Designated Confirmer Signatures With Unified Verification

Guilin Wang [1], Fubiao Xia [2], and Yunlei Zhao [3]

[1] University of Wollongong, Australia (guilin@uow.edu.au)
[2] University of Birmingham, UK (F.Xia@cs.bham.ac.uk)
[3] Fudan University, China (yunleizhao@gmail.com)

March 2011

Abstract

After the introduction of designated confirmer signatures (DCS) by Chaum in 1994, considerable researches have been done to build generic schemes from standard digital signatures and construct efficient concrete solutions. In DCS schemes, a signature cannot be verified without the help of either the signer or a semi-trusted third party, called *the designated confirmer*. If necessary, the confirmer can further convert a DCS into an ordinary signature that is publicly verifiable. However, there is one limit in most existing schemes: the signer is *not* given the ability to disavow invalid DCS signatures. Motivated by this observation, in this paper we first propose a new variant of DCS model, called *designated confirmer signatures with unified verification*, in which both the signer and the designated confirmer can run the same protocols to confirm a valid DCS or disavow an invalid signature. Then, we present the first DCS scheme with unified verification and prove its security in the random oracle (RO) model and under a new computational assumption, called *Decisional Co-efficient Linear* (D-co-L) assumption, whose intractability in pairing settings is shown to be equivalent to the well-known *Decisional Bilinear Diffie-Hellman* (DBDH) assumption. The proposed scheme is constructed by encrypting Boneh, Lynn and Shacham's pairing based short signatures with *signed* ElGamal encryption. The resulting solution is efficient in both aspects of computation and communication. In addition, we point out that the proposed concept can be generalized by allowing the signer to run different protocols for confirming and disavowing signatures.

Keywords: Designated Confirmer Signature, Digital Signatures, Unified Verification.

1 Introduction

Background. Digital signatures introduced by Diffie and Hellman [10] are employed to achieve the integrity and authenticity of electronic documents. However, the signer may hope the recipient of a signature would not be able to show its validity to other parties. Hence, how to control the public verifiability of signatures is an important issue in both paper world and digital world. Chaum and van Antwerpen [7] introduced the concept of *undeniable signatures* to solve this problem. Unlike the ordinary signature's verification, in an undeniable signature scheme a verifier needs the help of the signer to verify an undeniable signature. Consequently, if the signer is unavailable or unwilling to help a verifier due to any possible reasons, this approach does not work in that situation either.

To overcome the above weakness in undeniable signatures, Chaum and van Antwerpen [8] introduced the concept of *designated confirmer signatures* (DCS). In a DCS scheme, both the signer and a semi-trusted third party called the *designated confirmer* can confirm the validity of a signature by running some interactive protocols with a verifier. However, such a verifier cannot further convince other parties of the same fact. Moreover, the confirmer can selectively convert a designated confirmer signature into an ordinary signature so that it is publicly verifiable. A number of DCS schemes [19, 16, 22, 24, 27, 30] have been presented, though most of them are either insecure or inefficient.

Previous Work. Many investigations on DCS have been targeted to construct efficient and secure generic schemes. Okamoto [24] provided the first formal definition of a DCS scheme and constructively proved that a DCS scheme is equivalent to public key encryption with respect to their existence.

But Michels and Stadler [22] pointed out that the confirmer can forge valid signatures on behalf of the signer in Okamoto's concrete schemes, and they also proposed a new security model and efficient schemes secure in their model. In 2000, however, Camenisch and Michels [5] identified an attack to link the validity of two signatures issued by different signers, when multiple signers in [22] share the same confirmer. Meanwhile, they proposed a new model which covers this attack and also suggested a generic DCS construction by encrypting a basic signature with the confirmer's public encryption key. This construction is actually straightforward but very inefficient, because no efficient zero knowledge proofs have been found for the assertion that the plaintext corresponding to a given ciphertext does contain a valid ordinary signature of some message. In 2004, Goldwasser and Waisbard [19] proposed a relaxed formal security model (referred to as GW model), and presented several DCS schemes via applying a generic transformation that relies on neither random oracles nor generic zero knowledge proofs. Their main idea is to weaken the security requirements of Okamoto by exploiting strong witness hiding proofs of knowledge (SWHPOK), rather than zero knowledge proofs. In Asiacrypt 2005, Gentry, Molnar and Ramzan [16] presented a generic DCS scheme by brilliantly introducing an indirect layer via using commitment schemes. That is, the signer generates a DCS by first issuing a basic signature on the commitment of a message and then encrypting the randomness used for the commitment separately. After that, Wang et al. [27] detected two security flaws in Gentry et al.'s scheme and repaired it by exploiting encryption with labels. In addition, a new general construction without explicit public key encryption was also given in [27].

In contrast, only a few concrete DCS schemes have been proposed. In 2008, Zhang et al. proposed an efficient DCS scheme based on bilinear pairings [30] but their scheme fails to meet *invisibility*, a crucial security requirement of DCS, as shown by Xia et al. [29]. Intuitively, invisibility requires that the validity or invalidity of an DCS must be invisible to any verifier, who can be an adaptive chosen-message attacker. Partially inspired by our preliminary work [28], Huang et al. presented another practical DCS scheme [20], which is proved to be secure in the standard model and also supports unified verification. In [20], it was not discussed how to adjust the verification protocol in the concurrent execution environment, though this is required for preventing possible attacks in the setting of DCS [16, 27]. In addition, the scheme in [20] suffers from the additional computational and communication costs incurred by the concurrent zero-knowledge transformation in the common reference string model (to be addressed later).

Motivation and Contribution. However, there is one limit in most existing DCS schemes: A signer is *not* given the ability to disavow invalid DCS signatures. Therefore, the current concept of DCS has not yet fully extended that of undeniable signatures, as the latter does grant the signer the ability of disavowal. Moreover, in many applications it seems more sensible to enable the signer having the same ability as the confirmer to confirm any valid DCS and deny any invalid DCS. In fact, this additional ability will not only alleviate the burden of the confirmer, but effectively prevents the signer from viciously claiming: "*This alleged DCS is not valid, but I am not able to show this*". Galbraith and Mao [13] first pointed out DCS schemes should allow a signer to be able to deny invalid signatures but they did not present any construction with this property. Motivated by this observation, in this paper we propose the concept of DCS with *unified verification*, together with a formal security model and a concrete construction. Simply speaking, in DCS scheme with unified verification both the signer and the designated confirmer can run the same protocol to confirm valid signatures, and another same protocol to disavow invalid signatures. Based on the security models in [5, 13], we first present a new security model for DCS with unified verification to capture all desirable security requirements (Section 3). We also point out that the proposed model can be easily generalized to accommodate *DCS with full verification*, in which the signer and the confirmer do not necessarily run the same protocols to confirm or disavow signatures.

Then, we consider how to construct a DCS with unified verification. This is a challenge, as simply revising the existing constructions does not work. The reason is that almost all previous DCS schemes [19, 16, 22, 24, 27] follow the approach of encrypting the signer's signature under the

confirmer's public key. So, without the confirmer's private key the signer is not able to show that a CCA2 ciphertext is not a proper encryption of his/her signature for a given message. In fact, it seems that even Wang et al.'s DCS [27] without using public encryption cannot be converted into a scheme supporting the signer's disavowal, since an alleged DCS may contain a non Diffie-Hellman tuple, for which the signer does not know a witness to prove this fact at all.

However, this does not mean that it is impossible to construct DCS schemes with unified verification or with full verification. Due to the amazing property of bilinear pairings, we constructed the first concrete DCS scheme which supports *unified verification* in the preliminary version of this work [28], though that scheme only achieves weak invisibility. By making a simple enhancement to our previous work [28], we get a new and secure DCS scheme with unified verification in this paper (Section 4) and prove its security in the random oracle model (Section 5). Specifically, the new scheme is constructed by encrypting the BLS pairing based short signature [3] under the *signed* ElGamal encryption [26]. Note that directly exploiting plain ElGamal encryption [12] cannot guarantee the *invisibility*, due to ElGamal's malleability (See more discussion in Section 4). Moreover, compared to the existing DCS schemes [19, 16, 27, 20], the proposed solution has a conceptual simpler structure and a short signature size, though the computational overhead is a little higher due to pairing evaluation. Another interesting observation about our construction is that the underlying signed ElGamal encryption is actually not CPA secure due to the fact that the DDH (Decisional Deffie-Hellman) problem is easy in pairing setting, though signed ElGamal encryption is proved to be CCA2-secure in the random oracle model and in the generic group model by Schnorr and Jakobsson [26]. This is seemingly contradictory to the result by Okamoto [24]. The likely reason for this is that our definitions on the security of DCS are different from Okamoto's, but we are not very sure at this moment. So, here we would like to promote this issue as an open problem.

2 Preliminaries

2.1 Bilinear Pairings and the BLS Signature

Basically, a pairing is a function that takes two points on an elliptic curve as input, and outputs an element of some multiplicative group. Weil pairing and Tate pairing are two known symmetric pairings, while some other pairings, e.g., Eta pairing and Ate Pairing [14], have been given more and more attentions.

Definition 1 *Suppose that G and G_t be two multiplicative cyclic groups of prime order q, while g is a generators of G. A bilinear pairing on (G, G_t) is a map $e : G \times G \to G_t$, which satisfies the following properties:*

Bilinearity: For all $u, v \in G$, and for all $a, b \in \mathbb{Z}_q$, $e(u^a, v^b) = e(u, v)^{ab}$.

Non-degeneracy: $e(g, g) \neq 1$, where 1 is the multiplicative identity of group G_t.

Computability: e can be efficiently computed.

We now review the BLS short signature scheme of Boneh, Lynn and Shacham [3]. In general, the scheme has three algorithms, *Key Generation*, *Sign*, and *Verify*. In addition, it needs a full-domain hash function H: $\{0,1\}^* \to G$.

Key Generation: A user randomly selects $x \in \mathbb{Z}_q^*$ as his private key, and computes $y = g^x$ as the corresponding public key.

Sign: To sign a message $m \in \{0,1\}^*$, the signer with private key x computes $h = H(m) \in G$, and the signature $\sigma = h^x \in G$.

Verify: Given a public key y and a message-signature pair (m, σ), a verifier first computes $h = H(m)$, and then checks if $e(g, \sigma) = e(y, h)$ holds or not. If it holds, he accepts the validity of (m, σ).

Note that the BLS signature is only one single element of G, so it is very short (for example, 171 bits) for some elliptic curves. In [3], it is proved that the security of the BLS signature scheme

follows the hardness of Computational Diffie-Hellman (CDH) problem in the random oracle model. In fact, the original BLS signature [3] is described in the setting of *asymmetric pairing* e, i.e., e is a bilinear map from $G_1 \times G_2$ to G_t, while G_1, G_2, and G_t are all multiplicative cyclic groups of prime order q. Here, for simplicity we just use *symmetric pairing* by letting $G = G_1 = G_2$. We note that to extend our DCS scheme for the asymmetric pairing is straightforward.

2.2 Complexity Assumptions

We introduce some complexity assumptions required in our proposal as below. Let $negl(n)$ denote any negligible function that grows slower than n^{-v} for any positive integer v and for all sufficiently large integer n. $x \in_R X$ denotes an random element x is picked from set X uniformly, and $x_1, x_2, ..., x_n \overset{R}{\leftarrow} X$ denotes $x_1, x_2, ..., x_n$ are random elements picked from set X uniformly. All the other alphabets and symbols follow the previous meanings.

Computational Diffie-Hellman Assumption (CDH). *Given $g, g^a, g^b \in G$, no probabilistic polynomial-time (PPT) algorithm can output $g^{ab} \in G$ with non-negligible probability, where $a, b \in_R \mathbb{Z}_q^*$.*

Decisional Bilinear Diffie-Hellman Assumption (DBDH) [1]. *With $g \in G$ described as above, Given g, g^a, g^b, g^c, where $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_q^*$, no PPT (Probablistic Polynomial Time) algorithm can distinguish between $e(g,g)^{abc}$ and a random element Z in G_t. Formally, for any PPT algorithm A, given g, g^a, g^b, g^c, where $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_q^*$, and $Z \in_R G_t$, the following defined advantage should be negligible:*

$$Adv_A^{DBDH}(n) = |\Pr[A(t, g^a, g^b, g^c, e(g,g)^{abc}) = 1] - Pr[A(t, g^a, g^b, g^c, Z) = 1]|$$

Now we propose a new assumption, called "Decisional-coefficient-Linear (D-co-L, in short) assumption", to serve our security analysis in Theorem 3. We shall show that the D-co-L assumption and the DBDH assumption are equivalent in the generic bilinear groups.

Decisional-coefficient-Linear Assumption (D-co-L): *With $g \in G$ and a pairing e described as above,given a tuple $(g, g^a, g^b, g^w, g^{by}, g^{wa+y}, g^z)$, where $a, b, w, y, z \overset{R}{\leftarrow} \mathbb{Z}_q^*$, no PPT algorithm A can distinguish between g^{wa+y} and a random element g^z in G. Formally, for any PPT algorithm A, given $(g, g^a, g^b, g^w, g^{by}, g^{wa+y}, g^z)$, where $a, b, y, w, z \overset{R}{\leftarrow} \mathbb{Z}_q^*$, we define the advantage of A:*

$$Adv_A^{D-co-L}(n) = |\Pr[A(t, g^a, g^b, g^w, g^{by}, g^{wa+y}) = 1] - \Pr[A(t, g^a, g^b, g^w, g^{by}, g^z) = 1]|$$

The probability is over the uniform random choice of the parameters to A, and over the coin tosses of A. We say the decisional-coefficient-linear assumption (T, ϵ)-holds, if there is no such A, which runs in time at most T and $Adv_A^{D-co-L}(n)$ is at least ϵ.

More clearly, the above assumption can be called D-co-L assumption in G. So, we can get the D-co-L assumption in G_t similarly. Due to the existence of bilinear mapping $e : G \times G \rightarrow G_t$, however, it is obvious to see that D-co-L assumption in G implies D-co-L assumption in G_t. In the proof of the following theorem, we actually use both of the assumptions, but for simplicity we just call them D-co-L assumption.

Theorem 1 *Decisional-coefficient-Linear Problem is equivalent to Decisional Bilinear Diffie-Hellman Problem in the generic bilinear groups.*

Proof: First we prove D-co-L assumption implies DBDH assumption, i.e., D-co-L assumption $\Rightarrow$ DBDH assumption. A D-co-L attacker A is given a challenge query $(g, g^a, g^b, g^w, g^{by}, g^{wa+y}, g^z)$, where $y, z, a, b, w \overset{R}{\leftarrow} \mathbb{Z}_q^*$, g is a generator of a generic bilinear group G. A can only perform group operations in G and G_t, and the bilinear map $e : G \times G \rightarrow G_t$, by interacting with an oracle

$\mathcal{O}$. All group elements are encoded as random strings, and only the equality can be tested by the adversary. Suppose that an DBDH attacking algorithm B, with access related oracles in $\mathcal{O}$ which provides **group actions** (G and G_t) and **pairings**, can solve the DBDH problem in polynomial time T. We now use B as a subroutine to construct A solving D-co-L problem in polynomial time. A first computes $e(g^{wa+y}, g^b)/e(g, g^{by}) = e(g^{wa}, g^b) = e(g, g)^{abw}$, sets $Z = e(g^z, g^b)/e(g, g^{by})$, and then outputs a DBDH challenge as $(g, g^a, g^b, g^w, e(g, g)^{abw}, Z)$ with other public parameters for B. Note that, in B's view, this is a uniformly selected DBDH challenge. Finally, A outputs 1, i.e, $g^{wa+y} = g^z$, if B outputs 1, i.e., $e(g, g)^{abw} = Z$; Otherwise, A answers 0 for $g^{wa+y} \neq g^z$. It is easy to check that, as long as B solves DBDH problem in polynomial time T, with making at most l queries, A can solve D-co-L problem in at most polynomial time T plus the time for evaluating two pairings and makes at most l queries.

Next we prove DBDH assumption implies D-co-L assumption, i.e., DBDH assumption $\Rightarrow$ D-co-L assumption. Suppose a DBDH attacker B is given a challenge query $(g, g^a, g^b, g^w, e(g, g)^{abw}, Z)$ with all parameters having the same meanings as in the above description. Now we will construct such an algorithm B who uses a subroutine A that solves a D-co-L instance.

B chooses y randomly from $\mathbb{Z}_p^*$, and computes g^y and g^{by}. We denote $Z = e(g, g)^z$ for some (unknown) z, and $e(g, g^b) = g_t \in G_t$. Then, B computes

$e(g, g^b)^a = e(g^a, g^b)$, $\quad e(g, g^b)^b = e(g^b, g^b)$,
$e(g, g^b)^w = e(g^w, g^b)$, $\quad e(g, g^b)^{by} = e(g^{by}, g^b)$,
$e(g, g^b)^{wa+y} = e(g^{wa+y}, g^b) = e(g^{wa}, g^b) \cdot e(g^y, g^b) = e(g, g)^{abw} \cdot e(g, g^b)^y$,
$e(g, g^b)^{z'} = e(g, g)^z \cdot e(g, g^b)^y = Z \cdot e(g, g^b)^y$, i.e., $z' = z/b + y$.

The output D-co-L tuple is $(g_t, g_t^a, g_t^b, g_t^w, g_t^{by}, g_t^{wa+y}, g_t^{z'})$. In A's view, this is a uniformly selected D-co-L challenge.

As long as A outputs 1, i.e., $g_t^{wa+y} = g_t^{z'}$, which implies $e(g, g)^{abw} = e(g, g)^z$, B outputs 1. Otherwise, B outputs 0. Note, $z' = z/b + y$ means b^{-1} is the inverse of b in $\mathbb{Z}_q^*$, and thus z' always exists in $\mathbb{Z}_q^*$. So, B can solve DBDH problem in the same polynomial time as a successful D-co-L attacker with the same numbers of oracle queries. Hence, the hardness of these two problems are equivalent. □

2.3 Concurrent Zero Knowledge From Honest-Verifier Zero-Knowledge

Zero-knowledge (ZK) protocols allow a prover to validate theorems to a verifier without giving away any other knowledge other than the theorems being true (i.e., existing witnesses) [18]. Traditional notion of ZK considers the security in a stand-alone (or sequential) execution of the protocol. Motivated by the use of such protocols in an asynchronous network like the Internet where many protocols are run concurrently at the same time, studying security properties of ZK protocols in such concurrent settings has attracted extensive research efforts in recent years [11]. Informally, a ZK protocol is called concurrent zero-knowledge (CZK) if the ZK related simulatability property holds in the concurrent settings, namely, when a malicious verifier concurrently interacts with a polynomial number of honest prover instances and schedules message exchanges as it wishes. We note, in DCS schemes, we require CZK protocols, because an adversary in DCS schemes may act as arbitrary cheating verifiers during the concurrent execution of protocols that confirm or deny all alleged DCS signatures.[1]

In this work, for presentation simplicity, we describe the Confirm and Disavow protocols with Σ-protocols (i.e., 3-round public-coin special honest verifier interactive zero-knowledge (HVIZK) with special soundness) directly.

Definition 2 *A 3-round public-coin protocol $\langle P, V \rangle$ is said to be a Σ-protocol for a relation R if the following hold:*

- *Completeness. If P, V follow the protocol, the verifier always accepts.*

[1] We note that the CZK issue was not realized in [20], where only stand-alone 4-round ZK is mentioned.

Common input. An element $x \in L$ of length n, where L is an $\mathcal{NP}$-language that admits Σ-protocols.

P's private input. A witness w for $x \in L$.

Random oracle. An *unprogrammable* random oracle denoted $\mathcal{O}$.

Round-1. The verifier V takes $e \in \{0,1\}^k$ uniformly at random, and sends $c = \mathcal{O}(e)$ to P.

Round-2. The prover P sends a (i.e., the first-round of the underlying Σ-protocol by running the underlying P_L) to V.

Round-3. V sends e to P.

Round-4. After receiving e from V, P first checks whether $c = \mathcal{O}(e)$. If not, P simply aborts; otherwise (i.e, $c = \mathcal{O}(e)$), P sends z (i.e., the last-round of the underlying Σ-protocol by running the underlying P_L) to V.

V's decision. V checks, by running the underlying V_L, whether (a, c, z) is an accepting conversation of the underlying Σ-protocol for showing $x \in L$.

Figure-1. Straight-line CZK protocol with unprogrammable RO

- *Special soundness. From any common input x of length n and any pair of accepting conversations on input x, (a, e, z) and (a, e', z') where $e \neq e'$, one can efficiently computes w such that $(x, w) \in R$. Here a, e, z stand for the first, the second and the third message respectively and e is assumed to be a string of length t (that is polynomially related to n) selected uniformly at random in $\{0,1\}^t$.*
- *Special honest verifier zero-knowledge (SHVZK). There exists a probabilistic polynomial-time (PPT) simulator S, which on input x and a random challenge string e, outputs an accepting conversation of the form (a, e, z), with the same probability distribution as the real conversation between the honest P, V on input x.*

Σ-protocols have been proved to be a very powerful cryptographic tool and are widely used. Transformation methodologies from Σ-protocols to CZK protocols, in the common reference string (CRS) model, are known (e.g., [9, 15]), but usually incurs much additional computational and communication complexity. Moreover, for CZK transformation in the CRS model, *the CRS should be included as a part in the public-key of the confirmer*, which additionally increases the public-key length of the confirmer. The transformation methodology proposed in [9] is recalled in Appendix A, which is among the most efficient transformations.

As we aim for DCS in the RO model, in this work we develop a highly efficient transformation from Σ-protocols to *straight-line* CZK in the *unprogrammable* RO model, where straight-line CZK means that the CZK simulator works in a straight-line way (*without rewinding the underlying adversary*). Given access to a random oracle $\mathcal{O}$, we can transform a Σ-protocol into a non-interactive zero-knowledge (NIZK) protocol via the Fiat-Shamir heuristics. But, the NIZK got this way loses deniability [23, 25], which is however required for DCS schemes. The deniability loss is due to the programmability of RO in the security analysis [23, 25]. To overcome the deniability loss of simulation with programmable RO, the works of [23, 25] proposed the unprogrammable RO model, and showed that ZK with unprogrammable RO reserves the deniability property. We remark that, in this work, unprogrammable RO is used only for achieving highly practical CZK, other parts of security analysis still rely on regular (programmable) random oracle.

Roughly speaking, before running the Σ-protocol (a, e, z), we require the verifier to first commit to its random challenge e by sending $c = H(e)$ on the top, where H is a hash function that is modeled as an unprogrammable RO in the analysis. The protocol is depicted in Figure-1. *Note that*

the additional computational complexity and communication complexity, incurred by this approach of transformation with unprogrammable RO, is minimal: only a hash value is incurred. Due to space limitation, the reader is referred to Appendix A for proof details.

3 Security Model

In this section, we present a new security model for designated confirmer signatures with unified verification. Essentially, we update Gentry et al's model [16] for DCS schemes with unified verification, also having consideration to the models given in [19, 5, 13]. After the model is described, we shall explain the difference between this model and the original models. In addition, we shall briefly mention how this new model can be modified to accommodate DCS with full verification.

Definition 3 (Syntax). *A* correct designated confirmer signature scheme with unified verification *involves three roles of parties, i.e., a signer S, a designated confirmer C, and a verifier V, and consists of the following components:*

Key Generation (G_s, G_c): *Given the security parameter n, denoted by* 1^n*, as input, probabilistic polynomial time (PPT) algorithm* G_s *outputs a pair of strings* (sk_S, pk_S) *as the signer's private key and public key, respectively. Similarly, PPT algorithm* G_c *that takes on input* 1^n*, outputs a pair of strings* (sk_C, pk_C) *as the designated confirmer's private key and public key, respectively.*

Sign: *Given a message m and a signer's private key* sk_S*, algorithm Sign produces a (standard) signature* σ *for message m. Namely,* $\sigma = Sign(m, sk_S)$.

Verify: *Given a public key* pk_S*, a message m, and a signature* σ*, algorithm Verify outputs Accept or Reject. For any key pair* (sk_S, pk_S)*, any message m,* $Verify(m, Sign(m, sk_S), pk_S)$ =*Accept.*

DCSSign: *Given a message m, a signer's private key* sk_S *and the confirmer's public key* pk_C*, algorithm DCSSign outputs* σ' *as a designated confirmer signature on message m. Namely,* $\sigma' = DCSSign(m, sk_S, pk_C)$.

Extract: *Given* $(m, \sigma', sk_C, pk_C, pk_S)$ *as input, algorithm Extract outputs a string* σ *such that Verify(m,* σ*,* pk_S*) = Accept or* $\perp$*. In the case Extract can successfully extract a valid standard signature* σ *from* σ'*, we say that* σ' *is* extractable *w.r.t. message m. Otherwise,* σ' *is* unextractable.

Confirm: *As an interactive protocol, either the signer S with private input* sk_S *or the designated confirmer C with private input* sk_C *can run Confirm protocol with a verifier V to confirm that an alleged DCS* σ' *for a message m is extractable. The common input for the protocol is (m,* σ'*,* pk_S*,* pk_C*). After the protocol is run, the verifier outputs* $b \in \{Accept, \perp\}$*. We say* σ' *is* valid *w.r.t. message m, if the verifier's output is Accept. Otherwise, the validity of* σ' *is undetermined. The Confirm protocol should be complete and sound.*

a) `Completeness`: *For all honest C, S, and V, if Verify(m, Extract(m,* σ'*,* sk_C*,* pk_C*,* pk_S*),* pk_S*)* =*Accept, then* $Confirm_{(C,V)}(m, \sigma', pk_S, pk_C)$ =*Accept, and* $Confirm_{(S,V)}(m, \sigma', pk_S, pk_C)$=*Accept.*

b) `Soundness:` *For any potentially cheating confirmer* C'*, any potentially cheating signer* S'*, and any honest verifier V, if Verify(m, Extract(*$m, \sigma', sk_C, pk_C, pk_S$*),* pk_S*)* $=\perp$*, then*

$$\Pr[Confirm_{(C',V)}(m, \sigma', pk_S, pk_C) = Accept] < negl(n), \text{ and}$$
$$\Pr[Confirm_{(S',V)}(m, \sigma', pk_S, pk_C) = Accept] < negl(n).$$

The probability is taken over all possible coins tossed by C'*,* S'*, V,* G_s*,* G_c*, and Extract. This means, neither a cheating confirmer* C' *nor a cheating signer* S' *can convince an honest verifier V that an un-extractable designated confirmer signature* σ' *is valid. In other words, all valid DCS signatures are extractable.*

Disavow: *As an interactive protocol, either the signer S with private input* sk_S *or the designated confirmer C with private input* sk_C *can run Disavow protocol with a verifier V to convince that an alleged DCS* σ' *is unextractable. The common input to the protocol is (m,* σ'*,* pk_S*,* pk_C*), while the verifier output is* $b \in \{Accept, \perp\}$*. If the verifier's output is Accept, we say* σ' *is* invalid *w.r.t.*

message m. Otherwise, the invalidity of σ' is undetermined. The Disavow protocol should be complete and sound.

a) `Completeness:` *For all honest C, S, and V, if Verify(m, Extract(m, σ', sk_C, pk_C, pk_S), pk_S) $= \perp$, then $Disavow_{(C,V)}(m, \sigma', pk_S, pk_C)$ =Accept, and $Disavow_{(S,V)}(m, \sigma', pk_S, pk_C)$ =Accept.*

b) `Soundness:` *For any potentially cheating confirmer C', any potentially cheating signer S', and any honest verifier V, if Verify(m, Extract(m, σ', sk_C, pk_C, pk_S), pk_S) =Accept, then*

$$\Pr\left[Disavow_{(C',V)}(m, \sigma', pk_S, pk_C) = Accept\right] < negl(n), \text{ and}$$
$$\Pr\left[Disavow_{(S',V)}(m, \sigma', pk_S, pk_C) = Accept\right] < negl(n).$$

The probability is taken over all possible coins tossed by C', S', V, G_s, G_c, and Extract. This means, neither a cheating confirmer C' nor a cheating signer S' can convince an honest verifier V that an extractable designated confirmer signature σ' is invalid. In other words, all invalid DCS must be unextractable. □

Remark 1. In contrast to the models given in [5, 19, 16, 27], there are three main differences in the above syntax definition. Firstly, we include the basic signature generation and verification algorithms to make the syntax more complete. Secondly, an algorithm *DCSSign* is now used to produce a DCS instead of an interactive protocol *ConfirmSign* in [19, 16, 27] to allow the signer generating a valid DCS and confirming it when it is just generated. The reason for this is that the signer will use the same *Confirm* protocol to show the validity of a DCS as does by the confirmer. Finally, in our model the signer is also able to use the *Disavow* protocol to show the invalidity of an alleged DCS. This is definitely necessary, as our DCS model targets to support unified verification.

Due to the above changes in syntax, we accordingly update the security definitions by including all necessary oracle accesses. Security for the signer or unforgeability requires that no adaptive PPT adversary can forge a valid DCS on a fresh message on behalf of a specific signer, even he compromises the secret keys of the confirmer and other signers. This means that forgeability for DCS should be fulfilled in multi-signer settings, i.e., in the scenario of multiple signers sharing the same confirmer. In our definition of unforgeability given below, the forging algorithm is not given oracle accesses for which the confirmer is the prover, since it already holds the confirmer's private key sk_C. Due to a similar reason, the *Sign* oracle for underlying signatures is not provided as the attacker can simulate this oracle by asking *DCSSign* queries and then running *Extract* to get basic signatures for any messages.

Definition 4. Security for the signer (Unforgeability): *Let $\mathcal{F}$ be a PPT forging algorithm, which on input 1^n, pk_S, pk_C and sk_C, can request oracle access in $\mathcal{O}_{\mathcal{F}}$=$\{DCSSign, Confirm_{(S,\mathcal{F})}, Disavow_{(S,\mathcal{F})}\}$ for polynomially many times for adaptively chosen inputs of its choice; and then outputs a DCS message-signature pair (m, σ') in which message m is not previously asked in DCSSign queries. We say a DCS scheme is* secure for the signer or existentially unforgeable*, if any such $\mathcal{F}$,*

$$\Pr[Verify(m, Extract(m, \sigma', sk_C, pk_C, pk_S), pk_S) = Accept] < negl(n).$$

The probability is taken over all possible coins used by $\mathcal{F}$, S, and key generation algorithm G_s, G_c.

Intuitively, security for the confirmer or invisibility means that no adaptive PPT adversary $\mathcal{D}$ can distinguish between a valid DCS and an invalid DCS for a given message (or two designated confirmer signatures).

Definition 5. Security for the confirmer (Invisibility): *Firstly, Key Generation algorithms are run for the signer and the confirmer on input 1^n. $\mathcal{D}$ is given pk_S and pk_C, which are the public keys of the signer and the confirmer. As a training purpose, $\mathcal{D}$ is allowed to create signature-key pairs $(sk_{\mathcal{D}}, pk_{\mathcal{D}})$ (not necessarily via Key Generations) and to interact with the confirmer with respect to these keys. Furthermore, $\mathcal{D}$ can make arbitrary oracle queries in $\mathcal{O}_{\mathcal{D}} = \{Sign, DCSSign,$*

$Confirm_{(S,\mathcal{D})}$, $Confirm_{(C,\mathcal{D})}$, $Disavow_{(S,\mathcal{D})}$, $Disavow_{(C,\mathcal{D})}$, $Extract\}$. *Then, the distinguisher has to present one message* m. *After a fair coin is flipped, the adversary is given a corresponding DCS* $\sigma' = DCSSign(m, sk_S, pk_C)$, *where* $b = 0$, *or a fake DCS signature chosen uniformly at random from the signature space where* $b = 1$. *Now* $\mathcal{D}$ *is again allowed to access the above oracles except that he can not enquire for* σ' *via any of these oracles. Finally, the distinguisher must output one bit information* b' *to guess the value of* b. *We say a DCS scheme with unified verification is* secure for the confirmer or invisible, *if for any PPT distinguisher* $\mathcal{D}$:

$$\Pr[b' = b] \leq 1/2 + negl(n).$$

The above probability is taken over the coin tosses of the signer, the confirmer, key generation algorithms and the oracles.

Remark 2. Note we adopt the definition of invisibility by Galbraith and Mao [13], which in turn is slightly different from the model by Camenisch and Michels [5]. What we defined here is actually to require that the adversary cannot decide a given DCS' validity with respect to the message he chose, without the help of the signer or the confirmer. However, the security requirement in [5], requires that the adversary should be unable to relate a valid DCS to a signed message and a message with a fake signature. Galbraith and Mao have proved that these two type of invisibility are actually equivalent in the standard model of computation. In addition, we disallow the adversary to have sk_S. Otherwise, it will be trivial for him to distinguish signatures via unified verification protocols.

Compared to GMR model [16], where another property "transcript simulatability" was proposed, it is hard to say which security requirement for the confirmer is better. We believe that transcript simulatability and invisibility are two different approaches for studying DCS schemes, as it is not known if one of these two properties implies the other or not [29]. In addition, note that the scheme in [20] is also analyzed for invisibility, not transcript simulatability. The main obstacle to prove transcript-simulatability in our scheme is that the signed ElGamal, our underlying encryption scheme, is even not CPA-secure. However, the generic DCS proposed in [16, 27] both rely on CCA2-secure public key encryption.

Definition 6 (Security). *We say a correct designated confirmer signature scheme is* secure, *if it satisfies security for the signer and for the confirmer. Namely, it is existentially unforgeable and invisible.*

In fact, [16, 27] also studies *security for verifier or unfoolability*, which requires that any DCS confirmed by running *Confirm* protocol must be extractable, and that every alleged DCS confirmed by running *Disavow* protocol must be unextractable. As this property follows the soundness of *Confirm* and *Disavow* protocols [16, 27], we do not separately specify it here.

Finally, from the above description we can see that by introducing additional *Confirm* and *Disavow* protocols for the signer, the formal model for DCS with unified verification can be directly generalized to accommodate DCS with full verification, in which the signer can also confirm and disavow a signature, but not necessarily runs the same protocols as the confirmer.

4 The Proposed Scheme

Based on BLS signature scheme [3], which has been reviewed in Section 2.1, we now present a designated confirmer signature scheme with unified verification. Basically, a DCS in our scheme is just the signed ElGamal encryption [26] of a BLS signature. After the scheme description, we shall give more explanations on the construction.

We use a symmetric bilinear map $e : G \times G \to G_t$, where G is a multiplicative cyclic group of prime order q and g is a generator of G. $H : \{0,1\}^* \to G$ is a full-domain hash function, and $H' : \{0,1\}^* \to \mathbb{Z}_q^*$ is a cryptographic hash function.

Key Generation: The signer picks $x_s \in_R \mathbb{Z}_q^*$ as his private key, and computes $y_s = g^{x_s}$ as his public key. Similarly, the confirmer sets its private/public key pair as $(x_c, y_c = g^{x_c})$, where $x_c \in_R \mathbb{Z}_q^*$.

Sign: Given a signer's private key x_s and a message m, output the signature $\sigma = h^{x_s} \in G$, where $h = H(m) \in G$.

Verify: Given (m, σ), check whether $e(g, \sigma) = e(y_s, h)$ holds, where $h = H(m) \in G$.

DCSSign: After generating a basic signature $\sigma = H(m)^{x_s}$ for message m by using the signer's private key x_s, output DCS for message m as $\sigma' = (\sigma_1, \sigma_2, s, t)$ by computing:

$$\sigma_1 = y_c^r,\ \sigma_2 = \sigma g^r,\ \text{where } r \in_R \mathbb{Z}_q^*;\ \text{and}$$
$$s = H'(y_c^k, \sigma_1, \sigma_2), t = k + sr \bmod q,\ \text{where } k \in_R \mathbb{Z}_q^*.$$

It is easy to see that σ' is exactly the signed ElGamal encryption [26] under the private/public key pair $(x_c^{-1}, g = y_c^{x_c^{-1}})$, which is equivalent to the confirmer's key pair $(x_c, y_c = g^{x_c})$. Namely, (σ_1, σ_2) is the naive ElGamal ciphertext of basic signature $\sigma = H(m)^{x_s}$ under the key pair $(x_c^{-1}, g = y_c^{x_c^{-1}})$, while (s, t) is a Schnorr signature on message (σ_1, σ_2) under the temporary private/public key pair $(r, \sigma_1 = y_c^r)$.

Extract: Given a message m and an *alleged* DCS $\sigma' = (\sigma_1, \sigma_2, s, t)$, which satisfies $s \equiv H'(y_c^t \sigma_1^{-s}, \sigma_1, \sigma_2)$, the confirmer extracts the basic signature $\sigma = \sigma_2 / \sigma_1^{x_c^{-1}}$ if $e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_s)^{x_c}$, where $h = H(m)$. Otherwise, $\perp$ is output.

Confirm: Given common input (m, σ', y_s, y_c), where $\sigma' = (\sigma_1, \sigma_2, s, t)$ is an alleged DCS, the confirmer C with the private key x_c can check the validity of the DCS by verifying whether $e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_s)^{x_c}$ holds or not, where $h = H(m)$. As $e(h, y_s)^{x_c} \equiv e(h, y_c)^{x_s}$, the signer S with the private key x_s can similarly know the validity of σ' by checking $e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_c)^{x_s}$. If σ' is valid, either the confirmer C or the signer S can run the following interactive zero knowledge protocol with a verifier V to convince this fact:

$$PK\{(x_c \vee x_s) : [e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_s)^{x_c} \wedge y_c = g^{x_c}] \vee [e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_c)^{x_s} \wedge y_s = g^{x_s}]\}.$$

Disavow: On input (m, σ', y_s, y_c), where $\sigma' = (\sigma_1, \sigma_2, s, t)$ is an alleged DCS, if $e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_s)^{x_c}$ or $e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_c)^{x_s}$, where $h = H(m)$, this means that σ' is an invalid DCS for message m. Then, either the confirmer C or the signer S can run the following interactive zero knowledge protocol with a verifier V to convince this fact:

$$PK\{(x_c \vee x_s) : [e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_s)^{x_c} \wedge y_c = g^{x_c}] \vee [e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_c)^{x_s} \wedge y_s = g^{x_s}]\}.$$

Note that the above PKs are for "the (in)equality of two discrete logarithms" $\vee$ "the (in)equality of another two discrete logarithms", and each part can be proved easily by using the standard techniques [6, 4, 21]. The implementation details of these zero knowledge proofs are given in Appendix B.

Remark 3. First, note that the idea of building a DCS here is inspired by the Boneh et al's verifiably encrypted signature (VES) scheme [2], which encrypts a basic BLS signature using ElGamal encryption with the adjudicator's key $(x_c, y_c = g^{x_c})$. The adjudicator in VES plays a similar role as the confirmer in DCS. Here, we exploit the same idea but change the format of the ciphtertext via effectively setting the confirmer's key pair $(x_c^{-1}, g = y_c^{x_c^{-1}})$. The result is very interesting, as we get a DCS scheme in which the validity of a hidden signature (i.e. DCS) is not publicly visible any more, compared to Boneh et al.'s hidden but publicly verifiable VES.

However, the above resulting scheme is actually not a secure DCS, as it fails to meet invisibility, due to the malleability of naive ElGamal encryption. That is, given a target DCS $(\sigma_1 = y_c^r, \sigma_2 = \sigma g^r)$ for a message m, an adaptive attacker can simply derive the validity of (σ_1, σ_2) by inquiring the validity of $(\sigma_1' = \sigma_1 y_c^{r'}, \sigma_2' = \sigma_2 g^{r'})$ w.r.t. the same message m by selecting a random number r'. Note that such an attack is allowed in the security definition of invisibility. To address this issue, signed ElGamal encryption [26] is exploited to add one Schnorr signature (s, t) showing that the creator of ciphertext (σ_1, σ_2) indeed knows the secret value of r which is used for encryption.

Equivalently, this implies that in the proposed scheme the issuer of a DCS $(\sigma_1, \sigma_2, s, t)$ knows the corresponding basic signature σ, as $\sigma = \sigma_2/g^r$. Hence, the above attack does not work any more, since such a DCS has a fixed format and is not malleable.

Remark 4. Note that the proposed DCS scheme is not strongly unforgeable (but is existentially unforgeable) for an attacker who has comprised the confirmer's private key x_c as explained below. Since a valid DCS for a message m has the form of $\sigma' = (\sigma_1, \sigma_2, s, t) = (y_c^r, \sigma g^r, s, t)$ satisfying $s \equiv H'(y_c^t \sigma_1^{-s}, \sigma_1, \sigma_2)$, the attacker with x_c can first extract the basic signature by computing $\sigma = \sigma_2/{\sigma_1}^{x_c^{-1}}$. Then, the attacker can trivially forge another valid DCS $\bar{\sigma}' = (\bar{\sigma}_1, \bar{\sigma}_2, \bar{s}, \bar{t})$ for the same message m. Nevertheless, this does not violate our definition of unforgeability specified in Definition 3, as a successful forger is required to produce a valid DCS on a *new* message, not a previously signed message.

Remark 5. According to Remark 2, both the signer and the confirmer can check a designated confirmer signature's validity or invalidity of an alleged DCS by using their own private keys. Then, either of them can run the same *Confirm* or *Disavow* interactive zero knowledge protocol with a verifier to show whether σ' is valid or not. Hence, to check the validity of a signature the verifier can interact with either the signer or the designated confirmer. Due to this reason, we call our scheme a DCS with *unified verification*. In particular, in our scheme the signer is granted the ability to disavow any invalid designated confirmer signature. This new feature is interesting, as our scheme serves a better extension of undeniable signatures [8], in which there is a disavow protocol for the signer; nevertheless no current DCS schemes except [20], which inspired by our prototype in [28] of this scheme, offer Disavow protocol for the signer.

Generalized Version. As discussed in Section 1, our unified verification DCS can be simply generalized to an full verification version. The idea is to get rid of the "OR" relation in the interactive zero-knowledge (IZK) protocols. For $Confirm_{(S,V)}$, the signer initially checks the validity of a given DCS, then runs a zero-knowledge protocol $PK\{x_s : e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_c)^{x_s} \wedge y_s = g^{x_s}\}$. For $Confirm_{(C,V)}$, the IZK protocol will be $PK\{x_c : e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_s)^{x_c} \wedge y_c = g^{x_c}\}$. To disavow an invalid signature, either the signer or the verifier can first the invalidity of a given DCS using their own secret key, and then runs $PK\{x_s : e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_c)^{x_s} \wedge y_s = g^{x_s}\}$ or $PK\{x_c : e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_s)^{x_c} \wedge y_c = g^{x_c}\}$, respectively. Hence, the above extension accommodates the generalized DCS model, where the signer and the confirmer can confirm (or disavow) signatures via different protocols.

Comparison. We give a brief comparison between our DCS scheme and the DCS schemes in [27, 20]. In comparison, our scheme has a smaller signature size over the DCS scheme of [27], which is also DCS scheme secure in the random oracle model. Comparing to Huang et al's scheme [20], our Confirm and Disavow protocols are much more efficient in both aspects of computation and communications, and the confirmer in our DCS scheme has shorter public-key, since their HVIZK requires more computational and communication costs when transformed into CZK in the common reference string (CRS) model (where the CRS needs to be included as a part in the public-key of confirmer). Moreover, our scheme is also conceptual simpler. A detailed comparison will be provided in full version of this work.

5 Security Proofs

Under the standard CDH assumption, the BLS signature scheme [3] is provably secure in the random oracle model. The unforgeability of our DCS relies upon the security of BLS scheme, *without direct use of random oracles*. The new D-co-L assumption, proposed in this work, gives rise to the invisibility of our DCS scheme in the random oracle model.

Theorem 2 *If the BLS signature scheme is* $(t', q'_H, q'_S, \varepsilon')$*-secure against existential forgery, then the proposed DCS scheme is* $(t, q_H, q_S, \varepsilon)$*-secure against existential forgery w.r.t. Definition 4, where*

$q'_H = q_H$, $q'_S = q_S$, $\varepsilon' = \varepsilon$, $t' \leq t + 6c \cdot (q_S + q_C + q_D)$, *where* c *is a constant, denoting the time to compute one pairing evaluation, one exponentiation in* G*, and one exponentiation in* G_t.

The proof of Theorem is given in Appendix C.

Theorem 3 *Under the D-co-L assumption, the proposed DCS scheme is invisible in the RO model.*

Proof: Suppose a challenger algorithm $\mathcal{C}$ is given the D-co-LA challenge, i.e., to distinguish two tuples, $(g, g^a, g^b, g^w, g^{by}, g^{wa+y})$ and $(g, g^a, g^b, g^w, g^{by}, g^z)$ where g is a generator in a multiplicative cyclic group G with prime order q. A pairing is constructed as $e : G \times G \rightarrow G_t$ where G_t is another multiplicative cyclic group with the same order q. Consider the invisibility game modeled in Definition 5, C needs to simulate a DCS environment for some PPT distinguisher $\mathcal{D}$, in which $\mathcal{D}$ tries to distinguish two pairs: (m, sig) and (m, sig_R), where $sig = DCSSign(m, sk_S, pk_C)$ and sig_R is chosen uniformly at random from the signature space. $\mathcal{D}$ can access the hash oracle, $Sign$, $DCSSign$, $Confirm$, $Disavow$ oracles before and after the challenge request phase. So, $\mathcal{C}$ can setup a DCS scheme instance and simulate the game for $\mathcal{D}$ as below.

First, $\mathcal{C}$ sets $pk_C = g^b$, and $pk_S = g^w$. Then, $\mathcal{C}$ simulates all the oracles for $\mathcal{D}$ as follows:

$Hash$ **query by** $\mathcal{D}$: Upon receiving $\mathcal{D}$'s queried message m, $\mathcal{C}$ picks u randomly and sets $H(m) = g^u$. Then, add (m, u) to a $\mathcal{H}$ list, which is initially empty.

$Sign$ **query by** $\mathcal{D}$: For a queried message m, $\mathcal{C}$ first checks if $(m, u) \in \mathcal{H}$ for some u. If yes, outputs basic signature $\sigma = g^{wu}$. Otherwise, selects u randomly, adds (m, u) to the H list, and outputs $\sigma = g^{wu}$.

$DCSSign$ **query by** $\mathcal{D}$: For a queried message m, similarly $\mathcal{C}$ can get a unique tuple $(m, u) \in \mathcal{H}$ for some u. Then, $\mathcal{C}$ computes basic signature $\sigma = g^{wu}$. By picking a random r, $\mathcal{C}$ computes $\sigma_1 = pk_C^r = g^{br}$ and $\sigma_2 = \sigma \cdot g^r = g^{wu+r}$. Since r as 'the signing key', $\mathcal{C}$ can simply produce a Schnorr signature (s, t) for message (σ_1, σ_2). Finally, $\mathcal{C}$ outputs $(\sigma_1, \sigma_2, s, t)$ to $\mathcal{D}$.

$Extract$ **query by** $\mathcal{D}$: For given an alleged DCS $(\sigma_1, \sigma_2, s, t)$ for message m, $\mathcal{C}$ outputs $\perp$ if it cannot find m in the $\mathcal{H}$ list. Otherwise, retrieves (m, u) from the $\mathcal{H}$ list and computes $\overline{\sigma} = g^{wu}$. Then, if $e(\sigma_2, g^b) \neq e(\sigma_1, g) \cdot e(\overline{\sigma}, g^b)$, $\mathcal{C}$ outputs $\perp$. Otherwise, $\mathcal{C}$ knows that the queried DCS signature-message pair is valid, so it outputs the basic signature $\overline{\sigma} = g^{wu}$.

$Confirm/Disavow$ **query by** $\mathcal{D}$: Similar with the $Extract$ oracle, $\mathcal{C}$ can check the queried DCS's validity easily. To convince $\mathcal{D}$ the validity of queried DCS, $\mathcal{C}$ just needs to straightforwardly run the underlying CZK simulator (refer to Appendixes A and B).

For the challenge message m' submitted by $\mathcal{D}$, $\mathcal{C}$ computes the DCS signature sig, in case the coin toss is head, as follows. Let $H(m') = g^a$, which *implicitly* sets the underlying BLS signature for m' as $H(m')^{sk_S} = g^{wa}$. Then, $\mathcal{C}$ sets $\sigma_1 = g^{by}$, where y is treated as the randomness used in the original DCSSign phase, and $\sigma_2 = g^{wa+y}$. For the Schnorr signature part, i.e., constructing (s, t), we assume $\mathcal{C}$ also controls $H'(\cdot)$ oracle. Thus it can simply selects s, t randomly, and let $g^{bk} = g^{bt} g^{-brs}$ mod q. Note that here the public key for the internal Schnorr signature is g^{br}, the secret key is r and the base of logarithm is g^b. Finally, $\mathcal{C}$ outputs the challenge DCS for message m as $sig = (\sigma_1, \sigma_2, s, t)$, which is a valid DCS on message m.

In case the coin toss is tail, $\mathcal{C}$ outputs a fake DCS $sig_R = (\sigma_1, \sigma_2, s, t)$ for message m', where $\sigma_1 = g^{by}$, $\sigma_2 = g^z$, and (s, t) is a simulated Schnorr signature showing that $(\sigma_1 = g^{by}, \sigma_2 = g^z)$ is a well formed ElGamal encryption.

After that, $\mathcal{C}$ can continuously answer $\mathcal{D}$'s oracle queries as simulated above. Finally, if $\mathcal{D}$ can distinguish the two signatures sig and sig_R by outputting a correct guess bit b', w.r.t. m' with a non-negligible advantage, it is straightforward to see that $\mathcal{C}$ can output the same bit b' to solve the given challenge also with a non-negligible advantage. □

6 Conclusion and Future Work

In this paper, based on BLS short signature [3] we presented a new efficient designated confirmer signature (DCS) scheme that additionally enables the signer to disavow any invalid signatures. We

call such a scheme as a DCS with full verification. As DCS has been considered for the extension of undeniable signatures, we believe this new feature is attracting in potential applications of DCS, like fair exchange of digital commitments between two users over the Internet. Moreover, our scheme achieves the unified verification, as both the signer and confirmer just use the same Confirm or Disavow protocol to convince a verifier that an alleged DCS is valid or invalid, respectively. Based on security models given in [5, 13], we have proposed a new security model to accommodate a DCS with unified verification, and showed the security of the proposed scheme in the random oracle model under a newly introduced computational assumption, which is independent of interest. In addition, we have proposed a very efficient way that transforms Σ-protocols into concurrent zero knowledge protocols. As the future work, it would be very interesting to explore the relations between invisibility and transcript-simulatability [16], and build new efficient DCS schemes with unified or full verification.

References

[1] Boneh, D. and Boyen, X. Efficient selective-id secure identity-based encryption without random oracles. *EUROCRYPT 2004, Springer-Verlag, LNCS 3027*, pages 223–238, 2004.

[2] Boneh, D. and Gentry, C. and Lynn, B. and Shacham, H. aggregate and verifiably encrypted signatures from bilinear maps. *Proc. Advances in Cryptology - Eurocrypt 2003, LNCS 2656*, page 641, 2003.

[3] Boneh, D. and Lynn, B. and Shacham, H. Short signatures from the weil pairing. *Proceedings of Asiacrypt 2001, LNCS 2248*, pages 514–532, 2001.

[4] Boudot, F. and Traore, J. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. *ICICS 1999, LNCS 1726*, pages 87–102, 1999.

[5] Camenisch, J. and Michels, M. Confirmer signature schemes secure against adaptive adversaries. *Proc. of Advances in Cryptology - EUROCRYPT '00, LNCS 1870*, pages 243–258, 2000.

[6] Chaum, D. and Pederson, T. P. Wallet databases with observers. *In proceedings of Crypto 1992, LNCS 740*, pages 89–105, 1992.

[7] Chaum, D. and van Antwerpen, H. Undeniable signatures. *In Proceedings of Crypto 1989, LNCS 435*, pages 212–216, 1989.

[8] Chaum, D. and van Antwerpen, H. Designated confirmer signatures. *In Proceedings of Eurocrypt 1994, LNCS 950*, pages 86–91, 1994.

[9] Damgard, I. Efficient concurrent zero-knowledge in the auxiliary string model. *Proc. of Advances in Cryptology - EUROCRYPT '00*, pages 418–430, 2000.

[10] Diffie, W. and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory, IT-22(6)*, pages 644–654, 1976.

[11] Dwork, C. and Naor, M. and Sahai, A. Concurrent zero-knowledge. *ACM STOC*, pages 409–418, 1998.

[12] ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Tran. on Information Theory*, 31(4):469–472, 1985.

[13] Galbraith, S.D. and Mao, W. Invisibility and anonymity of undeniable and confirmer signatures. *Proceedings of the 2003 RSA conference on The cryptographers' track*, pages 80–97, 2003.

[14] Galbraith, S.D. and Paterson, K.G. and Smart, N.P. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[15] Gennaro, R. Multi-trapdoor commitments and their applications to proofs of knowledgee secure under concurrent man-in-the-middle attacks. *Advances in Cryptology - CRYPTO '04, LNCS 3152*, pages 220–236, 2004.

[16] Gentry, C. and Molnar, D. and Ramzan, Z. Efficient designated confirmer signatures without random oracles or general zero-knowledge proofs. *Advances in Cryptology - ASIACRYPT 2005, LCNS 3788*, pages 662–681, 2005.

[17] Goldreich, O. and Micali, S. and Wigderson, A. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of ACM*, 38(1):691–792, 1991.

[18] Goldwasser, S. and Micali, S. and Rackoff, C. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing, volume 18*, (1):186–208, 1984.

[19] Goldwasser, S. and Waisbard, E. Transformation of digital signature schemes into designated confirmer signature schemes. *In Proceedings of TCC 2004, LCNS 2951*, pages 77–100, 2004.

[20] Huang, Q. and Wong, D. S. and Susilo, W. A new construction of designated confirmer signature and its application to optimistic fair exchange. *Pairing 2010, Springer-Verlag, LNCS 6487*, pages 41–64, 2010.

[21] Kurosawa, K. and Heng, S. H. 3-move undeniable signature scheme. *Proc. of Advances in Cryptology - EUROCRYPT '05, LNCS 3494*, pages 181–197, 2005.

[22] Michels, M. and Stadler, M. Generic constructions for secure and efficient confirmer signature schemes. *In Proceedings of Eurocrypt 1998, LNCS 473*, pages 458–464, 1998.

[23] Nielsen, J. Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. *Proc. of Crypto 2002*, pages 111–126, 2002.

[24] Okamoto, T. Designated confirmer signatures and public-key encryption are equivalent. *Proc. of Crypto 1994, LNCS 2894*, pages 61–74, 1994.

[25] Pass, R. On deniability in the common reference string and random oracle models. *Proc. of Crypto 2003*, pages 316–337, 2003.

[26] C.P Schnorr and M. Jakobsson. Security of signed elgamal encryption. *Proc. of ASIACRYPT '00, LNCS 1976*, pages 73–89, 2000.

[27] Wang, G. and Baek, J. and Wong, D. S. and Bao, F. On the generic and efficient constructions of secure designated confirmer signatures. *In Proceedings of PKC 2007, LNCS 4450*, pages 43–60, 2007.

[28] Wang, G. and Xia, F. A pairing based designated confirmer signature scheme with unified verification. *Technical Report 29, School of Computer Science, Univerisity of Birmingham*, Dec. 2009. ISSN: 0962-3671.

[29] Xia, F. and Wang, G. and Xue, R. On the invisibility of designated confirmer signatures. *Proc. of Aisaccs 2011, to appear.*

[30] Zhang, F. and Chen, X. and Wei, B. Efficient designated confirmer signature from bilinear pairings. *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 363–368, 2008.

A CZK Transformation From Σ-Protocols

Zero-knowledge (ZK) protocols allow a prover to validate theorems to a verifier without giving away any other knowledge other than the theorems being true (i.e., existing witnesses). This notion was introduced by Goldwasser, Micali and Rackoff [18] and its generality was demonstrated by Goldreich, Micali and Wigderson [17]. Since its introduction ZK has found numerous and extremely useful applications, and by now has been playing the central role in modern cryptography.

Traditional notion of ZK considers the security in a stand-alone (or sequential) execution of the protocol. Motivated by the use of such protocols in an asynchronous network like the Internet where many protocols are run concurrently at the same time, studying security properties of ZK protocols in such concurrent settings has attracted extensive research efforts in recent years, initiated by Dwork, Naor and Sahai [11]. Informally, a ZK protocol is called concurrent zero-knowledge (CZK) if the ZK related simulatability property holds in the concurrent settings, namely, when a malicious verifier concurrently interacts with a polynomial number of honest prover instances and schedules message exchanges as it wishes.

We note, in DCS schemes, we require CZK protocols, because an adversary in DCS schemes may act as arbitrary cheating verifiers during the concurrent execution of protocols that confirm or deny all alleged DCS signatures. In this work, for presentation simplicity, we describe the Confirm and Disavow protocols with Σ-protocols (i.e., 3-round public-coin special honest verifier zero-knowledge with special soundness) directly. We then discuss transformation methodologies from Σ-protocols to CZK protocols in the common reference string (CRS) model or in the *unprogrammable* random oracle model.

A.1 Transformation from Σ-Protocol into CZK in the CRS Model

There are several general methodologies that transform Σ-protocols into CZK arguments in the CRS model (e.g., [9, 15]). To our knowledge, the approach proposed in [9] is the most efficient and has conceptual simple structure, which is suggested to use in this work.

For the transformation proposed in [9], the CRS consists of the public-key for a trapdoor commitment scheme. In this work, we use the DL-based trapdoor commitment, where the public-key is $h = g^x$, and the commitment to a value $v \in Z_q$ is $c = g^r h^v$, where r is randomly taken from Z_q and is served as the decommitment information. Note that for this concrete implementation, to commit to a value in Z_q, the committer needs to perform about 1.5 exponentiations, and the receiver needs to perform also about 1.5 exponentiations. The communication complexity, besides the transmission of the committed value v (that is sent in the decommitment stage), is about $2|q|$ (suppose the commitment c is of about $|q|$ bits).

To transform a Σ-protocol (a, e, z) into CZK, the key idea of [9] is to send $C(a)$, rather then the plain a, at the first-round of the transformed protocol, where C denotes the trapdoor commitment scheme; in the third-round, the prover opens the value a and computes the third-round message z.

Moreover, for the general transformation in the CRS model, *the CRS should be included as a part in the public-key of the confirmer*, which additionally increases the public-key length of the confirmer.

A.2 CZK from Σ-protocols with *unprogrammable* RO

Given access to a random oracle (RO) $\mathcal{O}$, we can transform a Σ-protocol into a non-interactive zero-knowledge (NIZK) protocol via the Fiat-Shamir heuristic. But, the NIZK got this way loses deniability [23, 25], which is however required for DCS schemes. The deniability loss is due to the programmability of RO in the security analysis [23, 25]. To overcome the deniability loss of simulation with programmable RO, the works of [23, 25] proposed the unprogrammable RO model where all parties have access to an unprogrammable (fixed) RO, where ZK with unprogrammable RO reserves the deniability property.

In this section, we give a general yet simple method of transforming Σ-protocols into *straight-line* CZK with *unprogrammable RO*, where straight-line CZK means that the CZK simulator works in a straight-line way (*without rewinding the underlying adversary*).

Given a Σ-protocol $\langle P_L, V_L\rangle(x)$ which consists of three rounds (a, e, z) for an $\mathcal{NP}$-language L, the transformed protocol, denoted $\langle P, V\rangle$ is presented in Figure-1 (page 6).

Roughly speaking, before running the Σ-protocol (a, e, z), we require the verifier to first commit to its random challenge e by sending $c = h(e)$ on the top, where h is a hash function that is modeled as RO in the analysis.

Note that the additional computational complexity and communication complexity, incurred by this approach of transformation with unprogrammable RO, is minimal: only a hash value is incurred.

Theorem 4 *The protocol depicted in Figure-1 is a* straight-line *CZK* proof *with* unprogrammable *RO for any language admitting Σ-protocol.*

Proof (outline). The completeness of the protocol $\langle P, V\rangle$ can be directly checked.

Perfect soundness. The perfect soundness of $\langle P, V\rangle$ is from the observations: the commitment c perfectly hides e in the RO model; Then, the perfect soundness of $\langle P, V\rangle$ is inherited from the special soundness of the underlying Σ-protocol $\langle P_L, V_L\rangle$. That is, the transformed protocol $\langle P, V\rangle$ is a *proof* rather than an *argument* (i.e., computationally sound protocol).

***Straight-line* CZK with *unprogrammable* RO.** For any concurrent malicious verifier V^*, the simulator S runs V^* as a subroutine and works as follows, with oracle access to a *unprogrammable* RO $\mathcal{O}$:

- For any oracle query made by V^* on input e, S makes the same query to the unprogrammable RO $\mathcal{O}$. S returns back the answer, denoted c, from $\mathcal{O}$ to V^*, and records (c, r) into a list $\mathcal{L}_{\mathcal{O}}$.
- Whenever V^* starts a new concurrent session, on a common input $x \in L$, by sending c (as the first-round message) to S, S works as follows:
  - S firstly checks whether $c \in \mathcal{L}_{\mathcal{O}}$. If not, S simply aborts the simulation, and outputs "failure". This failure is called "*Case-1 failure*" for presentation simplicity.
  - If $c \in \mathcal{L}_{\mathcal{O}}$, S retrieves the record (c, e) in $\mathcal{L}_{\mathcal{O}}$ and works as follows: S runs the underlying SHVZK simulator S_L (guaranteed for the underlying Σ-protocol $\langle P_L, V_L\rangle$) on the input (x, e), denoted $S_L(x, e)$, to get a simulate transcript (a, e, z) of the underlying Σ-protocol $\langle P_L, V_L\rangle$. Then S sends a to V^* as the second-round message of the current session. If V^* returns back e to S in the third-round, S returns back z in the fourth-round and successfully completes the simulation of the current session; If V^* returns back $e' \neq e$ in the third-round, S simply aborts the simulation, and outputs "failure". This failure is called "*Case-2 failure*" for presentation simplicity.

  It is easy to check that S outputs "failure" (either Case-1 failure or Case-2 failure) with negligible probability in the RO model. Specifically, for Case-1 failure, with overwhelming probability V^* cannot guess the correct value c without querying the RO $\mathcal{O}$ with e; For Case-2 failure, with overwhelming probability V^* cannot get two different values e, e' such that $c = \mathcal{O}(e) = \mathcal{O}(e')$.

  Conditioned on S does not output "failure", the simulation of S is identical to the real view of V^*, which establishes the CZK property. Furthermore, S works in the unprogrammable RO model, as S never programs the RO $\mathcal{O}$ by itself. Specifically, S only accesses the unprogrammable RO $\mathcal{O}$ to see the queries made by the underlying V^*. Moreover, the simulation of S with restricted RO is *straight-line*, as S never rewinds the underlying V^*. □

B HVIZKs on Confirm and Disavow Protocols

In this section, we show how to run honest verifier interactive zero-knowledge proof (HVIZK), actually Σ-protocols, to complete the *Confirm* and *Disavow* protocols. To this end, we directly adapt the protocols given in [21], and depict the implementation details as below.

In the *Confirm* protocol, the knowledge statement is $PK\{(x_c \vee x_s)) : [e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_s)^{x_c} \wedge y_c = g^{x_c}] \vee [e(\sigma_2, y_c)/e(\sigma_1, g) = e(h, y_c)^{x_s} \wedge y_s = g^{x_s}]\}$. So, once the prover (either the signer or the confirmer) and a verifier pre-compute $A_1 = A_2 = e(\sigma_2, y_c)/e(\sigma_1, g)$, $B_1 = e(h, y_s)$, $B_2 = e(h, y_c)$, $C_1 = y_c$, $C_2 = y_s$, and $D_1 = D_2 = g$, they can run the *Confirm* protocol as shown in Figures 2 and 3 to prove "$(A_1 = B_1^{x_c} \wedge C_1 = D_1^{x_c}) \vee A_2 = B_2^{x_s} \wedge C_2 = D_2^{x_s}$)".

In the *Disavow* protocol, the knowledge statement is $PK\{(x_c \vee x_s)) : [e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_s)^{x_c} \wedge y_c = g^{x_c}] \vee [e(\sigma_2, y_c)/e(\sigma_1, g) \neq e(h, y_c)^{x_s} \wedge y_s = g^{x_s}]\}$. So, once the prover (either the signer or the confirmer) and a verifier pre-compute $A_1 = A_2 = e(\sigma_2, y_c)/e(\sigma_1, g)$, $B_1 = e(h, y_s)$, $B_2 = e(h, y_c)$, $C_1 = y_c$, $C_2 = y_s$, and $D_1 = D_2 = g$, they can run the *Disavow* protocol as shown in Figures 4 and 5 to prove "$(A_1 \neq B_1^{x_c} \wedge C_1 = D_1^{x_c}) \vee A_2 \neq B_2^{x_s} \wedge C_2 = D_2^{x_s}$)".

As mentioned in Section 2.3, both the above *Confirm* and *Disavow* HVIZK protocols should be converted into CZK so that they can be executed with multiple verifiers concurrently.

C Proof of Theorem 2

Proof. Given a forgery algorithm $\mathcal{F}$ for the proposed DCS scheme, we shall construct a forgery algorithm $\mathcal{F}'$ for the underlying BLS signature scheme. For presentation simplicity, we assume $\mathcal{F}$ *behaves well* in the random oracle model, i.e., $\mathcal{F}$ always requests the hash of a message m before requesting a designated confirmer signature.

The BLS forger $\mathcal{F}'$ is given the signer's public key y_s for which the private key is unknown to $\mathcal{F}'$ and has access to the *Sign* and hash oracles. As the challenger for $\mathcal{F}$, $\mathcal{F}'$ simulates and runs interactions with $\mathcal{F}$ as follows.

Setup. $\mathcal{F}'$ generates a key-pair (x_c, y_c) randomly by running G_c, which serves as the confirmer's key pair. Then $\mathcal{F}'$ runs $\mathcal{F}$, providing him as input the public keys y_s and y_c, and also the confirmer's private key x_c.

Hash Queries. When $\mathcal{F}$ requests a hash on m, $\mathcal{F}'$ makes a query for m to its own hash oracle, and receives some value $h \in G$, then it responds h to $\mathcal{F}$.

DCSSign Queries. When $\mathcal{F}$ requests a DCS on some m (it would have already queried the hash oracle on m), $\mathcal{F}'$ queries its own *Sign* oracle on message m, obtaining $\sigma \in G$. Then $\mathcal{F}'$ selects two random numbers $r, k \in \mathbb{Z}_q^*$, generates a DCS $\sigma' = (\sigma_1, \sigma_2, s, t)$ according to Eq. (5) and returns it to $\mathcal{F}$.

Confirm and Disavow Queries. Whenever $\mathcal{F}$ asks to run either $Confirm_{(S,\mathcal{F})}$ or $Disavow_{(S,\mathcal{F})}$ protocol w.r.t. a DCS message-signature pair (m, σ'), $\mathcal{F}'$ can first checks the validity of σ' by using the secret x_c and then convinces $\mathcal{F}$ by running $Confirm_{(C,\mathcal{F})}$, $Disavow_{(C,\mathcal{F})}$ respectively in the role of the confirmer C. Note that in the view of point of algorithm $\mathcal{F}$, such interactions are indistinguishable from those running in the role of the signer S. Note that for the proof of unforgeability, we actually need the witness indistinguishability property of Confirm and Disavow protocols, which does hold for the HVIZK protocols presented in Appendix B (without the need of transforming them into CZK with unprogrammable RO).

Output. Finally, if $\mathcal{F}$ halts declaring failure, $\mathcal{F}'$ declares failure too. Otherwise, $\mathcal{F}$ provides a valid and nontrivial DCS $\sigma'^* = (\sigma_1^*, \sigma_2^*, s^*, t^*)$ to $\mathcal{F}'$ on a message m^*. Then, $\mathcal{F}'$ computes $\sigma^* = \sigma_2^*/(\sigma_1^*)^{x_c^{-1}}$, which is a valid BLS signature on m^* under signer's public key y_s. A nontrivial forgery means that $\mathcal{F}$ did not query the *DCSSign* oracle on m^*, for which $\mathcal{F}'$ did not query its *Sign* oracle on m^*. Hence, (m^*, σ^*) forms a nontrivial BLS forgery.

Now we analyze the success probability and the running time of $\mathcal{F}'$. Algorithm $\mathcal{F}'$ succeeds whenever $\mathcal{F}$ does, so the success probability of $\mathcal{F}$ equals to that of $\mathcal{F}'$, i.e., $\varepsilon = \varepsilon'$. The running

time of $\mathcal{F}'$ is the running time of $\mathcal{F}$ plus the time it takes to respond q_H hash queries and q_S *DCSSign* queries, to run q_C *Confirm* queries and q_D *Disavow* queries, together the time to transform the final forged DCS into a BLS signature. Hash queries impose no overhead. Each *DCSSign* query requires $\mathcal{F}'$ to perform three exponentiations in G. For each *Confirm* query, $\mathcal{F}'$ will evaluate four paring computations and five exponentiations in G_t, while each *Disavow* query requires five paring computations and six exponentiations in G_t. The final signature transformation needs one exponentiation in G. Denote the time for pairing computation by pr, the time for exponentiation in G by ex_G, and the time for exponentiation in G_t by ex. We get that the total running time t' for F' is at most $t + (3q_S + 1) \cdot ex_G + 5(q_C + q_D) \cdot pr + (5q_C + 6q_D) \cdot ex$.

In a summary, if $\mathcal{F}$ can $(t, q_H, q_S, q_C, q_D, \varepsilon)$-forges a DCS in the proposed DCS scheme, then $\mathcal{F}'$ can $(t', q'_H, q'_S, \varepsilon')$-forges a BLS signature, where $q'_H = q_H$, $q'_S = q_S$, $\varepsilon' = \varepsilon$, and $t' \leq t + 6c \cdot (q_S + q_C + q_D)$, where $c = pr + ex_G + ex$ is a constant. □

Common Input: A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2		
Signer S (with private input x_s)		Verifier V
$a, b_1, c_1 \in_R \mathbb{Z}_q$, $z_1 = {B_1}^{b_1} \cdot {A_1}^{c_1}$, $z_2 = {D_1}^{b_1} \cdot {C_1}^{c_1}$,		
$z_3 = {B_2}^{a}$, $z_4 = {D_2}^{a}$;	(1) $\xrightarrow{(z_1, z_2, z_3, z_4)}$	$c \in_R \mathbb{Z}_q$
$c_2 = c$ - $c_1 \bmod q$,	(2) $\xleftarrow{c}$	
$b_2 = a$ - $c_2 x_s \bmod q$.	(3) $\xrightarrow{(b_1, b_2, c_1, c_2)}$	Output Accept iff $c_1 + c_2 \equiv c \bmod q$, $z_1 = {B_1}^{b_1} \cdot {A_1}^{c_1}$, $z_2 = {D_1}^{b_1} \cdot {C_1}^{c_1}$, $z_3 = {B_2}^{b_2} \cdot {A_2}^{c_2}$, $z_4 = {D_2}^{b_2} \cdot {C_2}^{c_2}$.

Figure 2. The $Confirm_{(S,V)}$ Protocol

Common Input: A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2		
Confirmer C (with private input x_c)		Verifier V
$a, b_2, c_2 \in_R \mathbb{Z}_q$, $z_1 = {B_1}^{a}$, $z_2 = {D_1}^{a}$, $z_3 = {B_2}^{b_2} \cdot {A_2}^{c_2}$,		
$z_4 = {D_2}^{b_2} \cdot {C_2}^{c_2}$;	(1) $\xrightarrow{(z_1, z_2, z_3, z_4)}$	$c \in_R \mathbb{Z}_q$
$c_1 = c$ - $c_2 \bmod q$,	(2) $\xleftarrow{c}$	
$b_1 = a$ - $c_1 x_c \bmod q$;	(3) $\xrightarrow{(b_1, b_2, c_1, c_2)}$	Output Accept iff $c_1 + c_2 \equiv c \bmod q$, $z_1 = {B_1}^{b_1} \cdot {A_1}^{c_1}$, $z_2 = {D_1}^{b_1} \cdot {C_1}^{c_1}$, $z_3 = {B_2}^{b_2} \cdot {A_2}^{c_2}$, $z_4 = {D_2}^{b_2} \cdot {C_2}^{c_2}$.

Figure 3. The $Confirm_{(C,V)}$ Protocol

	Common Input: A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2.	
Signer S (with private input x_s) $a, b_1, b_2, c_1, e, e', \beta \in_R \mathbb{Z}_q/\{1\}$, $\beta' = (D_2{}^{x_s}/C_2)^a$, $z'_1 = D_2{}^e/C_2{}^{e'}$, $z'_2 = B_2{}^e/A_2{}^{e'}$, $z_1 = \beta^{c_1} D_1{}^{b_1}/C_1{}^{b_2}$,		Verifier V
$z_2 = B_1{}^{b_1}/A_1{}^{b_2}$;	(1) $\xrightarrow{(\beta, \beta', z_1, z_2, z'_1, z'_2)}$	Iff $\beta \neq 1$ and $\beta' \neq 1$,
	(2) $\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$c_2 = c$ - $c_1 \bmod q$,		
$b'_1 = e$ - $c_2 x_s a \bmod q$, $b'_2 = e'$ - $c_2 a \bmod q$;	(3) $\xrightarrow{(b_1, b_2, b'_1, b'_2, c_1, c_2)}$	Output Accept iff $c_1 + c_2 \equiv c \bmod q$, $z_1 = \beta^{c_1} D_1{}^{b_1}/C_1{}^{b_2}$, $z_2 = B_1{}^{b_1}/A_1{}^{b_2}$, $z'_1 = \beta'^{c_2} D_2{}^{b'_1}/C_2{}^{b'_2}$, $z'_2 = B_2{}^{b'_1}/A_2{}^{b'_2}$.

Figure 4. $Disavow_{(S,V)}$ Protocol

	Common Input: A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2	
Confirmer C (with private input x_c) $a, b'_1, b'_2, c_2, e, e', \beta' \in_R \mathbb{Z}_q/\{1\}$, $\beta = (D_1{}^{x_c}/C_1)^a$, $z_1 = D_1{}^e/C_1{}^{e'}$, $z_2 = B_1{}^e/A_1{}^{e'}$, $z'_1 = \beta'^{c_2} D_2{}^{b'_1}/C_2{}^{b'_2}$,		Verifier V
$z'_2 = B_2{}^{b'_1}/A_2{}^{b'_2}$;	(1) $\xrightarrow{(\beta, \beta', z_1, z_2, z'_1, z'_2)}$	Iff $\beta \neq 1$ and $\beta' \neq 1$,
	(2) $\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$c_1 = c$ - $c_2 \bmod q$,		
$b_1 = e$ - $c_1 x_c a \bmod q$, $b_2 = e'$ - $c_1 a \bmod q$;	(3) $\xrightarrow{(b_1, b_2, b'_1, b'_2, c_1, c_2)}$	Output Accept iff $c_1 + c_2 \equiv c \bmod q$, $z_1 = \beta^{c_1} D_1{}^{b_1}/C_1{}^{b_2}$, $z_2 = B_1{}^{b_1}/A_1{}^{b_2}$, $z'_1 = \beta'^{c_2} D_2{}^{b'_1}/C_2{}^{b'_2}$, $z'_2 = B_2{}^{b'_1}/A_2{}^{b'_2}$.

Figure 5. The $Disavow_{(C,V)}$ Protocol