined *signcryption* following Zheng [Zh97]. Their definition of signcryption lacks public verification. We present here formal syntax and security definitions for publicly verifiable signcryption schemes. Publicly verifiable signcryption schemes do not suffer from key-spoofing attacks and support non-repudiation.

A *publicly verifiable signcryption scheme* $\mathcal{SC}_{asym}$ = (*EncKeyGen*, *SigKeyGen*, *EncSign*, *VerDec*, *SCVer*) consists of five algorithms:

- The randomized key generation algorithm *EncKeyGen* takes as input a security parameter $k \in N$ and outputs a pair of keys (*CEK*, *DK*). *CEK* is the committing encryption key, which is made public, and *DK* is the decryption key, which is kept secret; we write $(CEK, DK) \xleftarrow{R} EncKeyGen(1^k)$.
- The randomized key generation algorithm *SigKeyGen* takes as input a security parameter $k \in N$ and outputs a pair of keys (*SK*, *VK*). *SK* is the signing key, which is kept secret, and *VK* is the verification key, which is made public. We write $(SK, VK) \xleftarrow{R} SigKeyGen(1^k)$.
- The signcryption algorithm *EncSign* takes as input the public key *CEK*, the signing key *SK*, a message *m* from the associated message space $\mathcal{M}$ and a random value *r*, and returns signcryption ciphertext *c*; we write $c \leftarrow EncSign_{CEK,SK}(m, r)$.
- The deterministic de-signcryption algorithm *VerDec* takes as input the keys (*CEK*, *DK*, *VK*), the signcryption ciphertext *c*, and returns either a pair (*m*, *hint*), where $m \in \mathcal{M}$ is the corresponding plaintext and *hint* is a value to help the public verification of the signcryption, or the symbol $\perp$ denoting failure; we write $(m, hint) \leftarrow VerDec_{CEK,DK,VK}(c)$.
- The deterministic signcryption verification algorithm *SCVer* takes as input the public verification key *VK*, the signcryption ciphertext *c*, a message *m* from the associated message space $\mathcal{M}$, and a *hint*, and returns an answer *a* which is either *succeed* in case the message *m* is the plaintext of *c*, or *fail* otherwise; we write $a \leftarrow SCVer_{VK}(c, m, hint)$.

We require, for any $m \in \mathcal{M}$, any keys (*CEK*, *DK*, *SK*, *VK*) and any random value *r*,

- $VerDec_{CEK,DK,VK}(EncSign_{CEK,SK}(m, r)) = (m, hint)$, and
- $SCVer_{VK}(EncSign_{CEK,SK}(m, r), VerDec_{CEK,DK,VK}(EncSign_{CEK,SK}(m, r))) = succeed.$

SECURITY OF PUBLICLY VERIFIABLE SIGNCRYPTION SCHEMES

Given any publicly verifiable signcryption scheme $\mathcal{SC}_{asym}$ = (*EncKeyGen*, *SigKeyGen*, *EncSign*, *VerDec*, *SCVer*), we define the corresponding induced asymmetric committing encryption scheme $\mathcal{CE}_{SC}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) and induced digital signature scheme $\mathcal{S}_{SC}$ = (*SigKeyGen*, *Sign*, *Msg*, *SigVer*).

INDUCED ASYMMETRIC COMMITTING ENCRYPTION SCHEME $\mathcal{CE}_{SC}$

For any adversarial signing/verification keys (*SK*, *VK*),

- $KeyGenSetup(1^k)$ simply outputs $(CEK, DK) \xleftarrow{R} EncKeyGen(1^k)$

We set the public committing encryption key *CEK*' = (*CEK*, *SK, VK*).

- $CmtEnc_{CEK'}(m, r)$ simply outputs $c \leftarrow EncSign_{CEK,SK}(m, r)$
- $DcmtDec_{CEK',DK}(c)$ simply outputs $(m, hint) \leftarrow VerDec_{CEK,DK,VK}(c)$
- $Ver_{CEK'}(c, m, hint)$ simply outputs $a \leftarrow SCVer_{VK}(c, m, hint)$

INDUCED DIGITAL SIGNATURE SCHEME $\mathcal{S}_{SC}$

For any adversarial committing encryption/decryption keys (*CEK*, *DK*),

- $SigKeyGen(1^k)$ simply outputs $(SK, VK) \xleftarrow{R} SigKeyGen(1^k)$

We set the public verification key *VK*' = (*CEK*, *DK*, *VK*) and the secret signing key *SK*' = (*CEK*, *SK*).

- $Sign_{SK'}(m)$ runs $c \leftarrow EncSign_{CEK,SK}(m, r)$, and outputs $s = c$
- $Msg_{VK'}(s)$ parse $c = s$, runs $(m, hint) \leftarrow VerDec_{CEK,DK,VK}(c)$, and outputs m
- $SigVer_{VK'}(s)$ parse $c = s$, runs $(m, hint) \leftarrow VerDec_{CEK,DK,VK}(c)$, and outputs $a \leftarrow SCVer_{VK}(c, m, hint)$

We say that the publicly verifiable signcryption $\mathcal{SC}$ is secure against the corresponding attack (e.g. gCCA2/BIND/RECOVER/CMA) on the privacy/non-ambiguity/viability/ authenticity property, if the corresponding induced committing encryption/signature is secure against the same attack. We will aim to satisfy CE-IND-gCCA2, CE-BIND, and CE-RECOVER -security for the induced committing encryption, and UF-CMA-security for the induced signature.

CONSTRUCTION OF COMMIT-ENCRYPT-THEN-SIGN (CETS) COMPOSITION

Let $\mathcal{CE}$ = (*KeyGenSetup*, *CmtEnc*, *DcmtDec*, *Ver*) be a secure asymmetric committing encryption scheme and $\mathcal{S}$ = (*SigKeyGen*, *Sign*, *Msg*, *SigVer*) be a secure digital signature scheme. Define a publicly verifiable signcryption scheme $\mathcal{SC}_{\mathcal{CE}tS}$ = (*EncKeyGen*, *SigKeyGen*, *EncSign*, *VerDec*, *SCVer*) as follows:

- $EncKeyGen(1^k)$ simply outputs $(CEK, DK) \xleftarrow{R} KeyGenSetup(1^k)$
- $SigKeyGen(1^k)$ simply outputs $(SK, VK) \xleftarrow{R} SigKeyGen(1^k)$
- $EncSign_{CEK,SK}(m, r)$ runs $c_{ce} \leftarrow CmtEnc_{CEK}(m, r)$, sets $c = c_{ce} \| CEK$, runs $s \xleftarrow{R} Sign_{SK}(c)$, and outputs $c_s = s$

- $VerDec_{CEK,DK,VK}(c_s)$ parse $s = c_s$, runs $a \leftarrow SigVer_{VK}(s)$, if $a = fail$ return $\perp$. Otherwise ($a = succeed$), runs $c \leftarrow Msg_{VK}(s)$, parse $c_{ce}||CEK = c$, and outputs $(m, hint) \leftarrow DcmtDec_{CEK,DK}(c_{ce})$
- $SCVer_{VK}(c_s, m, hint)$ parse $s = c_s$, runs $a \leftarrow SigVer_{VK}(s)$, if $a = fail$ return $fail$. Otherwise ($a = succeed$), runs $c \leftarrow Msg_{VK}(s)$, parse $c_{ce}||CEK = c$, and outputs $a \leftarrow Ver_{CEK}(c_{ce}, m, hint)$

SECURITY OF CETS COMPOSITION

We prove $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ security against the strongest security notion of the committing encryption $\mathcal{CE}$ and signature $\mathcal{S}$, i.e. CE-IND-gCCA2 and UF-CMA. Weaker notions e.g. CE-IND-CPA, CE-IND-CCA1, and UF-NMA could easily be proved as well.

Theorem 5 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then we have:

(1) $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ satisfies the syntactic properties of a publicly verifiable signcryption scheme.

(2) $\mathcal{CE}$ is CE-IND-gCCA2-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ is CE-IND-gCCA2-secure.

(3) $\mathcal{CE}$ is CE-BIND-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ is CE-BIND-secure.

(4) $\mathcal{CE}$ is CE-RECOVER-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ is CE-RECOVER-secure.

(5) $\mathcal{S}$ is UF-CMA-secure $\Rightarrow$ $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ is UF-CMA-secure.

The formal proof for this theorem is given in appendix A. We remark the crucial use of CE-IND-gCCA2-security when proving the security of $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$. Indeed, we can call two signcryption ciphertexts c_{s1} and c_{s2} equivalent for $\mathcal{CE}_{SC}$, if each c_{si} is a valid signature (w.r.t. $\mathcal{S}$) of $c_{cei}||CEK = Msg_{VK}(c_{si})$, and c_{ce1} and c_{ce2} are equivalent (e.g., equal) w.r.t. the equivalence relation of $\mathcal{CE}$. In other words, a different signature of the same committing encryption clearly corresponds to the same message, and we should not reward the adversary for achieving such a trivial task. The task is indeed trivial, since the adversary has the signing key.

7 Conclusion and Open Questions

Compared to commitment schemes and digital signatures, conventional notions of encryption schemes do not have public verification facility. Usually, there is no way one can prove to a third party that a message m is the plaintext of ciphertext c without exposing the secret key and thus exposing all other messages too (a special case is the randomness-recovering asymmetric encryption scheme). Committing encryption schemes have that ability. Having the public verification facility provide also non-repudiation.

Committing encryption schemes provide also the ability for the sender (encrypt side) to prove to a third party that the recipient (decrypt side) is able to open the ciphertext and retrieve the original plaintext. Having these properties, committing encryption schemes eliminate key-spoofing attacks and can be signed.

We have provided a formal definition of committing encryption schemes. The definition is essential for cryptanalysts. Given an encryption algorithm that is going to be used for commitment, cryptanalysts can analyze and check according to the committing encryption

definition whether the encryption algorithm can be used to provide commitment also along with confidentiality or not.

We presented several simple and efficient constructions of symmetric and asymmetric committing encryption schemes.

Contrary to Zheng [Zh97] and An, Dodis and Rabin [ADR02] *signcryption* scheme that is not publicly verifiable, we have presented in this work a *publicly verifiable signcryption* scheme that does not suffer from key-spoofing attacks and support non-repudiation. We provided a simple and efficient construction of *commit-encrypt-then-sign* (*CEtS*) composition that preserves security properties of both ingredients committing encryption and signature including the non-repudiation feature of the signature scheme.

In our construction of symmetric committing encryption scheme in section 5 we changed the definition of the symmetric committing encryption scheme from section 2 and add a key setup protocol. Is it possible to construct a symmetric committing encryption scheme without changing the definition of the symmetric committing encryption scheme i.e., use the *KeyGenSetup* function as defined originally in section 2?

References

[ADR02] Jee Hea An, Yevgeniy Dodis, Tal Rabin. *On the Security of Joint Signature and Encryption*. In L. Knudsen, editor, Advances in Cryptology EUROCRYPT 2002. http://citeseer.nj.nec.com/an02security.html

[AN95] Ross Anderson, Roger Needham. *Robustness Principles for Public Key Protocols*. In Proceedings of Int'l. Conference on Advances in Cryptology (CRYPTO 95), Vol. 963 of Lecture Notes in Computer Science, pp. 236-247, Springer-Verlag, 1995. http://citeseer.nj.nec.com/anderson95robustness.html

[BR94] Mihir Bellare, Phillip Rogaway. *Optimal Asymmetric Encryption – How to Encrypt with RSA*. In A. De Santis, editor, Advances in Cryptology – Eurocrypt '94, Vol. 950 of Lecture Notes in Computer Science, pp. 92-111, Springer Verlag, 1995. http://citeseer.nj.nec.com/bellare94optimal.html

[NY95] Moni Naor, Moti Yung. *Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attack*. Proceedings of the 22nd Annual Symposium on the Theory of Computing, ACM STOC, pp. 427-437, May 14-16, 1990. http://citeseer.nj.nec.com/naor95publickey.html

[PKCS1v2.0] RSA Laboratories. *PKCS #1 v2.0: RSA Encryption Standard*. October 1998. Available from http://www.rsasecurity.com/rsalabs/pkcs/

[PKCS1v2.1] RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard*. 14, June 2002. Available from http://www.rsasecurity.com/rsalabs/pkcs/

[RS91] C. Rackoff, D. Simon. *Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack*. Advances in Cryptology, CRYPTO '91, Lecture Notes in Computer Science, Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[ZG96a] Jianying Zhou, Dieter Gollmann. *A fair non-repudiation protocol*. In Proceedings of the IEEE Symposium on Research in Security and Privacy [IEE96], pp. 55-61, Oakland, CA, May 1996. http://citeseer.nj.nec.com/62704.html

[Zh97] Yuliang Zheng. *Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption)*. In Advances in Cryptology - CRYPTO'97, Berlin, New York, Tokyo, Vol. 1294 of LNCS, pp. 165--179, Springer-Verlag, 1997. http://citeseer.nj.nec.com/zheng97digital.html

Appendix A

PROOFS

Proof of Theorem 1 We prove $\mathcal{CE}_{RR}$ security against the strongest security notion of the encryption $\mathcal{E}$, i.e. E-IND-CCA2 and E-IND-gCCA2. Weaker notions e.g. E-IND-CPA and E-IND-CCA1 could easily be proved as well.

The theorem is immediately follows from Lemmas 1.1 – 1.4. ■

Lemma 1.1 Assume that $\mathcal{E}$ satisfies the syntactic properties of a randomness-recovering asymmetric encryption scheme. Let $\mathcal{CE}_{RR}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$ as defined above. Then $\mathcal{CE}_{RR}$ satisfies the syntactic properties of an asymmetric committing encryption scheme.

Proof From the syntax requirement of the randomness-recovering asymmetric encryption scheme $\mathcal{E}$, *Dec* deterministically recovers the plaintext m and the randomness r that were input to *Enc* i.e., $Dec_{DK}(Enc_{EK}(m, r)) = (m, r)$. From the definitions of *DcmtDec* and *CmtEnc* follows that $DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r)) = Dec_{DK}(Enc_{EK}(m, r)) = (m, r)$.

Let $c = CmtEnc_{CEK}(m, r)$ then from the definition of *CmtEnc* also $c = Enc_{EK}(m, r)$. From the definition of *Ver* follows that: $Ver_{CEK}(CmtEnc_{CEK}(m, r), DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r))) = Ver_{CEK}(c, m, r) = succeed$. ■

Lemma 1.2 Assume that $\mathcal{E}$ satisfies the syntactic properties of a randomness-recovering asymmetric encryption scheme. Let $\mathcal{CE}_{RR}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$ as defined above. Then

$\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2) -secure $\Leftrightarrow$ $\mathcal{CE}_{RR}$ is CE-IND-CCA2 (CE-IND-gCCA2) -secure.

Proof We prove CE-IND-CCA2-security and CE-IND-gCCA2-security of $\mathcal{CE}_{RR}$ simultaneously. For CE-IND-gCCA2-security, let $\mathcal{R}$ be the equivalence relation w.r.t. which $\mathcal{E}$ is secure. We define the equivalence relation for $\mathcal{CE}_{RR}$ to be $\mathcal{R}'(c_1, c_2)$ = true iff $\mathcal{R}(c_1, c_2)$ = true. For the uniformity and simplicity of the proof we define also for CE-IND-CCA2-security an equivalence relation for $\mathcal{CE}_{RR}$ to be $\mathcal{R}'(c_1, c_2)$ = true iff $c_1 = c_2$.

$\Rightarrow$ Assume adversary $\mathcal{A}'$ can break the CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{RR}$. We can easily construct adversary $\mathcal{A}$ that can break the E-IND-CCA2-security (E-IND-gCCA2-security) of $\mathcal{E}$. $\mathcal{A}$ runs $\mathcal{A}'$ internally, passing every de-committing decryption queries made by $\mathcal{A}'$ to its own decryption oracle. When $\mathcal{A}'$ outputs x_0, and x_1, $\mathcal{A}$ outputs them also. When $\mathcal{A}$ is presented with the challenge $c_b = Enc_{EK}(x_b, r)$ (for unknown b), it hands it to $\mathcal{A}'$ and continue running $\mathcal{A}'$ waiting to its answer. Now, the definition of $\mathcal{R}'$ tells us that $\mathcal{A}'$ is disallowed to de-commit decrypt any c satisfying $\mathcal{R}'(c_b, c)$ = true. But such c are the only queries that $\mathcal{A}$ itself is disallowed to ask its decryption oracle! Thus, $\mathcal{A}$ can still handle all the legal de-committing decryption queries of $\mathcal{A}'$, in the same manner as before. Finally, $\mathcal{A}$ outputs the same guess b' that $\mathcal{A}'$ outputs, which clearly gives $\mathcal{A}$ the same probability of being correct as $\mathcal{A}'$ has.

$\Leftarrow$ Assume adversary $\mathcal{A}$ can break the E-IND-CCA2-security (E-IND-gCCA2-security) of $\mathcal{E}$. We can easily construct adversary $\mathcal{A}'$ that can break the CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{RR}$. $\mathcal{A}'$ runs $\mathcal{A}$ internally, passing every decryption queries made by $\mathcal{A}$ to its own de-committing decryption oracle. When $\mathcal{A}$ outputs x_0, and x_1, $\mathcal{A}'$ outputs them

also. When $\mathcal{A}$' is presented with the challenge $c_b = CmtEnc_{CEK}(x_b, r)$ (for unknown b), it hands it to $\mathcal{A}$ and continue running $\mathcal{A}$ waiting to its answer. Now, the definition of $\mathcal{R}$' tells us that $\mathcal{A}$' is disallowed to de-commit decrypt any c satisfying $\mathcal{R}'(c_b, c) = \text{true}$. But such c are the only queries that $\mathcal{A}$ itself is disallowed to ask its decryption oracle! Thus, $\mathcal{A}$' can still handle all the legal de-committing decryption queries of $\mathcal{A}$, in the same manner as before. Finally, $\mathcal{A}$' outputs the same guess b' that $\mathcal{A}$ outputs, which clearly gives $\mathcal{A}$' the same probability of being correct as $\mathcal{A}$ has. ■

Lemma 1.3 Assume that $\mathcal{E}$ satisfies the syntactic properties of a randomness-recovering asymmetric encryption scheme. Let $\mathcal{CE}_{RR}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$ as defined above. Then $\mathcal{CE}_{RR}$ is CE-BIND-secure.

Proof Assume $\mathcal{CE}_{RR}$ is not CE-BIND-secure. We can show that $\mathcal{E}$ cannot be syntactically an encryption scheme. Let c, (m_1, $hint_1$), (m_2, $hint_2$) be a collision for $\mathcal{CE}_{RR}$. That is $Ver_{CEK}(c, m_1, hint_1) = Ver_{CEK}(c, m_2, hint_2) = succeed$ and $m_1 \neq m_2$. From the definition of *Ver* follows that $c = Enc_{EK}(m_1, r_1)$ and also $c = Enc_{EK}(m_2, r_2)$. But from the syntax requirement of the encryption scheme *Dec* deterministically recovers the plaintext i.e.,

$Dec_{DK}(c) = Dec_{DK}(Enc_{EK}(m_1, r_1)) = (m_1, r_1) \neq (m_2, r_2) = Dec_{DK}(Enc_{EK}(m_2, r_2)) = Dec_{DK}(c)$, contradiction. ■

Lemma 1.4 Assume that $\mathcal{E}$ satisfies the syntactic properties of a randomness-recovering asymmetric encryption scheme. Let $\mathcal{CE}_{RR}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$ as defined above. Then $\mathcal{CE}_{RR}$ is CE-RECOVER-secure.

Proof Assume $\mathcal{CE}_{RR}$ is not CE-RECOVER-secure. We can show that $\mathcal{E}$ cannot be syntactically an encryption scheme. Let c, m, and $hint$, be s.t. $Ver_{CEK}(c, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c) = \perp$. From the definition of *Ver* follows that $c = Enc_{EK}(m, r)$. From the definition of *DcmtDec* follows that $DcmtDec_{CEK,DK}(c) = Dec_{DK}(c) = Dec_{DK}(Enc_{EK}(m, r)) \neq (m, r)$, but from the syntax requirement of the encryption scheme *Dec* deterministically recovers the plaintext, contradiction. ■

Proof of Theorem 2 The theorem is immediately follows from Lemmas 2.1 – 2.3. ■

Recall that:

(i) From the syntax requirement of the asymmetric encryption scheme $\mathcal{E}$, *Dec* deterministically recovers the plaintext m that was the input to *Enc* i.e., $Dec_{DK}(Enc_{EK}(m)) = m$.

(ii) From the syntax requirement of the commitment scheme $\mathcal{C}$, *Reveal* deterministically recovers the message m that was the input to *Commit* and *Decommit* i.e., $Reveal_{CK}(Commit_{CK}(m, r), Decommit_{CK}(m, r)) = m$.

(iii) From the syntax requirement of the digital signature scheme $\mathcal{S}$, *Msg* deterministically recovers the message m that was signed by *Sign* i.e., $Msg_{VK}(Sign_{SK}(m)) = m$.

Lemma 2.1 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then $\mathcal{CE}_{CtE}$ satisfies the syntactic properties of an asymmetric committing encryption scheme.

Proof Let $c = CmtEnc_{CEK}(m, r_c||r_e)$. From the definition of *CmtEnc* we have $c = (c_c, c_e) = (Commit_{CK}(m, r_c), Enc_{EK}(Decommit_{CK}(m, r_c), r_e))$.

$(*)$ $Reveal_{CK}(Commit_{CK}(m, r_c), Msg_{VK}(Sign_{SK}(Decommit_{CK}(m, r_c)))) =_{\text{[using (iii)]}} Reveal_{CK}(Commit_{CK}(m, r_c), Decommit_{CK}(m, r_c)) =_{\text{[using (ii)]}} m$.

From the definitions of *DcmtDec* and *CmtEnc* follows that $DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r))$

$= (Reveal_{CK}(Commit_{CK}(m, r_c), Dec_{DK}(Enc_{EK}(Decommit_{CK}(m, r_c), r_e))), Sign_{SK}(Dec_{DK}(Enc_{EK}(Decommit_{CK}(m, r_c), r_e))))$

$=_{\text{[using (i)]}} (Reveal_{CK}(Commit_{CK}(m, r_c), Decommit_{CK}(m, r_c)), Sign_{SK}(Decommit_{CK}(m, r_c)))$
$=_{\text{[using (ii)]}} (m, Sign_{SK}(Decommit_{CK}(m, r_c))) = (m, hint)$.

From the definition of *Ver* (input parameters given by the recipient) follows that:

$Ver_{CEK}(CmtEnc_{CEK}(m, r), DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r))) = Ver_{CEK}(c, m, hint) = Ver_{CEK}(c, m, Sign_{SK}(Decommit_{CK}(m, r_c))) =_{\text{[using (*)]}} succeed.$ ■

Lemma 2.2 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{C}$ satisfies the binding property $\Leftrightarrow$ $\mathcal{CE}_{CtE}$ is CE-BIND-secure.

Proof

$\Rightarrow$ Assume adversary $\mathcal{A}$' can find collisions for $\mathcal{CE}_{CtE}$ i.e., $\mathcal{A}$' can find $c = (c_c, c_e)$, $(m, hint)$, $(m', hint')$ such that $Ver_{CEK}(c, m, hint) = succeed$, $Ver_{CEK}(c, m', hint') = succeed$ and $m \neq m'$.

Recall that $Ver_{CEK}(c, m, hint)$ has two options to succeed.

(a) Input parameters given by the sender: parse *hint* as $r_c||r_e$, check if $c = (c_c, c_e) = CmtEnc_{CEK}(m, r_c||r_e) = (Commit_{CK}(m, r_c), Enc_{EK}(Decommit_{CK}(m, r_c), r_e))$.

(b) Input parameters given by the recipient: parse *hint* as $Sign_{SK}(d)$, check if $m = Reveal_{CK}(Commit_{CK}(m, r_c), Msg_{VK}(Sign_{SK}(Decommit_{CK}(m, r_c))))$.

We now show that in either case (a) or (b), if $\mathcal{A}$' succeeds to find collisions for $\mathcal{CE}_{CtE}$, then either $\mathcal{E}$ cannot be syntactically an encryption scheme or $\mathcal{C}$ cannot be syntactically a commitment scheme, or we can find collisions for $\mathcal{C}$.

Proof of case (a) Assume $\mathcal{A}$' succeeds to find collisions for $\mathcal{CE}_{CtE}$ s.t. the first option for *Ver* success holds. We can show that either $\mathcal{E}$ cannot be syntactically an encryption scheme or $\mathcal{C}$ cannot be syntactically a commitment scheme, or we can find collisions for $\mathcal{C}$.

From the definition of *CmtEnc* we have:

$c = (c_c, c_e) = (Commit_{CK}(m, r_{c1}), Enc_{EK}(Decommit_{CK}(m, r_{c1}), r_{e1}))$ and also, $c = (c_c, c_e) = (Commit_{CK}(m', r_{c2}), Enc_{EK}(Decommit_{CK}(m', r_{c2}), r_{e2}))$.

Thus, $c_c = Commit_{CK}(m, r_{c1}) = Commit_{CK}(m', r_{c2})$, and $c_e = Enc_{EK}(Decommit_{CK}(m, r_{c1}), r_{e1}) = Enc_{EK}(Decommit_{CK}(m', r_{c2}), r_{e2})$.

Now, if $Decommit_{CK}(m, r_{c1}) = Decommit_{CK}(m', r_{c2})$ then, denote $d = Decommit_{CK}(m, r_{c1}) = Decommit_{CK}(m', r_{c2})$, we have:

$Reveal_{CK}(c_c, d) = Reveal_{CK}(Commit_{CK}(m, r_{c1}), Decommit_{CK}(m, r_{c1})) = m \neq m' = Reveal_{CK}(Commit_{CK}(m', r_{c2}), Decommit_{CK}(m', r_{c2})) = Reveal_{CK}(c_c, d)$, contradiction.

Otherwise, denote $d_1 = Decommit_{CK}(m, r_{c1})$ and $d_2 = Decommit_{CK}(m', r_{c2})$, we have: $Dec_{DK}(c_e) = Dec_{DK}(Enc_{EK}(d_1, r_{c1})) = d_1 \neq d_2 = Dec_{DK}(Enc_{EK}(d_2, r_{c2})) = Dec_{DK}(c_e)$, contradiction.

Moreover, (c_c, d_1, d_2) is a collision for $\mathcal{C}$, since:

$Reveal_{CK}(c_c, d_1) = Reveal_{CK}(Commit_{CK}(m, r_{c1}), Decommit_{CK}(m, r_{c1})) = m \neq m' = Reveal_{CK}(Commit_{CK}(m', r_{c2}), Decommit_{CK}(m', r_{c2})) = Reveal_{CK}(c_c, d_2)$, contradiction.

Proof of case (b) If $\mathcal{A}'$ succeeds with the second option of *Ver*, we can easily construct adversary $\mathcal{A}$ that can find collisions for $\mathcal{C}$. $\mathcal{A}$ views the commitment key *CK* and by itself picks a pair of encryption/decryption keys $(EK, DK) \leftarrow EncKeyGen(1^k)$, and a pair of signing/verification keys $(SK, VK) \leftarrow SigKeyGen(1^k)$, and sets $CEK = (CK, EK, VK)$. $\mathcal{A}$ then hands *CEK* to $\mathcal{A}'$ as the public committing encryption key. $\mathcal{A}$ runs $\mathcal{A}'$ to find triple c, $(m, hint)$, $(m', hint')$ which is a collision for $\mathcal{CE}_{CtE}$. Then, $\mathcal{A}$ sets $d = Msg_{VK}(hint)$ and $d' = Msg_{VK}(hint')$, and outputs the triple c_c, d, d' which is a collision for $\mathcal{C}$. It is easy to see that (c_c, d) and (c_c, d') are valid commitments for m and m' and $m \neq m'$ since $m = Reveal_{CK}(c_c, d) \neq Reveal_{CK}(c_c, d') = m'$.

$\Leftarrow$ Assume adversary $\mathcal{A}$ can find collisions for $\mathcal{C}$ i.e., $\mathcal{A}$ can find c_c, d, d' such that (c_c, d) and (c_c, d') are valid commitments for m and m' but $m \neq m'$. We can easily construct adversary $\mathcal{A}'$ that can find collisions for $\mathcal{CE}_{CtE}$. $\mathcal{A}'$ views the commitment key *CK* and by itself picks a pair of encryption/decryption keys $(EK, DK) \leftarrow EncKeyGen(1^k)$, a pair of signing/verification keys $(SK, VK) \leftarrow SigKeyGen(1^k)$, and sets $CEK = (CK, EK, VK)$. $\mathcal{A}'$ then hands *CK* to $\mathcal{A}$ as the public commitment key. $\mathcal{A}'$ runs $\mathcal{A}$ to find triple c_c, d, d' which is a collision for $\mathcal{C}$. Then, $\mathcal{A}'$ runs internally $m \leftarrow Reveal_{CK}(c_c, d)$, $m' \leftarrow Reveal_{CK}(c_c, d')$, sets $hint = Sign_{SK}(d)$, $hint' = Sign_{SK}(d')$, then $\mathcal{A}'$ chooses $\hat{d} = d$ or d' and runs $c_e \leftarrow Enc_{EK}(\hat{d}, r_e)$, and outputs the triplet $c = (c_c, c_e)$, $(m, hint)$, $(m', hint')$ which is a collision for $\mathcal{CE}_{CtE}$. It is easy to see that $Ver_{CEK}(c, m, hint) = succeed$ and $Ver_{CEK}(c, m', hint') = succeed$ for m and m' and $m \neq m'$. ■

Lemma 2.3 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$$\mathcal{S} \text{ is UF-NMA-secure} \quad \Rightarrow \quad \mathcal{CE}_{CtE} \text{ is CE-RECOVER-secure.}$$

Proof Assume $\mathcal{CE}_{CtE}$ is not CE-RECOVER-secure i.e., adversary $\mathcal{A}'$ can find c, m, and *hint*, such that $Ver_{CEK}(c, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c) = \perp$.

Recall that $Ver_{CEK}(c, m, hint)$ has two options to succeed.

(a) Input parameters given by the sender: parse *hint* as $r_c \| r_e$, check if $c = (c_c, c_e) = CmtEnc_{CEK}(m, r_c \| r_e) = (Commit_{CK}(m, r_c), Enc_{EK}(Decommit_{CK}(m, r_c), r_e))$.

(b) Input parameters given by the recipient: parse *hint* as $Sign_{SK}(d)$, check if $m = Reveal_{CK}(Commit_{CK}(m, r_c), Msg_{VK}(Sign_{SK}(Decommit_{CK}(m, r_c))))$.

We now show that in either case (a) or (b), if $\mathcal{A}'$ succeeds to find c, m, and *hint*, such that $Ver_{CEK}(c, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c) = \perp$, then either $\mathcal{E}$ cannot be syntactically an encryption scheme or $\mathcal{C}$ cannot be syntactically a commitment scheme, or $\mathcal{S}$ cannot be UF-NMA-secure.

Proof of case (a) Assume $\mathcal{A}$' succeeds to find c, m, and *hint* s.t. the first option for *Ver* success holds and $DcmtDec_{CEK,DK}(c) = \perp$. We can show that either $\mathcal{E}$ cannot be syntactically an encryption scheme or $\mathcal{C}$ cannot be syntactically a commitment scheme.

Indeed, if $DcmtDec_{CEK,DK}(c) = \perp$ then it follows that: $DcmtDec_{CEK,DK}(c) = DcmtDec_{CEK,DK}(c_c, c_e)$

$= (Reveal_{CK}(c_c, Dec_{DK}(c_e)), Sign_{SK}(Dec_{DK}(c_e))$

$= (Reveal_{CK}(Commit_{CK}(m, r_c), Dec_{DK}(Enc_{EK}(Decommit_{CK}(m, r_c), r_e)), Sign_{SK}(Dec_{DK}(Enc_{EK}(Decommit_{CK}(m, r_c), r_e))))$

$\neq (m, hint)$, contradiction.

Proof of case (b) If $\mathcal{A}$' succeeds with the second option of *Ver*, we can easily construct adversary $\mathcal{A}$ that can forge signatures for $\mathcal{S}$ without any assistance from a signing oracle. $\mathcal{A}$ views the verification key *VK* and by itself picks a commitment key $CK \leftarrow KeySetup(1^k)$, and a pair of encryption/decryption keys $(EK, DK) \leftarrow EncKeyGen(1^k)$, and sets $CEK = (CK, EK, VK)$. $\mathcal{A}$ then hands *CEK* to $\mathcal{A}$' as the public committing encryption key. $\mathcal{A}$ runs $\mathcal{A}$' to find triple c, m, and *hint*, which for the second option of *Ver* succeeds. $\mathcal{A}$ outputs *hint*. ■

Proof of Theorem 3 We prove $\mathcal{CE}_{CtE}$ security against the strongest security notion of the encryption $\mathcal{E}$, i.e. E-IND-CCA2 and E-IND-gCCA2. Weaker notions e.g. E-IND-CPA and E-IND-CCA1 could easily be proved as well.

The theorem is immediately follows from Lemmas 3.1 – 3.2. ■

Lemma 3.1 Assume that $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2) -secure, $\mathcal{C}$ and $\mathcal{S}$ satisfy the syntactic properties of a commitment scheme and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{CE}_{CtE}$ is CE-IND-CCA2 (CE-IND-gCCA2) -secure $\Rightarrow$ $\mathcal{C}$ satisfies the hiding property.

Proof Assume adversary $\mathcal{A}$ can break hiding property of $\mathcal{C}$ then we can show that $\mathcal{CE}_{CtE}$ cannot pass the indistinguishability test even without using the decryption oracle at all (E-IND-CPA-security), let alone E-IND-gCCA2-secure.

Indeed, if $\mathcal{A}$ can find m_0, and m_1 s.t. it can distinguish the commitment of m_0, $c_c(m_0)$, from the commitment of m_1, $c_c(m_1)$, then obviously we can distinguish also $CmtEnc_{CEK}(m_0, r_c||r_e) \equiv (c_c(m_0), Enc_{EK}(d(m_0), r_e))$ from $CmtEnc_{CEK}(m_1, r_c||r_e) \equiv (c_c(m_1), Enc_{EK}(d(m_1), r_e))$, contradicting E-IND-CPA-security and certainly E-IND-gCCA2-security. ■

Lemma 3.2 Assume that $\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2) -secure, $\mathcal{C}$ and $\mathcal{S}$ satisfy the syntactic properties of a commitment scheme and a digital signature scheme respectively. Let $\mathcal{CE}_{CtE}$ be an asymmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{C}$ satisfies the hiding and binding properties $\Rightarrow$ $\mathcal{CE}_{CtE}$ is CE-IND-CCA2 (CE-IND-gCCA2) -secure.

Proof We will show CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{CtE}$ based on E-IND-CCA2-security (E-IND-gCCA2-security) of $\mathcal{E}$ and hiding and binding properties of $\mathcal{C}$.

For CE-IND-gCCA2-security, let $\mathcal{R}$ be the equivalence relation w.r.t. which $\mathcal{E}$ is secure. We define the equivalence relation for $\mathcal{CE}_{CtE}$ to be $\mathcal{R}'$ $(c_1, c_2) = \mathcal{R}'$ $((c_{c1}, c_{e1}), (c_{c2}, c_{e2}))$ = true iff $\mathcal{R}(c_{e1}, c_{e2})$ = true and $c_{c1} = c_{c2}$. It is easy to see that $\mathcal{R}'$ is decryption-respecting, since if $d_i = Dec_{DK}(c_{ei})$, then $\mathcal{R}'$ $((c_{c1}, c_{e1}), (c_{c2}, c_{e2}))$ = true implies that $(c_{c1}, d_1) = (c_{c2}, d_2)$, which implies that $m_1 = Reveal_{CK}(c_{c1}, d_1) = Reveal_{CK}(c_{c2}, d_2) = m_2$.

For the uniformity and simplicity of the proof we define also for CE-IND-CCA2-security an equivalence relation for $\mathcal{CE}_{CtE}$ to be $\mathcal{R}'$ $(c_1, c_2) = \mathcal{R}'$ $((c_{c1}, c_{e1}), (c_{c2}, c_{e2}))$ = true iff $c_1 = c_2$ i.e., $c_{e1} = c_{e2}$ and $c_{c1} = c_{c2}$.

We now show CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{CtE}$ w.r.t. $\mathcal{R}'$. For that, let Env_1 denote the usual environment where we place any adversary $\mathcal{A}$ for $\mathcal{CE}_{CtE}$. Namely,

(1) In *find*, Env_1 honestly answers the de-committing decryption queries of $\mathcal{A}$.

(2) After m_0 and m_1 are selected, Env_1 picks at random $b \leftarrow \{0, 1\}$, r_c, and r_e, then applies $c_{cb} \leftarrow Commit_{CK}(m_b, r_c)$, $d_b \leftarrow Decommit_{CK}(m_b, r_c)$ and $c_{eb} \leftarrow Enc_{EK}(d_b, r_e)$, and returns $c_b = (c_{cb}, c_{eb})$.

(3) In *guess*, Env_1 honestly answers de-committing decryption query $c = (c_c, c_e)$ provided that $\mathcal{R}'$ $((c_c, c_e), (c_{cb}, c_{eb})) = false$.

For CE-IND-gCCA2-security, we can assume that $\mathcal{A}$ never asks a query $c = (c_c, c_e)$ where $\mathcal{R}(c_e, c_{eb}) = true$ but $c_c \neq c_{cb}$. Indeed, by our assumption only the value $c_c = c_{cb}$ will check with d_b, so the answer to queries with $c_c \neq c_{cb}$ is $\perp$ (and $\mathcal{A}$ knows it). Hence, we can assume that $\mathcal{R}'$ $((c_c, c_e), (c_{cb}, c_{eb})) = false$ implies that $\mathcal{R}(c_e, c_{eb}) = false$.

We let $Succ_1(\mathcal{A})$ denote the probability $\mathcal{A}$ succeeds in Env_1 in predicting b.

Then we define the following "fake" environment Env_2. It is identical to Env_1 above, except for one aspect: in step (2) it would return bogus committing encryption $(c_c(0), c_{eb})$, i.e. puts the commitment to the zero string 0 instead of the expected c_{cb} by applying $c_c(0) \leftarrow Commit_{CK}(0, r_c)$. In particular, step (3) is the same as before with the understanding that $\mathcal{R}'$ $((c_c, c_e), (c_{cb}, c_{eb}))$ is evaluated with the fake challenge $(c_c(0), c_{eb})$. We let $Succ_2(\mathcal{A})$ denote the probability $\mathcal{A}$ succeeds in Env_2. We make two claims:

(a) Using hiding property of $\mathcal{C}$, no PPT adversary $\mathcal{A}$ can distinguish Env_1 from Env_2, that is

$\mathcal{C}$ satisfies the hiding property $\Rightarrow$ $| Succ_1(\mathcal{A}) - Succ_2(\mathcal{A}) | \leq negl(k)$

(b) Using E-IND-gCCA2-security of $\mathcal{E}$, no PPT adversary $\mathcal{A}$ can succeed in Env_2, that is

$\mathcal{E}$ is E-IND-gCCA2-secure $\Rightarrow$ $Succ_2(\mathcal{A}) < ½ + negl(k)$

Combined, claims (a) and (b) imply the theorem.

Proof of claim (a) If for some $\mathcal{A}$, $Succ_1(\mathcal{A}) - Succ_2(\mathcal{A}) > \varepsilon$ for non-negligible ε, we create $\mathcal{A}_1$ that will break the hiding property of $\mathcal{C}$. $\mathcal{A}_1$ picks, by itself, a pair of encryption/decryption keys $(EK, DK) \leftarrow EncKeyGen(1^k)$, a pair of signing/verification keys $(SK, VK) \leftarrow SigKeyGen(1^k)$, sets $CEK = (CK, EK, VK)$, and runs $\mathcal{A}$ (answering his de-committing decryption queries using DK, SK) until $\mathcal{A}$ outputs m_0 and m_1. At this stage $\mathcal{A}_1$ picks at random $b \leftarrow \{0, 1\}$, and outputs 0 and m_b, and claim to be able to distinguish $c_c(0) \leftarrow Commit_{CK}(0, r_c)$ from $c_{cb} = c_c(m_b) \leftarrow Commit_{CK}(m_b, r_c)$. $\mathcal{A}_1$ computes $c_{eb} \leftarrow Enc_{EK}(d(m_b), r_e) = Enc_{EK}(Decommit_{CK}(m_b, r_c), r_e)$. When presented with c_c' – a commitment to either 0 or m_b –

$\mathcal{A}_1$ will return to $\mathcal{A}$ the committing encryption (c_c', c_{eb}). $\mathcal{A}_1$ will then again run $\mathcal{A}$ to completion, refusing to de-committing decrypting (c_c, c_e) such that $\mathcal{R}$' ((c_c, c_e), (c_c', c_{eb})) = true. When $\mathcal{A}$ outputs b', $\mathcal{A}_1$ says that the message was m_b if $\mathcal{A}$ succeeds (b' = b), and says 0 otherwise. It is easy to check that in case c_c' = $c_c(m_b)$ = c_{cb}, $\mathcal{A}$ was run exactly in Env_1, otherwise – in Env_2, which easily implies that Pr [$\mathcal{A}_1$ succeeds] ≥ ½ + ε/2, a contradiction.

Proof of claim (b) If for some $\mathcal{A}$, $Succ_2(\mathcal{A}) > ½ + \varepsilon$, we create $\mathcal{A}_2$ that will break the IND-CCA2-security (IND-gCCA2-security) of $\mathcal{E}$. Specifically, $\mathcal{A}_2$ can simulate the de-committing decryption query (c_c, c_e) of $\mathcal{A}$ by asking its own decryption oracle to decrypt $d = Dec_{DK}(c_e)$, and returning $Reveal_{CK}(c_c, d)$. When $\mathcal{A}$' outputs m_0 and m_1, $\mathcal{A}_2$ runs $c_{ci} \leftarrow Commit_{CK}(m_i, r)$ and $d_i \leftarrow Decommit_{CK}(m_i, r)$ for $i = \{0, 1\}$, and claim to distinguish d_0 and d_1. When given challenge $c_{eb} \leftarrow Enc_{EK}(d_b, r)$, for unknown b, $\mathcal{A}_2$ gives $\mathcal{A}$' the challenge ($c_c(0)$, c_{eb}). Then, again, $\mathcal{A}_2$ uses its own decryption oracle to answer all queries (c_c, c_e) as long as $\mathcal{R}$' ((c_c, c_e), ($c_c(0)$, c_{eb})) = *false*. For CE-IND-gCCA2-security, from the definition of $\mathcal{R}$' and our assumption earlier, we see that $\mathcal{R}(c_e, c_{eb})$ = *false* as well, so all such queries are legal. Since $\mathcal{A}_2$ exactly recreates the environment Env_2 for $\mathcal{A}$, $\mathcal{A}_2$ succeeds with probability $Succ_2(\mathcal{A}) > ½ + \varepsilon$. ■

Proof of Theorem 4 We prove $\mathcal{CE}_{KSP}$ security against the strongest security notion of the encryption $\mathcal{E}$, i.e. E-IND-CCA2 and E-IND-gCCA2. Weaker notions e.g. E-IND-CPA and E-IND-CCA1 could easily be proved as well.

The theorem is immediately follows from Lemmas 4.1 – 4.4. ■

Recall that:

(i) From the syntax requirement of the symmetric encryption scheme $\mathcal{E}$, *Dec* deterministically recovers the plaintext m that was the input to *Enc* i.e., $Dec_K(Enc_K(m)) = m$.

(ii) From the syntax requirement of the commitment scheme $\mathcal{C}$, *Reveal* deterministically recovers the message m that was the input to *Commit* and *Decommit* i.e., $Reveal_{CK}(Commit_{CK}(m, r), Decommit_{CK}(m, r)) = m$.

(iii) From the syntax requirement of the digital signature scheme $\mathcal{S}$, *Msg* deterministically recovers the message m that was signed by *Sign* i.e., $Msg_{VK}(Sign_{SK}(m)) = m$.

Lemma 4.1 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of a symmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{KSP}$ be a symmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then $\mathcal{CE}_{KSP}$ satisfies the syntactic properties of a symmetric committing encryption scheme.

Proof From the definitions of *DcmtDec* and *CmtEnc*, and using (i), follows that $DcmtDec_{CK,K}(CmtEnc_{CK,K}(m, r)) = (Dec_{Ki}(Enc_{Ki}(m)), hint) = (m, hint)$.

From the definition of *Ver* in order to retrieve the secret key K_i that was used to encrypt message m we apply:

(∗) $Reveal_{CK}(Commit_{CK}(K_i, r_c), Msg_{VK}(Sign_{SK}(Decommit_{CK}(K_i, r_c)))) =$ [using (iii)] $Reveal_{CK}(Commit_{CK}(K_i, r_c), Decommit_{CK}(K_i, r_c)) =$ [using (ii)] K_i.

Let $c = CmtEnc_{CK,K}(m, r)$ then from the definition of *CmtEnc* also $c = Enc_{Ki}(m)$. Thus, from the definition of *Ver* follows that:

$Ver_{CK}(CmtEnc_{CK,K}(m, r), DcmtDec_{CK,K}(CmtEnc_{CK,K}(m, r))) = Ver_{CK}(Enc_{Ki}(m), m, hint) = Ver_{CK}(Enc_{Ki}(m), m, Sign_{SK}(Decommit_{CK}(K_i, r_c))) =_{[\text{using } (*)]} succeed.$ ■

Lemma 4.2 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of a symmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{KSP}$ be a symmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{E}$ is E-IND-CCA2 (E-IND-gCCA2) -secure $\Leftrightarrow$ $\mathcal{CE}_{KSP}$ is CE-IND-CCA2 (CE-IND-gCCA2) -secure.

Proof We prove CE-IND-CCA2-security and CE-IND-gCCA2-security of $\mathcal{CE}_{KSP}$ simultaneously. For CE-IND-gCCA2-security, let $\mathcal{R}$ be the equivalence relation w.r.t. which $\mathcal{E}$ is secure. We define the equivalence relation for $\mathcal{CE}_{KSP}$ to be $\mathcal{R}'(c_1, c_2) =$ true iff $\mathcal{R}(c_1, c_2) =$ true. For the uniformity and simplicity of the proof we define also for CE-IND-CCA2-security an equivalence relation for $\mathcal{CE}_{KSP}$ to be $\mathcal{R}'(c_1, c_2) =$ true iff $c_1 = c_2$.

$\Rightarrow$ Assume adversary $\mathcal{A}'$ can break the CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{KSP}$. We can easily construct adversary $\mathcal{A}$ that can break the E-IND-CCA2-security (E-IND-gCCA2-security) of $\mathcal{E}$. $\mathcal{A}$ runs $\mathcal{A}'$ internally, passing every de-committing decryption queries made by $\mathcal{A}'$ to its own decryption oracle. When $\mathcal{A}'$ outputs x_0, and x_1, $\mathcal{A}$ outputs them also. When $\mathcal{A}$ is presented with the challenge $c_b = Enc_{Ki}(x_b, r)$ (for unknown b), it hands it to $\mathcal{A}'$ and continue running $\mathcal{A}'$ waiting to its answer. Now, the definition of $\mathcal{R}'$ tells us that $\mathcal{A}'$ is disallowed to de-commit decrypt any c satisfying $\mathcal{R}'(c_b, c) =$ true. But such c are the only queries that $\mathcal{A}$ itself is disallowed to ask its decryption oracle! Thus, $\mathcal{A}$ can still handle all the legal de-committing decryption queries of $\mathcal{A}'$, in the same manner as before. Finally, $\mathcal{A}$ outputs the same guess b' that $\mathcal{A}'$ outputs, which clearly gives $\mathcal{A}$ the same probability of being correct as $\mathcal{A}'$ has.

$\Leftarrow$ Assume adversary $\mathcal{A}$ can break the E-IND-CCA2-security (E-IND-gCCA2-security) of $\mathcal{E}$. We can easily construct adversary $\mathcal{A}'$ that can break the CE-IND-CCA2-security (CE-IND-gCCA2-security) of $\mathcal{CE}_{KSP}$. $\mathcal{A}'$ runs $\mathcal{A}$ internally, passing every decryption queries made by $\mathcal{A}$ to its own de-committing decryption oracle. When $\mathcal{A}$ outputs x_0, and x_1, $\mathcal{A}'$ outputs them also. When $\mathcal{A}'$ is presented with the challenge $c_b = CmtEnc_{CK,K}(x_b, r)$ (for unknown b), it hands it to $\mathcal{A}$ and continue running $\mathcal{A}$ waiting to its answer. Now, the definition of $\mathcal{R}'$ tells us that $\mathcal{A}'$ is disallowed to de-commit decrypt any c satisfying $\mathcal{R}'(c_b, c) =$ true. But such c are the only queries that $\mathcal{A}$ itself is disallowed to ask its decryption oracle! Thus, $\mathcal{A}'$ can still handle all the legal de-committing decryption queries of $\mathcal{A}$, in the same manner as before. Finally, $\mathcal{A}'$ outputs the same guess b' that $\mathcal{A}$ outputs, which clearly gives $\mathcal{A}'$ the same probability of being correct as $\mathcal{A}$ has. ■

Lemma 4.3 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of a symmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{KSP}$ be a symmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{C}$ satisfies the binding property $\Rightarrow$ $\mathcal{CE}_{KSP}$ is CE-BIND-secure.

Proof Assume $\mathcal{CE}_{KSP}$ is not CE-BIND-secure. We can show that $\mathcal{E}$ cannot be syntactically an encryption scheme or we can find collisions for $\mathcal{C}$.

Let c, (m_1, $hint_1$), (m_2, $hint_2$) be a collision for $\mathcal{CE}_{KSP}$. That is $Ver_{CK}(c, m_1, hint_1) = Ver_{CK}(c, m_2, hint_2) = succeed$ and $m_1 \neq m_2$.

If the committing ciphertext c was produced for message m_1 and m_2 honestly, i.e. using the agreed key K_i, and *hint* is set honestly to the agreed $s_{iS}||s_{iR}$, then $\mathcal{E}$ cannot be syntactically an encryption scheme. From the definition of *CmtEnc* we have:

$c = CmtEnc_{CK,K}(m_1, r_1) = Enc_{Ki}(m_1, r_1)$ and also $c = CmtEnc_{CK,K}(m_2, r_2) = Enc_{Ki}(m_2, r_2)$

But for the deterministic *Dec* we require:

$Dec_{Ki}(c) = Dec_{Ki}(Enc_{Ki}(m_1, r_1)) = m_1 \neq m_2 = Dec_{Ki}(Enc_{Ki}(m_2, r_2)) = Dec_{Ki}(c)$, contradiction.

If the committing ciphertext c was produced for either message m_1 or m_2 dishonestly, i.e. using key $K \neq K_i$, and *hint* is set to appropriate $d_S(K)$ and $d_R(K)$ generated dishonestly by the sender (i.e. $hint = d_{KS}||d_{KR}$), then (c_{iS}, d_{iS}, d_{KS}) and (c_{iR}, d_{iR}, d_{KR}) are both collisions for $\mathcal{C}$, since: $Reveal_{CK_R}(c_{iS}, d_{iS}) = K_{iS} \neq K = Reveal_{CK_R}(c_{iS}, d_{KS})$ and $Reveal_{CK_S}(c_{iR}, d_{iR}) = K_{iR} \neq K = Reveal_{CK_S}(c_{iR}, d_{KR})$. ■

Lemma 4.4 Assume that $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ satisfy the syntactic properties of a symmetric encryption scheme, a commitment scheme, and a digital signature scheme respectively. Let $\mathcal{CE}_{KSP}$ be a symmetric committing encryption scheme constructed from $\mathcal{E}$, $\mathcal{C}$, and $\mathcal{S}$ as defined above. Then

$\mathcal{C}$ satisfies the binding property $\Rightarrow$ $\mathcal{CE}_{KSP}$ is CE-RECOVER-secure.

Proof Assume $\mathcal{CE}_{KSP}$ is not CE-RECOVER-secure. We can show that $\mathcal{E}$ cannot be syntactically an encryption scheme or we can find collisions for $\mathcal{C}$.

Let c, m, and *hint*, be s.t. $Ver_{CK}(c, m, hint) = succeed$ but $DcmtDec_{CK,K}(c) = \perp$.

If the committing ciphertext c was produced for message m_i honestly, i.e. using the agreed key K_i, and *hint* is set honestly to the agreed $s_{iS}||s_{iR}$ then $\mathcal{E}$ cannot be syntactically an encryption scheme, since:

$DcmtDec_{CK,K}(c) = (Dec_{Ki}(c), hint) = (Dec_{Ki}(Enc_{Ki}(m)), hint) \neq (m, hint)$, contradiction.

If the committing ciphertext c was produced for message m_i dishonestly, i.e. using key $K \neq K_i$, and *hint* is set to appropriate $d_S(K)$ and $d_R(K)$ generated dishonestly by the sender (i.e. $hint = d_{KS}||d_{KR}$), then (c_{iS}, d_{iS}, d_{KS}) and (c_{iR}, d_{iR}, d_{KR}) are both collisions for $\mathcal{C}$, since: $Reveal_{CK_R}(c_{iS}, d_{iS}) = K_{iS} \neq K = Reveal_{CK_R}(c_{iS}, d_{KS})$ and $Reveal_{CK_S}(c_{iR}, d_{iR}) = K_{iR} \neq K = Reveal_{CK_S}(c_{iR}, d_{KR})$, contradiction. ■

Proof of Theorem 5 The theorem is immediately follows from Lemmas 5.1 – 5.5. ■

Lemma 5.1 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}tS}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then $\mathcal{SC}_{\mathcal{CE}tS}$ satisfies the syntactic properties of a publicly verifiable signcryption scheme.

Proof Recall that:

(i) From the syntax requirement of the asymmetric committing encryption scheme $\mathcal{CE}$, *DcmtDec* deterministically recovers the plaintext m that was the input to *CmtEnc* and the *hint* i.e., $DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m)) = (m, hint)$.

(ii) From the syntax requirement of the digital signature scheme $\mathcal{S}$, *Msg* deterministically recovers the message m that was signed by *Sign* i.e., $Msg_{VK}(Sign_{SK}(m)) = m$.

Let $c_s = EncSign_{CEK,SK}(m, r)$. From the definition of *EncSign* we have $c_s = Sign_{SK}(CmtEnc_{CEK}(m, r)||CEK)$. In the following we omit the concatenation of *CEK* in the signature following the definitions of *VerDec* and *SCVer*. From the definition of *VerDec* and *EncSign* follows that: $VerDec_{CEK,DK,VK}(EncSign_{CEK,SK}(m, r))$

$= DcmtDec_{CEK,DK}(Msg_{VK}(Sign_{SK}(CmtEnc_{CEK}(m, r))))$

$=_{\text{[using (ii)]}} DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r)) =_{\text{[using (i)]}} (m, hint)$.

From the definition of *SCVer* follows that:

$SCVer_{VK}(EncSign_{CEK,SK}(m, r), VerDec_{CEK,DK,VK}(EncSign_{CEK,SK}(m, r)))$

$= Ver_{CEK}(Msg_{VK}(Sign_{SK}(CmtEnc_{CEK}(m, r))), DcmtDec_{CEK,DK}(Msg_{VK}(Sign_{SK}(CmtEnc_{CEK}(m, r)))))$

$=_{\text{[using (ii)]}} Ver_{CEK}(CmtEnc_{CEK}(m, r), DcmtDec_{CEK,DK}(CmtEnc_{CEK}(m, r))) = succeed.$ ■

Lemma 5.2 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then

$\mathcal{CE}$ is CE-IND-gCCA2-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ is CE-IND-gCCA2-secure.

Proof We will show CE-IND-gCCA2-security of $\mathcal{CE}_{SE}$ based on E-IND-gCCA2-security of $\mathcal{CE}$. Let $\mathcal{R}$ be the equivalence relation w.r.t. which $\mathcal{CE}$ is secure. We define the equivalence relation for $\mathcal{CE}_{SC}$ to be $\mathcal{R}'(c_{s1}, c_{s2}) =$ true iff $\mathcal{R}(c_{ce1}, c_{ce2}) =$ true where $c_{ce1}||CEK = Msg_{VK}(c_{s1})$, and $c_{ce2}||CEK = Msg_{VK}(c_{s2})$. We now show CE-IND-gCCA2-security of $\mathcal{CE}_{SC}$ w.r.t. $\mathcal{R}'$.

$\Rightarrow$ Assume adversary $\mathcal{A}'$ can break the CE-IND-gCCA2-security of $\mathcal{CE}_{SC}$. We can easily construct adversary $\mathcal{A}$ that can break the E-IND-gCCA2-security of $\mathcal{CE}$. $\mathcal{A}$ views the public committing encryption key *CEK* and by itself picks a pair of signing/verification keys $(SK, VK) \leftarrow SigKeyGen(1^k)$, and sets $CEK' = (CEK, SK, VK)$. $\mathcal{A}$ then hands CEK' to $\mathcal{A}'$ as the public committing encryption key. $\mathcal{A}$ uses its own de-committing decryption oracle to answer the de-signcryption queries it receives from $\mathcal{A}'$ as follows: $\mathcal{A}$ interprets the signcryption ciphertext c_s as a signature s and derives $c_{ce}||CEK \leftarrow Msg_{VK}(s)$, then $\mathcal{A}$ passes c_{ce} to its own de-committing decryption oracle. When $\mathcal{A}'$ outputs x_0, and x_1, $\mathcal{A}$ outputs them also. When $\mathcal{A}$ is presented with the challenge $c_{ce} = CmtEnc_{CEK}(x_b, r)$ (for unknown b), it sets $c = c_{ce}||CEK$, runs $s \xleftarrow{R} Sign_{SK}(c)$ and passes s as signcryption ciphertext c_s to $\mathcal{A}'$ and continue running $\mathcal{A}'$ waiting to its answer. Now, the definition of $\mathcal{R}'$ tells us that $\mathcal{A}'$ is disallowed to de-signcrypt any c_s' satisfying $\mathcal{R}'(c_s, c_s') =$ true. But such c_s' results in $c_{ce}'||CEK = Msg_{VK}(c_s')$ and c_{ce}' are the only queries that $\mathcal{A}$ itself is disallowed to ask its de-committing decryption oracle! Thus, $\mathcal{A}$ can still handle all the legal de-signcryption queries of $\mathcal{A}'$, in the same manner as before. Finally, $\mathcal{A}$ outputs the same guess b' that $\mathcal{A}'$ outputs, which clearly gives $\mathcal{A}$ the same probability of being correct as $\mathcal{A}'$ has.

$\Leftarrow$ Assume adversary $\mathcal{A}$ can break the CE-IND-gCCA2-security of $\mathcal{CE}$. We can easily construct adversary $\mathcal{A}'$ that can break the CE-IND-gCCA2-security of $\mathcal{CE}_{SC}$. $\mathcal{A}'$ views the

public committing encryption key $CEK' = (CEK, SK, VK)$ and hands CEK to $\mathcal{A}$ as the public committing encryption key. $\mathcal{A}'$ uses its own de-signcryption oracle to answer the de-committing decryption queries it receives from $\mathcal{A}$ as follows: $\mathcal{A}'$ first appends the public committing encryption key CEK to the committing ciphertext c_{ce} it receives from $\mathcal{A}$ and have $c = c_{ce}||CEK$, then $\mathcal{A}'$ runs $s \xleftarrow{R} Sign_{SK}(c)$, and passes the signature s as signcryption ciphertext c_s to its own de-signcryption oracle. When $\mathcal{A}$ outputs x_0, and x_1, $\mathcal{A}'$ outputs them also. When $\mathcal{A}'$ is presented with the challenge $c_s = EncSign_{CEK,SK}(x_b, r)$ (for unknown b), it interprets c_s as signature s and derives $c_{ce}||CEK \leftarrow Msg_{VK}(s)$. It passes the retrieved c_{ce} to $\mathcal{A}$ and continue running $\mathcal{A}$ waiting to its answer. Now, the definition of $\mathcal{R}'$ tells us that $\mathcal{A}'$ is disallowed to de-signcrypt any c_s' satisfying $\mathcal{R}'(c_s, c_s') = \text{true}$. But such c_s' results in $c_{ce}'||CEK = Msg_{VK}(c_s')$ and c_{ce}' are the only queries that $\mathcal{A}$ itself is disallowed to ask its de-committing decryption oracle! Thus, $\mathcal{A}'$ can still handle all the legal de-signcryption queries of $\mathcal{A}$, in the same manner as before. Finally, $\mathcal{A}'$ outputs the same guess b' that $\mathcal{A}$ outputs, which clearly gives $\mathcal{A}'$ the same probability of being correct as $\mathcal{A}$ has. ■

Lemma 5.3 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}tS}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then

$\mathcal{CE}$ is CE-BIND-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}tS}$ is CE-BIND-secure.

Proof

$\Rightarrow$ Assume adversary $\mathcal{A}'$ can find collisions for $\mathcal{CE}_{SC}$ i.e., $\mathcal{A}'$ can find c_s, (m, *hint*), (m', *hint*') such that $SCVer_{CEK'}(c_s, m, hint) = succeed$, $SCVer_{CEK'}(c_s, m', hint') = succeed$ and $m \neq m'$. We can easily construct adversary $\mathcal{A}$ that can find collisions for $\mathcal{CE}$. $\mathcal{A}$ views the public committing encryption key CEK and by itself picks a pair of signing/verification keys (SK, VK) $\leftarrow SigKeyGen(1^k)$, and sets $CEK' = (CEK, SK, VK)$. $\mathcal{A}$ then hands CEK' to $\mathcal{A}'$ as the public committing encryption key. $\mathcal{A}$ runs $\mathcal{A}'$ to find triplet c_s, (m, *hint*), (m', *hint*') which is a collision for $\mathcal{CE}_{SE}$. Then, $\mathcal{A}$ interprets the signcryption ciphertext c_s as signature s and derives $c_{ce}||CEK \leftarrow Msg_{VK}(s)$, and output the triplet c_{ce}, (m, *hint*), (m', *hint*'). It is easy to see that $Ver_{CEK}(c_{ce}, m, hint) = succeed$, $Ver_{CEK}(c_{ce}, m', hint') = succeed$ and $m \neq m'$.

$\Leftarrow$ Assume adversary $\mathcal{A}$ can find collisions for $\mathcal{CE}$ i.e., $\mathcal{A}$ can find c_{ce}, (m, *hint*), (m', *hint*') such that $Ver_{CEK}(c_{ce}, m, hint) = succeed$, $Ver_{CEK}(c_{ce}, m', hint') = succeed$ and $m \neq m'$. We can easily construct adversary $\mathcal{A}'$ that can find collisions for $\mathcal{CE}_{SC}$. $\mathcal{A}'$ views the public committing encryption key $CEK' = (CEK, SK, VK)$ and hands CEK to $\mathcal{A}$ as the public committing encryption key. $\mathcal{A}'$ runs $\mathcal{A}$ to find triplet c_{ce}, (m, *hint*), (m', *hint*') which is a collision for $\mathcal{CE}$. Then, $\mathcal{A}'$ appends the public committing encryption key CEK to the committing ciphertext c_{ce} and have $c = c_{ce}||CEK$, then $\mathcal{A}'$ runs $s \xleftarrow{R} Sign_{SK}(c)$, sets $c_s = s$, and output the triplet c_s, (m, *hint*), (m', *hint*'). It is easy to see that $SCVer_{CEK'}(c_s, m, hint) = succeed$, $SCVer_{CEK'}(c_s, m', hint') = succeed$ and $m \neq m'$. ■

Lemma 5.4 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}tS}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then

$\mathcal{CE}$ is CE-RECOVER-secure $\Leftrightarrow$ $\mathcal{SC}_{\mathcal{CE}tS}$ is CE-RECOVER-secure.

Proof

$\Rightarrow$ Assume $\mathcal{CE}_{SC}$ is not CE-RECOVER-secure and adversary $\mathcal{A}$' can find c_s, m, and *hint* such that $SCVer_{CEK'}(c_s, m, hint) = succeed$ but $VerDec_{CEK,DK,VK}(c_s) = \perp$. Then we can easily construct adversary $\mathcal{A}$ that can find c_{ce}, m, and *hint* such that $Ver_{CEK}(c_{ce}, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c_{ce}) = \perp$ and thus break CE-RECOVER security of $\mathcal{CE}$. $\mathcal{A}$ views the public committing encryption key *CEK* and by itself picks a pair of signing/verification keys (*SK*, *VK*) $\leftarrow SigKeyGen(1^k)$, and sets *CEK*' = (*CEK*, *SK*, *VK*). $\mathcal{A}$ then hands *CEK*' to $\mathcal{A}$' as the public committing encryption key. $\mathcal{A}$ runs $\mathcal{A}$' to find c_s, m, and *hint* such that $SCVer_{CEK'}(c_s, m, hint) = succeed$ but $VerDec_{CEK,DK,VK}(c_s) = \perp$. Then, $\mathcal{A}$ interprets the signcryption ciphertext c_s as signature s and derives $c_{ce}||CEK \leftarrow Msg_{VK}(s)$, and outputs c_{ce}, m, and *hint*. It is easy to see that $Ver_{CEK}(c_{ce}, m, hint) = succeed$ and $DcmtDec_{CEK,DK}(c_{ce}) = \perp$.

$\Leftarrow$ Assume $\mathcal{CE}$ is not CE-RECOVER-secure and adversary $\mathcal{A}$ can find c_{ce}, m, and *hint* such that $Ver_{CEK}(c_{ce}, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c_{ce}) = \perp$ then we can easily construct adversary $\mathcal{A}$' that can find c_s, m, and *hint* such that $SCVer_{CEK'}(c_s, m, hint) = succeed$ but $VerDec_{CEK,DK,VK}(c_s) = \perp$ and thus break CE-RECOVER security of $\mathcal{CE}_{SC}$. $\mathcal{A}$' views the public committing encryption key *CEK*' = (*CEK*, *SK*, *VK*) and hands *CEK* to $\mathcal{A}$ as the public committing encryption key. $\mathcal{A}$' runs $\mathcal{A}$ to find c_{ce}, m, and *hint* such that $Ver_{CEK}(c_{ce}, m, hint) = succeed$ but $DcmtDec_{CEK,DK}(c_{ce}) = \perp$. Then, $\mathcal{A}$' appends the public committing encryption key *CEK* to the committing ciphertext c_{ce} and have $c = c_{ce}||CEK$, $\mathcal{A}$' runs $s \xleftarrow{R} Sign_{SK}(c)$, sets $c_s = s$, and output the triplet c_s, m, and *hint*. It is easy to see that $SCVer_{CEK'}(c_s, m, hint) = succeed$ and $VerDec_{CEK,DK,VK}(c_s) = \perp$. ■

Lemma 5.5 Assume that $\mathcal{CE}$ and $\mathcal{S}$ satisfy the syntactic properties of an asymmetric committing encryption scheme and a digital signature scheme respectively. Let $\mathcal{SC}_{\mathcal{CE}t\mathcal{S}}$ be a publicly verifiable signcryption scheme constructed from $\mathcal{CE}$ and $\mathcal{S}$ as defined above. Then

$$\mathcal{S} \text{ is UF-CMA-secure} \quad \Rightarrow \quad \mathcal{SC}_{\mathcal{CE}t\mathcal{S}} \text{ is UF-CMA-secure.}$$

Proof Assume adversary $\mathcal{A}$' can forge signatures for $\mathcal{S}_{SC}$ then we can easily construct adversary $\mathcal{A}$ that can forge signatures for $\mathcal{S}$ without any assistance from a signing oracle. $\mathcal{A}$ views the verification key *VK* and by itself picks a pair of committing encryption keys (*CEK*, *DK*) $\leftarrow KeyGenSetup(1^k)$, and sets *VK*' = (*CEK*, *DK*, *VK*). $\mathcal{A}$ then hands *VK*' to $\mathcal{A}$' as the public verification key. $\mathcal{A}$ runs $\mathcal{A}$' to forge signature s such that $SigVer_{VK'}(s) = succeed$. Then, $\mathcal{A}$ outputs the same signature s that $\mathcal{A}$' outputs. ■