Designated Confirmer Signatures Revisited*

Douglas Wikström

ETH Zürich, Department of Computer Science
douglas@inf.ethz.ch

26th February 2007

Abstract

Previous definitions of designated confirmer signatures in the literature are incomplete, and the proposed security definitions fail to capture key security properties, such as unforgeability against malicious confirmers and non-transferability. We propose new definitions.

Previous schemes rely on the random oracle model or set-up assumptions, or are secure with respect to relaxed security definitions. We construct a practical scheme that is provably secure with respect to our security definition under the strong RSA-assumption, the decision composite residuosity assumption, and the decision Diffie-Hellman assumption.

To achieve our results we introduce several new relaxations of standard notions. We expect these techniques to be useful in the construction and analysis of other efficient cryptographic schemes.

*This paper is the corrected full version of the extended abstract [29] that appeared in the proceedings of the Fourth Theory of Cryptography Conference 2007.

Contents

1 Introduction

In a digital signature scheme, as introduced by Diffie and Hellman [12], a signer computes a signature of a message using its secret key, and then anybody holding the public key can verify the signature. This means that the receiver of a signature can show the signature to anybody. If the signer does not want the signer to transfer the signature it can use undeniable signatures [6] or designated verifier signatures [21], but then the holder of the signature no longer holds any indisputable evidence of a signature. Chaum [5] proposed designated confirmer signatures (DC-signatures) as a means to get the best of both worlds at the price of the introduction of a semi-trusted third party called the confirmer.

An example application for DC-signatures is a job offer scenario. Alice is offered a job by Bob and wishes to receive a formal signed offer at some point, but Bob wants to avoid that Alice shows this offer to his competitor Eve. To solve the problem Carol comes to the rescue. Bob computes a DC-signature using his own secret key and Carols public key. Then he proves to Alice that he formed the signature in this way. The DC-signature is special in that it can only be verified directly by Carol, and its distribution is indistinguishable from a distribution that can be computed using only the public keys of Carol and Bob. Furthermore, given a valid/invalid DC-signature, Carol can either convert it into a valid/invalid ordinary signature of Bob that can be verified by anybody, or she can prove that she has the ability to do this. Bob can assume that nobody can forge a signature for his public key, and that as long as Carol is honest nobody learns that he signed an offer. Alice can safely assume that Bob can not fool her, and that if Bob denies having signed an offer and Carol is honest, then Carol can prove to anybody that Bob is lying.

1.1 Previous Work

The first formal model of DC-signatures was given by Okamoto [24], but he did not consider the problem of signer coercion. Thus, a signer could be coerced into confirming/denying a signature without the randomness of the signature computation. This problem was considered by Camenisch and Michels [3], who provided stronger definitions. They also proposed both a scheme based on general primitives and more practically oriented schemes, and sketched a security proof for the general construction. In their work on verifiable encryption of discrete logarithms Camenisch and Shoup [4] give a very brief sketch of a DC-signature scheme where most interactive protocols use Schnorr-style techniques.

Goldwasser and Waisbard [20] proposed a relaxed security definition to allow the proofs of knowledge to be strong witness hiding instead of zero-knowledge, and thus allow concurrency. They give a transformation that converts an ordinary signature scheme into a DC-signature scheme secure according to their relaxed definition. They use no random oracles, but the disavowal protocol is based on general zero-knowledge techniques, and the other protocols are based on cut-and-choose techniques.

Gentry, Molnar, and Ramzan [16] considered another relaxation based on an observation originally made by Michels and Stadler [23]. Instead of computing a signature of the message directly, the signer computes a "confirmer commitment" of the message, and then sign the commitment. The constructions in [16] are efficient and do not rely on the random oracle model, but they require the existence of trusted RSA-parameters.

1.2 Our Contributions

Firstly, we take a careful look at existing definitions of DC-signatures. It turns out that two protocols that are not mentioned in previous works, are needed for successful deployment: a proof of correct conversion of a signature, and a proof that a public key is "well formed". We also observe that the definitions of security proposed by Camenisch and Michels [3], Goldwasser and Waisbard [20], and Gentry et al. [16] respectively do not ensure unforgeability when the confirmer is malicious. Furthermore, we note that the relaxed definition in [20] does not prevent transferability, which is arguably a key property of DC-signatures, and the definition in [3] is flawed and can not be satisfied at all. Thus, previous definitions do not capture the notion of DC-signatures correctly. We propose new definitions that correct these deficiencies.

Secondly, we consider how to construct a secure DC-signature scheme. We prove the security of a generic construction with respect to the new security definition. We then describe an instantiation of the generic construction that is secure under the strong RSA-assumption, the decision composite residuosity assumption, and the decision Diffie-Hellman assumption. In contrast to the scheme briefly sketched by Camenisch and Shoup [4] our scheme does not rely on the random oracle model, and it satisfies stronger security requirements than the schemes proposed in [20] and [16]. Furthermore, the setting we consider is stricter in that we do not assume the existence of a trusted key generator as is done in [4]. Despite this our scheme is practical.

Thirdly, our approach to the problem of constructing DC-signatures is different from previous in that instead of relaxing the security definitions of DC-signatures, we relax the security definitions of the primitives used to construct them and prove that weaker primitives suffice. The relaxed notions we introduce and our techniques are of general interest in the construction of efficient and provably secure cryptographic schemes.

Note to reader. *After reading Goldreich's paper [18], we have learned that in the original full version, and also in the conference version [29], our treatment of expected polynomial time adversaries was incorrect. Thus, we have reformulated our results in terms of* strict *polynomial time adversaries.*

1.3 Notation

We consider security with respect to *uniform* algorithms and our assumptions are also uniform in nature, but our results are easily translated to their non-uniform analogs. We use PT and PPT to denote the set of uniform, and uniform and probabilistic polynomial time Turing machines respectively. We let κ be the main security parameter. We say that a function $\epsilon(\kappa)$ is negligible if for every constant c and sufficiently large κ it holds that $\epsilon(\kappa) < \kappa^{-c}$. We say that $f(\kappa)$ is overwhelming if $1 - f(\kappa)$ is negligible. For any two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ we define $\mathcal{R}_1 \vee \mathcal{R}_2$ as the set of pairs $((x_1, x_2), w)$ such that $(x_1, w) \in \mathcal{R}_1$ or $(x_2, w) \in \mathcal{R}_2$. We write $\langle V(x), P(y)\rangle(z)$ to denote the output of V with private input x when it interacts with P with private input y on common input z. We write $A[S(x)]$ to denote that A has "oracle access" to an interactive Turing machine S with private input x. Formally, we assume that A has a separate pair of communication tapes over which it communicates with S. Whenever we write an expression of the form $\langle V(x), A[S(y)](z)\rangle(w)$ it is assumed that communication takes place in two phases. Before any message is communicated between V and A, A and S

may communicate freely. Then some messages are communicated between V and A. When some message is again communicated between A and S, communication is no longer possible between V and A. Finally, when A chooses part of the common input on which it interacts with V, we write $\langle V, A[S(x)](y)\rangle(z, \cdot)$. We abuse notation and say that an interactive proof is sound if it is overwhelmingly sound and we say that a protocol is a proof of knowledge only if it is also an interactive proof.

We use 1 and 0, and logical true and false interchangeably. We denote the natural numbers by $\mathbb{N}$, the integers by $\mathbb{Z}$, the integers modulo n by $\mathbb{Z}_n$, the multiplicative group modulo n by $\mathbb{Z}_n^*$ and the subgroup of squares modulo n by SQ_n. We call a prime integer p safe if $(p-1)/2$ is prime.

We use the variation of the strong RSA-assumption which says that given a product N of two random safe primes of the same bit-size and a random $g \in SQ_N$, it is infeasible to compute (b, η) such that $b^\eta = g \bmod N$ and $\eta \neq \pm 1$. We use the variation of the decision composite residuosity assumption (DCR), which says that given a product n of two random safe primes of the same bit-size, it is infeasible to distinguish the uniform distribution on elements in $\mathbb{Z}_{n^2}^*$ from the uniform distribution on nth residues in $\mathbb{Z}_{n^2}^*$. We use the decision Diffie-Hellman assumption for the subgroup G_Q of squares of $\mathbb{Z}_P^*$, where $P = 2Q+1$ is a safe prime. This says that if g generates G_Q and $\alpha, \beta, \gamma \in \mathbb{Z}_Q$ are random, then the distributions of $(g^\alpha, g^\beta, g^{\alpha\beta})$ and $(g^\alpha, g^\beta, g^\gamma)$ are indistinguishable.

1.4 Organization

In Section 2 we propose a new definition of security of DC-signatures. In Section 3 we introduce some theoretical notions needed for the result on the generic construction of DC-signatures given in Section 4. The primitives we need to instantiate the generic construction are introduced in Section 5. A detailed description of the instantiation is then given in Section 6. In Appendix A we give a more detailed account of the problems we have found with previous definitions of security of DC-signatures. Formal definitions of the cryptographic assumptions we use are given in Appendix C, and some standard definitions from the literature are given in Appendix D.

2 Definition of Designated Confirmer Signature Schemes

A DC-signature scheme consists of algorithms and interactive protocols. There are two key generation algorithms $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}$ and $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$ for the signer and confirmer respectively. There is a single signature algorithm Sig^dc that given a secret signature key, a message, and a confirmer public key outputs a signature. The signature can not be verified directly, but the confirmer can use a conversion algorithm Con^dc with his secret key and the signer public key to transform it into a signature that can be verified by anybody holding the public key of the signer using the verification algorithm Vf^dc. The protocols π_{wf} and π_c are used by the confirmer to prove that it formed its key correctly and the correctness of a conversion. The protocols π_v and π_e are used by the signer and confirmer respectively, to convince a verifier that a given DC-signature is valid and valid/invalid respectively. The protocols π_{wf} and π_c are not present in previous formalizations.

Definition 2.1 (DC Signature Scheme). A designated confirmer signature scheme $\mathcal{DCS}$ consists of algorithms $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}, \mathsf{Kg}_\mathsf{c}^\mathsf{dc}, \mathsf{Sig}^\mathsf{dc} \in \mathrm{PPT}$ and $\mathsf{Con}^\mathsf{dc}, \mathsf{Vf}^\mathsf{dc} \in \mathrm{PT}$, and interactive protocols $\pi_{wf} = (P_{wf}, V_{wf})$, $\pi_c = (P_c, V_c)$, $\pi_v = (P_v, V_v)$, and $\pi_e = (P_e, V_e)$ with the following completeness properties. For every $\kappa \in \mathbb{N}$, for every possible outputs (ssk, spk) of $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}(1^\kappa)$ and (sk, pk) of $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}(1^\kappa)$ respectively, for every $m \in \{0,1\}^*$, and for every $r, \sigma_0 \in \{0,1\}^*$, and with $\sigma_1 = \mathsf{Sig}_{ssk,r}^\mathsf{dc}(m, pk)$

1. $\mathsf{Vf}_{spk}^\mathsf{dc}(m, \mathsf{Con}_{sk}^\mathsf{dc}(\sigma_1, spk)) = 1$,
2. $\Pr[\langle V_{wf}, P_{wf}(sk)\rangle(pk) = 1]$ is overwhelming,
3. $\Pr[\langle V_c, P_c(sk)\rangle(\sigma_0, \mathsf{Con}_{sk}^\mathsf{dc}(\sigma_0, spk), pk) = 1]$ is overwhelming,
4. $\Pr[\langle V_e, P_e(sk)\rangle(m, \sigma_0, \mathsf{Vf}_{spk}^\mathsf{dc}(m, \mathsf{Con}_{sk}^\mathsf{dc}(\sigma_0, spk)), pk, spk) = 1]$ is overwhelming, and
5. $\Pr[\langle V_v, P_v(ssk, r)\rangle(m, \sigma_1, pk, spk) = 1]$ is overwhelming.

2.1 Well-Formed Keys and Signatures

We introduce the notion of well-formed confirmer keys to formalize the set of strings which behave as keys functionally, and we introduce the notion of well-formed signatures as a generalization of the set of honestly generated signatures.

Definition 2.2 (Well-Formed Keys). Let $\mathcal{DCS}$ be a DC-signature scheme. We say that the tuple $(\mathsf{Kg}_{\mathsf{c},1}^\mathsf{dc}, \mathsf{Kg}_{\mathsf{c},2}^\mathsf{dc}, \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc})$ splits $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$ if

1. $\mathsf{Kg}_{\mathsf{c},1}^\mathsf{dc}$ and $\mathsf{Kg}_{\mathsf{c},2}^\mathsf{dc}$ are probabilistic and $\mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}$ is a deterministic polynomial time (in their first parameters) algorithms,
2. on input 1^κ, $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$ computes $pk_1 = \mathsf{Kg}_{\mathsf{c},1}^\mathsf{dc}(1^\kappa)$, $sk = \mathsf{Kg}_{\mathsf{c},2}^\mathsf{dc}(pk_1)$, and $pk_2 = \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}(1^\kappa, pk_1, sk)$, and outputs $(pk, sk) = ((pk_1, pk_2), sk)$,
3. for every $\kappa \in \mathbb{N}$ and $pk_1, pk_2 \in \{0,1\}^*$ there exists at most one $sk \in \{0,1\}^*$ such that $pk_2 = \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}(1^\kappa, pk_1, sk)$, and
4. if $pk_2 = \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}(1^\kappa, pk_1, sk)$, then for every output (spk, ssk) of $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}(1^\kappa)$ and $m, r \in \{0,1\}^*$: $\mathsf{Vf}_{spk}^\mathsf{dc}(m, \mathsf{Con}_{sk}^\mathsf{dc}(\mathsf{Sig}_{ssk,r}^\mathsf{dc}(m, pk), spk)) = 1$.

We say that (pk_1, pk_2) is well-formed with respect to the splitting if $pk_2 = \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}(pk_1, sk)$ for some sk. We say that $((pk_1, pk_2), sk)$ is well-formed for such a secret key sk.

If the signer or the confirmer proves to the verifier that a DC-signature is valid/invalid relative a well-formed confirmer public key, then the verifier is confident that if converted, the result is also valid/invalid in a consistent way. Every key generator can be trivially split, but we are interested in splittings that given (pk_1, pk_2) allow a simple proof of knowledge of sk such that $((pk_1, pk_2), sk)$ is well-formed.

We abuse notation and eliminate the input 1^κ from $\mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc}$.

Definition 2.3 (Well-Formed Signature). Let $\mathcal{DCS}$ be a DC-signature scheme and let $wf : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a polynomially computable function. We say that wf is a well-formedness function with respect to $\mathcal{DCS}$ and a splitting of $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$ if for every well-formed (pk, sk), every output (spk, ssk) of $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}(1^\kappa)$, and every possible output $s = \mathsf{Con}_{sk}^\mathsf{dc}(\mathsf{Sig}_{ssk}^\mathsf{dc}(m, pk), spk)$, we have $wf(s, pk, spk) = 1$.

All honestly generated signatures are well formed, but some valid signatures may not be. It is trivial to see that there exists a well-formedness function for every DC-signature scheme, but we are interested in well-formedness functions that simplify the construction of our protocols.

2.2 Definition of Security

Assume that some splitting and well-formedness functions are fixed and define the following relations.

Definition 2.4 (Relations).

1. Well-Formed Confirmer Keys. Denote by $\mathcal{R}_{wf}$ the set of all well-formed key pairs $((pk_1, pk_2), sk)$.
2. Correct Conversion. Denote by $\mathcal{R}_c$ the set of pairs $((\sigma, s, pk, spk), sk)$ such that $(pk, sk) \in \mathcal{R}_{wf}$ and $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)$.
3. Correct Evaluation. Denote by $\mathcal{R}_e$ the set of pairs $((m, \sigma, c, pk, spk), sk)$ such that $(pk, sk) \in \mathcal{R}_{wf}$, and $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)$ for some s such that $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m, s) = c$ and $wf(s, pk, spk) = 1$.
4. Proof of Validity. Denote by $\mathcal{R}_v$ the set of pairs $((m, \sigma, pk, spk), w)$, where w is a witness that $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m, \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)) = 1$ for every sk such that $(pk, sk) \in \mathcal{R}_{wf}$.

The relation $\mathcal{R}_e$ only considers *well-formed* signatures. If a signature is not well-formed, it was by definition computed by a corrupt signer. In this case the confirmer need not protect the signer and can simply convert the signature and prove that it did so correctly. The relation $\mathcal{R}_v$ does *not* capture the set of valid signatures. It captures the set of signatures that are valid *given that the public confirmer key is well-formed.*

We now consider what each honest party or group of honest parties in a DC-signature scheme might expect from a secure implementation.

Honest Verifier. An honest verifier naturally expects that it is infeasible to convince it of a false statement. Furthermore, it seems reasonable that the verifier only accepts a proof of well-formedness of a public key if it shows that the prover knows the secret key, since otherwise it can not be confident that the confirmer actually is able to convert a signature.

Definition 2.5 (Soundness). A DC-signature scheme $\mathcal{DCS}$ is sound if π_c, π_e, and π_v are interactive proofs for the relations $\mathcal{R}_c$, $\mathcal{R}_e$, and $\mathcal{R}_v$ respectively, and π_{wf} is a proof of knowledge for the relation $\mathcal{R}_{wf}$.

Note that if a confirmer public key pk is well-formed and σ is a candidate signature for a signer public key spk, then well-formedness implies that a signature can be converted in only one way. Thus, the confirmer can not choose if a signature should be considered valid or not. Well-formedness also implies that if a signer proves the validity of σ, it can not be converted into an invalid one.

It may be dangerous for a signer to use a confirmer public key that is not well-formed. Thus, we assume that any signer (or somebody the signer trusts) executes π_{wf} with the confirmer before using its public key.

Honest Signer. It must be infeasible for the adversary to convince anybody that the honest signer has signed a message m unless this is the case. This must hold even when the adversary can ask arbitrary signature queries and execute the proof of validity protocol π_v. In other words we need a slight generalization of security against chosen message attacks [19].

We formalize the honest signer S to be a probabilistic interactive Turing machine that accepts as input a key pair (spk, ssk). Whenever it receives a message with prefix π_{wf}, $\mathsf{Sig}^{\mathsf{dc}}$, or π_v on its communication tape it halts the execution of any executing interactive protocol and proceeds as follows. Given a message (π_{wf}, pk) it executes the verifier V_{wf} of the protocol π_{wf} on common input pk. If V_{wf} accepts the proof, then S stores pk. Given a message $(\mathsf{Sig}^{\mathsf{dc}}, m, pk)$ it checks if it has stored pk. If not, it returns $\perp$, and otherwise it computes $\sigma = \mathsf{Sig}^{\mathsf{dc}}_{ssk,r}(m, pk)$ and writes σ on the communication tape. Given a message (π_v, m, σ, pk), where it previously computed $\sigma = \mathsf{Sig}^{\mathsf{dc}}_{ssk,r}(m, pk)$ it executes the prover of the protocol π_v on common input (m, σ, pk, spk) and private input some witness w that $((m, \sigma, pk, spk), w) \in \mathcal{R}_v$. When we use several copies of S below we index them for easy reference.

Experiment 2.6 (CMA-Security, $\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{DCS},A}(\kappa)$).

$$\begin{aligned} (spk, ssk) &\leftarrow \mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}(1^\kappa) \\ (m, s) &\leftarrow A[S(spk, ssk)](spk) \end{aligned}$$

If $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m, s) = 0$ *or if* S *signed* m *return 0 and otherwise 1.*

Definition 2.7 (CMA-Security). A DC-signature scheme $\mathcal{DCS}$ is secure against chosen message attacks (CMA-secure) if for every $A \in$ PPT: $\Pr[\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{DCS},A}(\kappa) = 1]$ is negligible.

In the definitions of Camenisch and Michels [3], Goldwasser and Waisbard [20], and Gentry et al. [16] security hold only with respect to *honestly* generated confirmer keys, i.e., their definitions do not ensure any form of CMA-security for the signer when the confirmer is corrupted.

The definition does not explicitly prevent an adversary to form a bit-string σ and then convince an honest verifier using π_v or π_e that this is a valid designated signature of some message m not signed by S, but this follows from the lemma below.

Lemma 2.8 (Indirect CMA-Security). *Let* $\mathcal{DCS}$ *be a sound and CMA-secure DC-signature scheme, and consider the following experiment for an adversary* A*.*

$$\begin{aligned} (spk, ssk) &\leftarrow \mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}(1^\kappa) \\ d_1 &\leftarrow \langle V_e, A[S_1(spk, ssk)](spk)\rangle(\cdot, \cdot, 1, \cdot, spk) \\ d_2 &\leftarrow \langle V_v, A[S_2(spk, ssk)](spk)\rangle(\cdot, \cdot, \cdot, spk) \end{aligned}$$

Denote by $(m_1, \sigma_1, 1, pk_1, spk)$ *and* $(m_2, \sigma_2, pk_2, spk)$ *the common inputs on which* V_e *and* V_v *are invoked. If* S_i *computed a signature of* m_i *set* $e = 0$*. If* S_i *did not store* pk_i *before the corresponding verifier received its common input, then set* $e = 0$*, and otherwise set* $e = d_1 \vee d_2$*. Then for every adversary* $A \in$ PPT *the probability* $\Pr[e = 1]$ *is negligible.*

We require that S_i stores pk_i before the verifier receives its input to model that no verifier is convinced when the confirmer public key is not well-formed.

Proof. Suppose that the lemma is false. Then there exists an adversary A such that $\Pr[e = 1]$ is non-negligible. Denote by E_i the event that S_i stored pk_i before the corresponding verifier received its common input, and that S_i did not compute a signature of m_i. Then the probability $\Pr[d_i = 1 \wedge E_i]$ is non-negligible for $i = 1$ or $i = 2$.

Consider the first case. Denote by A' the adversary in Experiment 2.6 that simulates the experiment of the lemma to A by simply forwarding requests except that if the verifier of the π_{wf} protocol accepts when A hands (π_{wf}, pk) to the simulated honest signer S_1, it invokes the knowledge extractor of π_{wf} to extract sk such that $(pk, sk) \in \mathcal{R}_{wf}$. Then it continues the simulation until A decides on the common input $(m_1, \sigma_1, 1, pk_1, spk)$, at which point it checks if pk_1 is stored by S_1. If this is the case it has already extracted some secret key sk_1 such that $(pk_1, sk_1) \in \mathcal{R}_{wf}$, and then it outputs (m_1, s), where $s = \mathsf{Con}^{\mathsf{dc}}_{sk_1}(\sigma_1, spk)$, and otherwise it outputs $\perp$. It follows from the soundness of the interactive proof π_e that conditioned on $d_1 = 1 \wedge E_i$, the probability that $((m_1, \sigma_1, 1, pk_1, spk), sk_1) \in \mathcal{R}_e$ is overwhelming. Thus, the probability that the output (m_1, s) satisfies $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m_1, s) = 0$ and $d_1 = 1$ is negligible. This implies that with non-negligible probability the adversary A' outputs a pair (m_1, s) such that S_1 never computed a signature of m_1 and $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m_1, s) = 1$. Thus, the CMA-security of $\mathcal{DCS}$ is broken.

The second case follows mutatis mutandi. The only essential difference is that if $((m_2, \sigma_2, pk_2, spk), w) \in \mathcal{R}_e$, then $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m_2, \mathsf{Con}^{\mathsf{dc}}_{sk'_2}(\sigma, spk))$ for every sk_2 such that $(pk_2, sk_2) \in \mathcal{R}_{wf}$. Thus, we can conclude directly that the output (m_1, s) satisfies $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m_1, s) = 0$ and $d_1 = 1$ is negligible. □

Honest Confirmer. Nobody except the confirmer should be able to play the role of the prover in the protocols π_{wf}, π_c, and π_e using the honest confirmers public key as common input, even after interacting with the real confirmer.

We formalize the honest confirmer C to be a probabilistic interactive Turing machine that accepts as input a key pair (pk, sk). Whenever it receives a message with prefix π_{wf}, $\mathsf{Con}^{\mathsf{dc}}$, or π_e on its input communication tape it halts the execution of any interactive protocol it is executing and proceeds as follows. Given a message (π_{wf}) it executes the prover of protocol π_{wf} on common input pk and private input sk. Given a message $(\mathsf{Con}^{\mathsf{dc}}, \sigma, spk)$ it computes $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)$, writes s on its output communication tape, and executes the prover of protocol π_c on common input (σ, s, pk, spk) and private input sk. On input (π_e, m, σ, spk) it computes $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)$. If $wf(s, pk, spk) = 1$, i.e., s is well-formed, then C computes $c = \mathsf{Vf}^{\mathsf{dc}}_{spk}(m, s)$, writes c on its output communication tape, and executes the prover of protocol π_e on common input (m, σ, c, pk, spk) and private input sk. Otherwise C writes $(\mathsf{malformed}, s)$ on its output communication tape and executes the prover of protocol π_c on common input (σ, s, pk, spk) and private input sk.

Experiment 2.9 (Impersonation-Resistance, $\mathsf{Exp}^{\mathsf{imp-res}}_{\mathcal{DCS},A}(\kappa)$).

$$
\begin{aligned}
(pk, sk) &\leftarrow \mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}(1^\kappa) \\
d_1 &\leftarrow \langle V_{wf}, A[C(pk, sk)](pk)\rangle(pk) \\
d_2 &\leftarrow \langle V_c, A[C(pk, sk)](pk)\rangle(\cdot, \cdot, pk, \cdot) \\
d_3 &\leftarrow \langle V_e, A[C(pk, sk)](pk)\rangle(\cdot, \cdot, \cdot, pk, \cdot)
\end{aligned}
$$

Return $d_1 \vee d_2 \vee d_3$.

Definition 2.10 (Impersonation Resistance). A designated confirmer signature scheme $\mathcal{DCS}$ is impersonation-resistant if for every $A \in \mathrm{PPT}$: $\Pr[\mathsf{Exp}^{\mathsf{imp-res}}_{\mathcal{DCS},A}(\kappa) = 1]$ is negligible.

Note that C never invokes P_e without executing $\mathsf{Con}^{\mathsf{dc}}$. This is without loss of generality, since $\mathsf{Con}^{\mathsf{dc}}$ is deterministic and the common input contains the extracted signature anyway. Note that this differs from the signature case, where a signer could potentially want to prove the correctness of a particular signature to several receivers.

Remark 2.11. A stronger definition would allow the adversary to interact with the confirmer and the verifier concurrently on other inputs. Unfortunately, such a definition requires the protocols π_c and π_e to be non-malleable [13] with respect to each other and themselves. General methods such as [26] can be used to construct non-malleable zero-knowledge protocols, but currently these techniques are far from practical. Thus, we do not follow this definitional path.

Honest Signer and Honest Confirmer. To start with we observe that from the point of view of an honest signer and honest verifier, or from the point of view of an honest verifier and honest confirmer, no additional requirements are natural to impose.

When the signer and the confirmer are honest we require that knowledge that the signer signed a particular message can not be transfered. Note that this is needed in the job offer scenario. Non-transferability can clearly only hold until a DC-signature has been converted.

We formalize this as follows. Let SC be the machine that simulates both S and C on inputs (pk, sk) and (spk, ssk) except for the following modifications. Given a message $(\mathsf{Sig}^{\mathsf{dc}}, m, pk')$ with $pk' = pk$ it waits for a message (m, σ), stores this, and writes σ on its communication tape. If later invoked on (π_v, m, σ, pk) it returns $\perp$ instead of invoking P_v. For $pk' \neq pk$ it behaves as S. Given a message $(\mathsf{Con}^{\mathsf{dc}}, \sigma, spk')$ such that $spk' = spk$ and (m, σ) is stored it checks if (m, σ, s) is stored for some s. If not, then it computes $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\mathsf{Sig}^{\mathsf{dc}}_{ssk}(m, pk), spk)$ and stores (m, σ, s). Finally, it writes s on its communication tape. It does not invoke the prover of π_c. If $spk' \neq spk$ or (m, σ) is not stored it behaves as C. Finally, given a message (π_e, m, σ, spk), where (m', σ) is stored for some m' it returns 0 if $m \neq m'$ and 1 otherwise. It does not execute the prover of π_e.

Intuitively, SC ***delays*** the computation of every DC-signature using the public key pk until it is converted. We want to say that if there is an adversary A that interacts with S and C, there is another adversary A' that interacts with SC such that its output is indistinguishable from that of A, despite that all its signature queries are "delayed". For this to make sense the order of messages sent to S, C, and SC must be given to the distinguisher as well. We say that an adversary is scheduled if whenever it writes a message with prefix $\mathsf{Sig}^{\mathsf{dc}}$, π_v, π_{wf}, $\mathsf{Con}^{\mathsf{dc}}$, and π_e the message and any return value (excluding the messages exchanged by the protocols that may be invoked) are stored on a special write only scheduling tape. Furthermore, when the adversary halts its output is prefixed by its scheduling tape. The additional input pk to S below is stored as a well-formed public key, and this is done in the simulation of SC as well.

Experiment 2.12 (Non-Transferability, $\mathsf{Exp}^{\mathsf{non-trans}-b}_{\mathcal{DCS},A,V}(\kappa)$).

$$((pk, sk), (spk, ssk)) \leftarrow (\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}(1^\kappa), \mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}(1^\kappa))$$

$$d \leftarrow \begin{cases} D(A(pk, spk, ssk)[SC(pk, sk, spk, ssk)]) & \text{if } b{=}0 \\ D(A(pk, spk, ssk)[S(pk, spk, ssk), C(pk, sk)]) & \text{if } b{=}1 \end{cases}$$

Definition 2.13 (Non-Transferability). A DC-signature scheme $\mathcal{DCS}$ is non-transferable if for every scheduled $A_1 \in$ PPT and every constant $c > 0$ there exists a scheduled $A_0 \in$ PPT such that for every distinguisher $D \in$ PPT: $|\Pr[\mathsf{Exp}^{\mathsf{non-trans}-1}_{\mathcal{DCS},A_1,D}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{non-trans}-0}_{\mathcal{DCS},A_0,D}(\kappa) = 1]| < \kappa^{-c}$.

Remark 2.14. Again, our experiment is not completely realistic. A stronger definition would allow the adversary to interact concurrently with S and C on other common inputs when trying to convince a verifier. Unfortunately, such definitions imply that the protocols π_{wf}, π_c, π_e, and π_v are non-malleable in a very strong sense. We are not aware of any general methods to achieve this.

Informally, a DC-signature scheme is coercion-free if a signer can reveal its secret signing key and still claim that it did not compute a particular DC-signature. Naturally, this can only hold as long as the DC-signature is not converted, or proved to be valid. Note that this is already captured in our definition of non-transferability.

The definition of non-transferability in Camenisch and Michels [3] can not be satisfied, since it requires the existence of a straight-line zero-knowledge simulator for an interactive proof without set-up assumptions. The definition of Goldwasser and Waisbard [20] only prevents the adversary from transferring confidence of validity of a signature using the confirmation protocol of the scheme. It says nothing about the possibility of using another confirmation protocol. The relaxed definition of Gentry et al. [16] explicitly allows some forms of transferability.

Most previous definitions require some form of indistinguishability of signatures computed by different signers, but this is unnecessarily strong. In any claim about a signature, the holder of the signature would disclose the identity of the claimed signer anyway, and our definition implies that anybody can generate something indistinguishable from a valid signature of any such signer.

Definition of Security. We now define security of a DC-signature scheme in the natural way.

Definition 2.15. A designated confirmer signature scheme $\mathcal{DCS}$ is secure if it is sound, CMA-secure, impersonation-resistant, and non-transferable.

On Concurrency. In our definitions the "oracle access" to the honest signer S and the honest confirmer C are sequential. Stronger definitions similar to those of Camenisch and Michels [3], where the adversary is given concurrent "oracle access", follow by giving the adversary access to several copies of S and C, each executing on the same input key pair.

3 Theoretical Tools

In this section we introduce novel variations of zero-knowledge and CCA2-security and define what it means for a signature scheme to be collision-free.

3.1 A Relaxation of Zero-Knowledge

The definition of zero-knowledge is very strong in that the simulation property must hold with respect to every verifier and *every instance* (x, w) in the relation $\mathcal{R}$ under consideration. As pointed out by Goldreich [17] it is quite natural to consider a uniform definition that only requires that it is *infeasible to find an instance* on which a verifier can gain knowledge.

In many cryptographic settings the instance can not be chosen completely freely by the adversary, e.g., the adversary may ask an honest party to prove that it performed a decryption correctly, where the keys to the cryptosystem are generated honestly. Furthermore, in some security proofs the simulator can be allowed an additional advice string *dependent* on the instance, e.g., if a decryption oracle is present in the environment where the simulator is invoked we may give the simulator decryptions of some ciphertexts. The following definition allows for both these settings.

Experiment 3.1 (Zero-Knowledge, $\mathsf{Exp}^{(T,F)-\mathsf{zk}-b}_{\pi,\mathcal{R},I,V^*,D}(\kappa)$).

$$\begin{array}{rcl} t & \leftarrow & T(1^\kappa) \\ (i, z) & \leftarrow & I(1^\kappa, t) \\ (x, w, a) & \leftarrow & F(t, i) \\ d & \leftarrow & \begin{cases} D(x, z, a, \langle V^*(z), P(w)\rangle(x)) & \text{if } b{=}0 \\ D(x, z, a, M(z, a, x)) & \text{if } b{=}1 \end{cases} \end{array}$$

Return 0 *if* $\mathcal{R}(x, w) = 0$ *and* d *otherwise.*

Definition 3.2. Let $\pi = (P, V)$ be an interactive protocol, let $T \in \mathrm{PPT}$ and $F : \{0,1\}^* \to \{0,1\}^*$, and let $\mathcal{R}$ be a relation. We say that π is (T, F)-zero-knowledge for $\mathcal{R}$ if for every verifier $V^* \in \mathrm{PPT}$ and every constant $c > 0$ there exists a simulator $M \in \mathrm{PPT}$ such that for every instance chooser $I \in \mathrm{PPT}$ and every distinguisher $D \in \mathrm{PPT}$: $|\Pr[\mathsf{Exp}^{(T,F)-\mathsf{zk}-0}_{\pi,\mathcal{R},I,V^*,D}(\kappa) = 1] - \Pr[\mathsf{Exp}^{(T,F)-\mathsf{zk}-1}_{\pi,\mathcal{R},I,V^*,D}(\kappa) = 1]| < \kappa^{-c}$.

Remark 3.3. We do not require that F is polynomial time in the definition, but the concrete protocols we present are (T, F)-zero-knowledge with efficiently computable functions F. This seems also essential to allow sequential composition as formalized below.

Remark 3.4. The definition makes sense for non-uniform adversaries as well. Furthermore, the definition can be both generalized and relaxed. One natural relaxation is to give only part of the sample t to the instance finder. Note that a probabilistic F is captured by this relaxation. A related definition gives the instance finder access to some specific oracle, i.e., we would talk about (T, F, O)-zero-knowledge for some specific oracle O. This seems to make most sense when the oracle is efficiently computable using the sample t (of which not all is given to the instance finder).

Denote by $T_1 \| T_2$ the algorithm that given 1^κ outputs $(T_1(1^\kappa), T_2(1^\kappa))$. Denote by $F_1 \| F_2$ the algorithm that on input $((t_1, t_2), (i_1, i_2))$ outputs $(F_1(t_1, i_1), F_2(t_2, i_2))$. Denote by $T_\emptyset$ the algorithm that gives $\emptyset$ as output. Denote by $F_\emptyset$ the algorithm such that $F_\emptyset(t, i) = t$, and denote by F_{id} the algorithm such that $F_{id}(t, i) = i$. Choose some canonical interpretation of strings such that the output of I is always of the form $((x_1, w_1), z)$. Then when the output of T is always of the form $t = (x_2, w_2)$, and $F(t, i) = ((x_1, x_2), (w_1, w_2), \emptyset)$, we simply say that the protocol is T-zero-knowledge.

Note that when $T = T_\emptyset$, $F = F_{id}$ and all machines in the definition are strict polynomial machines, (T, F)-zero-knowledge reduces to uniform zero-knowledge [17].

If a simulator satisfies the relaxed definition, then it can be plugged in instead of a real protocol execution polynomially many times sequentially as long as F is polynomial time. This extension is necessary in our analysis. More precisely we have the following lemma.

Experiment 3.5 (Sequential Composition, $\mathsf{Exp}^{\mu(\kappa)-(T,F)-\mathsf{zk}-b}_{\pi,\mathcal{R},I,V^*,D}(\kappa)$).

$$\begin{array}{rcl}
t & \leftarrow & T(1^\kappa) \quad z'_0 \leftarrow t \\
(i_j, z_j) & \leftarrow & I(1^\kappa, z'_{j-1}) \\
(x_j, w_j, a_j) & \leftarrow & F(t, i_j) \\
z'_j & \leftarrow & \begin{cases} \langle V^*(z_j), P(w_j)\rangle(x_j) & \text{if } b{=}0 \\ M(z_j, a_j, x_j) & \text{if } b{=}1 \end{cases} \\
d & \leftarrow & D(z'_{\mu(\kappa)})
\end{array}$$

Return 0 *if* $\mathcal{R}(x_j, w_j) = 0$ *and* d *otherwise.*

Lemma 3.6 (Sequential Composition). *Let* $\pi = (P, V)$ *be a* (T, F)*-zero-knowledge protocol with* $F \in \mathrm{PT}$ *for a relation* $\mathcal{R}$ *and let* $\mu(\kappa)$ *be bounded by a polynomial. Then for every verifier* $V^* \in \mathrm{PPT}$ *and every constant* $c > 0$ *there exists a simulator* $M \in \mathrm{PPT}$ *such that for every instance chooser* $I \in \mathrm{PPT}$ *and every distinguisher* $D \in \mathrm{PPT}$*:* $|\Pr[\mathsf{Exp}^{\mu(\kappa)-(T,F)-\mathsf{zk}-0}_{\pi,\mathcal{R},I,V^*,D}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mu(\kappa)-(T,F)-\mathsf{zk}-1}_{\pi,\mathcal{R},I,V^*,D}(\kappa) = 1]| < \kappa^{-c}$

Proof. Consider a verifier V^*, a constant $c > 0$, an instance chooser I, and a distinguisher D. Let M be the simulator for V^* and a constant c' such that $\mu(\kappa)\kappa^{-c'} < \kappa^{-c}$, guaranteed by the (T, F)-zero-knowledge property, and write H^b for the algorithm that simulates the experiment $\mathsf{Exp}^{\mu(\kappa)-(T,F)-\mathsf{zk}-b}_{\pi,\mathcal{R},I,V^*,D}(\kappa)$.

Denote by H^0_l the machine H^0, except that if $j \leq l$, then the simulator behaves as H^1. Thus, H^0_0 is identical to H^0 and H^0_μ is identical to H^1. Denote by I_l the instance chooser that simulates H^0_{l-1} until (i_l, z_l) is output in the simulation. Denote by D_l the distinguisher that takes z'_{l+1} as input and computes z_μ exactly as is done in the simulation H^0, by simulating the honest prover. It follows that H^0_{l-1} is identically distributed to $\mathsf{Exp}^{(T,F)-\mathsf{zk}-0}_{\pi,\mathcal{R},I_l,V^*,D_l}(\kappa)$. and H^0_l is identically distributed to $\mathsf{Exp}^{(T,F)-\mathsf{zk}-1}_{\pi,\mathcal{R},I_l,V^*,D_l}(\kappa)$.

Suppose that the lemma is false. Then there exists a verifier V^*, an instance chooser I, and a distinguisher D such that $|\Pr[H^0 = 1] - \Pr[H^1 = 1]| \geq \kappa^{-c}$, for the simulator identified above. A hybrid argument shows that there exists a fixed l such that $|\Pr[H^0_{l-1} = 1] - \Pr[H^0_l = 1]| \geq \kappa^{-c}/\mu(\kappa) \geq \kappa^{-c'}$, but this contradicts the (T, F)-zero-knowledge property and the lemma must be true. □

3.2 Cryptosystems With Labels and Δ-CCA2-Security

Our starting point is the generalization of CCA2-security for cryptosystems with labels introduced by Shoup and Gennaro [27].

This simply means that the input to the cryptosystem is not only a message, but also a label, and similarly for the decryption algorithm. In other words we use the following definition.

Definition 3.7 (Public Key Cryptosystem With Labels). A public key cryptosystem with labels $\mathcal{CS} = (\mathsf{CSKg}, \mathsf{Enc}, \mathsf{Dec})$ consists of three probabilistic polynomial-time algorithms.

1. A key generation algorithm CSKg that on input 1^κ outputs a public key pk and a private key sk. The public key defines a plaintext space $\mathcal{M}_{pk}$.
2. An encryption algorithm Enc that on input a public key pk, a label L, and a message $m \in \mathcal{M}_{pk}$ outputs a ciphertext c.
3. A decryption algorithm Dec that on input a private key sk, a label L, and a ciphertext c outputs a message $m \in \mathcal{M}_{pk}$ or $\perp$.

Furthermore, for every output (sk, pk) of $\mathsf{CSKg}(1^\kappa)$, every label $L \in \{0,1\}^*$, and every $m \in \mathcal{M}_{pk}$ it must hold that $\mathsf{Dec}_{sk}(L, \mathsf{Enc}_{pk}(L, m)) = m$.

The definition of CCA2-security is strict in that the indistinguishability property of ciphertexts holds for *any* two messages. In our setting a weaker property suffices, namely that any two encrypted signatures from the *same* signer are indistinguishable. Thus, we introduce the following relaxed definition.

Experiment 3.8 (Δ-CCA2-Security, $\mathsf{Exp}_{\mathcal{CS},A}^{\Delta-\mathsf{cca2}-b}(\kappa)$).

$$\begin{array}{rcl}
(pk, sk) & \leftarrow & \mathsf{CSKg}(1^\kappa) \\
(r, m_0, m_1, \mathrm{state}) & \leftarrow & A^{\mathsf{Dec}_{sk}(\cdot,\cdot)}(\mathsf{choose}, pk) \\
c & \leftarrow & \mathsf{Enc}_{pk}(\Delta(r, m_b)) \\
d & \leftarrow & A^{\mathsf{Dec}_{sk}(\cdot,\cdot)}(\mathsf{guess}, \mathrm{state}, c)
\end{array}$$

Interpret $\Delta(r, m_b)$ as a pair (L, m_b'). The experiment returns 0 if $\mathsf{Dec}_{sk}(\cdot,\cdot)$ was queried on (L, c), and otherwise d.

Definition 3.9 (Δ-CCA2-Security). Let $\Delta \in \mathrm{PPT}$. A public key cryptosystem $\mathcal{CS}$ with labels is said to be Δ-CCA2-secure if for every adversary $A \in \mathrm{PPT}$:
$|\Pr[\mathsf{Exp}_{\mathcal{CS},A}^{\Delta-\mathsf{cca2}-0}(\kappa) = 1] - \Pr[\mathsf{Exp}_{\mathcal{CS},A}^{\Delta-\mathsf{cca2}-1}(\kappa) = 1]|$ is negligible in κ.

Remark 3.10. It is easy to transform a CCA2-secure secure cryptosystem constructed using hash-proofs [10] into a CCA2-secure cryptosystem with labels: simply let the label be part of the input to the collision-free hash function.

3.3 Collision-Free Signature Schemes

We say that a signature scheme is collision-free if it is infeasible to find two distinct messages and a signature such that the signature is a valid with respect to both messages, even if the adversary is given the honestly generated secret key and the public key.

Although this property does not follow from the CMA-security of a signature scheme it is in fact not very demanding, e.g., we do not need to take any additional precautions in the construction of our signature scheme to satisfy this property.

Definition 3.11 (Collision-Free). Let $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme, and define a random variable $(spk, ssk) = \mathsf{Kg}(1^\kappa)$. Then $\mathcal{SS}$ is collision-free if for every $A \in \mathrm{PPT}$ the probability $\Pr[A(spk, ssk) = (m_0, m_1, s) \wedge m_0 \neq m_1 \wedge \mathsf{Vf}_{spk}(m_0, s) = \mathsf{Vf}_{spk}(m_1, s) = 1]$ is negligible.

4 A Generic Construction of Designated Confirmer Signatures

It is natural to construct DC-signatures from a CMA-secure signature scheme and a CCA2-secure cryptosystem. A signer holds a secret key for the signature scheme and the confirmer holds a secret key of the cryptosystem. A DC-signature is simply an ordinary signature encrypted with the cryptosystem, conversion corresponds to decryption, and zero-knowledge proofs of knowledge are used to instantiate the protocols. The theorem below implies that this is secure, but for most signature schemes and cryptosystems it is prohibitively inefficient.

As a first step in the construction of an *efficient* DC-signature scheme we prove that weaker primitives suffice to construct a secure DC-signature scheme, but the basic idea is the same.

4.1 The Construction

Let $\mathcal{CS} = (\mathsf{CSKg}, \mathsf{Enc}, \mathsf{Dec})$ be a cryptosystem with labels and let $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme with fixed size signatures (this is easy to ensure by padding) that fit in the plaintext space of $\mathcal{CS}$. Define $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}$ to compute $(spk, ssk) = \mathsf{Kg}(1^\kappa)$ and $s_\perp = \mathsf{Sig}_{ssk}(\perp)$, where $\perp$ is a special symbol, and output $((spk, s_\perp), ssk)$ (we drop $s_\perp$ from our notation when convenient). Define $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}(1^\kappa)$ to output $\mathsf{CSKg}(1^\kappa)$, and define Vf^dc to be Vf except that $\mathsf{Vf}_{spk}^\mathsf{dc}(\perp, \cdot) = 0$. On input ssk, m, and pk the DC-signature algorithm Sig^dc is defined to compute $s = \mathsf{Sig}_{ssk}(m)$ and $\sigma = \mathsf{Enc}_{pk}(spk, s)$, and output σ. On input sk, σ, and spk the conversion algorithm Con^dc is defined to output $s = \mathsf{Dec}_{sk}(spk, \sigma)$. In other words, the public signature key spk is used as a label. Let $(\mathsf{Kg}_{\mathsf{c},1}^\mathsf{dc}, \mathsf{Kg}_{\mathsf{c},2}^\mathsf{dc}, \mathsf{Kg}_{\mathsf{c},3}^\mathsf{dc})$ be a splitting of $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$ and let wf be some well-formedness function with respect to $\mathcal{DCS}$. Let π_{wf}, π_c, π_e, and π_v be interactive protocols, complete with respect to the relations $\mathcal{R}_{wf}$, $\mathcal{R}_c$, $\mathcal{R}_e$, and $\mathcal{R}_v$. It is easy to see that $\mathcal{DCS} = (\mathsf{Kg}_\mathsf{s}^\mathsf{dc}, \mathsf{Kg}_\mathsf{c}^\mathsf{dc}, \mathsf{Con}^\mathsf{dc}, \mathsf{Vf}^\mathsf{dc}, \pi_{wf}, \pi_c, \pi_e, \pi_v)$ is a DC-signature scheme.

The algorithm $\Delta(r, (r', m))$ first computes $(spk, ssk) = \mathsf{Kg}_r(1^\kappa)$ and $s = \mathsf{Sig}_{ssk,r'}(m)$, and then outputs (spk, s). Define $T_{hs}(1^\kappa) = (\mathsf{Kg}_\mathsf{c}^\mathsf{dc}(1^\kappa), \mathsf{Kg}_\mathsf{s}^\mathsf{dc}(1^\kappa))$. Define F_{hs} to take as input the pair $((pk, sk, spk, ssk), (r, m, m'))$, compute $\sigma = \mathsf{Sig}_{ssk,r}^\mathsf{dc}(m)$, and output $((m', \sigma, \mathsf{Vf}_{spk}^\mathsf{dc}(m', \mathsf{Con}_{sk}^\mathsf{dc}(\sigma)), pk, spk), sk, \emptyset)$. Let $T_{cs}(1^\kappa)$ simply output $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}(1^\kappa)$. Define F_{cs} to take input $((pk, sk), (m, \sigma, c, spk))$ and output the tuple $((m, \sigma, c, pk, spk), sk, \mathsf{Con}_{sk}^\mathsf{dc}(\sigma, spk))$.

Theorem 4.1. [1] *Suppose that $\mathcal{CS}$ is Δ-CCA2-secure and that $\mathcal{SS}$ is CMA-secure and collision-free, and that π_{wf}, π_c, π_e, and π_v are proofs of knowledge for the relations $\mathcal{R}_{wf}$, $\mathcal{R}_c$, $\mathcal{R}_e$, and $\mathcal{R}_v$. Suppose that π_{wf} and π_c are $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$-zero-knowledge for the relations $\mathcal{R}_{wf}$ and $\mathcal{R}_c$ respectively. Suppose that π_e is both (T_{hs}, F_{hs})-zero-knowledge and (T_{cs}, F_{cs})-zero-knowledge for the relation $\mathcal{R}_e$. Suppose π_v is $\mathsf{Kg}_\mathsf{s}^\mathsf{dc}$-zero-knowledge for the relation $\mathcal{R}_v$. Then $\mathcal{DCS}$ is secure.*

[1] Camenisch and Michels [3] claim a similar, but weaker, theorem according to their definition, but as explained above their definition can not be satisfied, and only a proof sketch is given.

4.2 On the Use of Two Distinct Weak Simulators

Perhaps the most interesting of our techniques is the use of two simulators for the same protocol, of which one requires additional advice. Consider the problem of constructing a black box-reduction of a successful attacker A against non-transferability into a successful attacker A' against the Δ-CCA2-security of the underlying cryptosystem. The Δ-CCA2-attacker A' takes a public key pk as input and must simulate the non-transferability experiment to the adversary without using the secret key sk. At some point A' outputs a random bit string r and two messages (r_0, m_0) and (r_1, m_1) to the Δ-CCA2-experiment, and it is given a ciphertext $\sigma = \mathsf{Enc}_{pk}(spk, \mathsf{Sig}_{ssk,r_b}(m_b))$ for a random $b \in \{0,1\}$, where $(spk, ssk) = \mathsf{Kg}_r(1^\kappa)$. The ciphertext σ is used in the simulation somehow, and finally A' outputs a bit. The simulation involves converting signatures, but A' may use its decryption oracle to answer such queries, as long as it never asks for a decryption of σ.

We observe that when the protocol π_e is simulated for some DC-signature $\sigma' \neq \sigma$ computed by A, the simulator is free to invoke the decryption oracle on σ', i.e., a (T_{cs}, F_{cs})-zero-knowledge simulator is sufficient. On the other hand, for the particular signature σ we can not proceed in this way, since that would violate the rules of the Δ-CCA2-experiment, but since σ is computed honestly using honestly formed signature keys a (T_{hs}, F_{hs})-zero-knowledge simulator suffices.

4.3 Analysis of the Construction

Proof (Theorem 4.1). The theorem follows immediately from the following lemmas.

Lemma 4.2. *The scheme is sound.*

Proof. This follows by construction, since every proof of knowledge is a proof. □

Lemma 4.3. *The scheme is CMA-secure.*

Proof. Suppose that there is an adversary A that breaks the CMA-security of $\mathcal{DCS}$, i.e., $\Pr[\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{DCS},A}(\kappa) = 1]$ is non-negligible.

Denote by $\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa)$ experiment $\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{DCS},A}(\kappa)$ except that instead of executing P_v the experiment invokes the $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}$-zero-knowledge simulator of the protocol π_v. It follows from the zero-knowledge property and Lemma 3.6 that $\Pr[\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa) = 1]$ is non-negligible.

Denote by A' the adversary in the ordinary CMA-experiment (Experiment D.2) with $\mathcal{SS}$ that on input spk simulates $\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa)$ except that instead of computing $\mathsf{Sig}^{\mathsf{dc}}_{ssk,r}(m, pk)$ on a query (m, pk) it requests a signature s of m from its $\mathsf{Sig}_{ssk}(\cdot)$-oracle and computes $\sigma = \mathsf{Enc}_{pk}(spk, s)$. By definition of the ordinary CMA-experiment this does not change the distribution of $\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa)$. When A halts, A' outputs the pair (m, s) output by the adversary A in the simulation of $\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa)$. The output of the experiment $\mathsf{Exp}^{\mathsf{cma}*}_{\mathcal{DCS},A}(\kappa)$ only equal 1 if $\mathsf{Vf}^{\mathsf{dc}}_{spk}(m, s) = 1$ and S never computed a signature of m. Thus, A' breaks the ordinary CMA-security of $\mathcal{SS}$, since $\mathsf{Vf} = \mathsf{Vf}^{\mathsf{dc}}$. □

Lemma 4.4. *The scheme is impersonation-resistant.*

Proof. Suppose that there is an adversary A that breaks the impersonation-resistance of $\mathcal{DCS}$. To start with we describe a modification to the experiment that only changes its distribution negligibly. We denote by $\mathsf{Exp}^{\mathsf{imp-res}*}_{\mathcal{DCS},A}(\kappa)$, the original experiment, that is $\mathsf{Exp}^{\mathsf{imp-res}}_{\mathcal{DCS},A}(\kappa)$, except for the following modifications. Instead of executing the prover of π_{wf} and π_c respectively, the experiment invokes the corresponding $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}$-zero-knowledge simulator. Instead of executing the prover of the protocol π_e it invokes the (T_{cs}, F_{cs})-zero-knowledge simulator. Recall that this simulator requires an additional advice string, namely the decryption of the DC-signature σ that is part of the common input (m, σ, c, pk, spk) in an invokation. It follows from the zero-knowledge properties of these protocols and Lemma 3.6 that the probability $\Pr[\mathsf{Exp}^{\mathsf{imp-res}*}_{\mathcal{DCS},A}(\kappa) = 1]$ is non-negligible.

Denote by A_{bas} the machine in the Δ-CCA2-experiment that accepts pk as input and simulates the experiment $\mathsf{Exp}^{\mathsf{imp-res}*}_{\mathcal{DCS},A}(\kappa)$ except that instead of computing $\mathsf{Dec}_{sk}(spk, \sigma)$ when necessary, either to output it or to give it as advice to a (T_{cs}, F_{cs})-zero-knowledge simulator, it requests the plaintext s of (spk, σ) from its $\mathsf{Dec}_{sk}(\cdot, \cdot)$ oracle. By definition we know that the distribution of A_{bas} is identical to the distribution of $\mathsf{Exp}^{\mathsf{imp-res}*}_{\mathcal{DCS},A}(\kappa)$.

Denote by d_1, d_2, and d_3 the random variables in the simulation. Then either $\Pr[d_1 = 1]$, $\Pr[d_2 = 1]$, or $\Pr[d_3 = 1]$ is non-negligible. Suppose that $\Pr[d_1 = 1]$ is non-negligible. Denote by A_{ext} the adversary in the Δ-CCA2-experiment that accepts pk as input and simulates A_{bas}, except that when A interacts with V_{wf} on common input pk and the verifier accepts, then A_{ext} invokes the knowledge extractor of π_{wf}. It follows that A_{ext} outputs the secret key sk corresponding to pk in expected polynomial time. This clearly contradicts the Δ-CCA2-security of $\mathcal{CS}$.

To see that $\Pr[d_2 = 1]$ and $\Pr[d_3 = 1]$ are negligible, apply the same argument mutatis mutandi and note that every witness sk that $((\sigma, s, pk), sk) \in \mathcal{R}_c$ or that $((m, \sigma, c, pk, spk), sk) \in \mathcal{R}_e$ satisfies $(pk, sk) \in \mathcal{R}_{wf}$ by definition. Thus, it follows in each case that the Δ-CCA2-security of $\mathcal{CS}$ is broken, and the claim follows.

Note that in the argument above the (T_{cs}, F_{cs})-simulator requires additional advice, but the needed advice is present in the form of the decryption oracle. □

Lemma 4.5. *The scheme is non-transferable.*

Proof. Denote by $\gamma > 0$ a constant to be determined below. It is used to define how well our zero-knowledge simulators simulate a real execution.

Given an adversary $A_1 \in$ PPT we define another adversary $A_0 \in$ PPT that runs A_1 as a black-box. It simulates A_1 except that whenever A_1 submits a message $(\mathsf{Sig}^{\mathsf{dc}}, m, pk)$ to S it forwards it to SC. Then it computes $\sigma = \mathsf{Enc}_{pk}(spk, s_\perp)$, stores (m, σ), and sends it to SC. When SC returns σ it writes σ on the communication tape of A_1. Furthermore, any invokation of P_v is replaced by an invokation of the $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}$-zero-knowledge simulator of the protocol π_v with the constant γ. Whenever A_1 submits a message $(\mathsf{Con}^{\mathsf{dc}}, \sigma, spk)$ such that (m, σ) is stored A_0 checks if it has stored (m, σ, s). If not, it submits $(\mathsf{Con}^{\mathsf{dc}}, \sigma, spk)$ to SC and waits for and stores the reply s. Finally, it writes s on the communication tape of A_1, and invokes the $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}$-zero-knowledge simulator of π_c with the constant γ. Whenever A_1 submits a message (π_e, m', σ, spk) where (m, σ) is stored it sets $c = 0$ if $m' \neq m$ and $c = 1$ otherwise, and writes c on the communication tape of A_1 and invokes the (T_{hs}, F_{hs})-zero-knowledge simulator of π_e with the constant γ. All other messages submitted to S or C by A_1 and the protocol executions they give rise to are simply forwarded to SC. Note that all other executions involve either a σ not computed by A_0 or a public key $spk' \neq spk$.

The reader should note at this point that none of the zero-knowledge simulators invoked by the adversary A_0 require an additional advice string. We stress that there are no apriori guarantees that the simulated proofs "look right", since the simulators are invoked for false statements. This is something that must be proved.

The adversary A_0 clearly runs in polynomial time, since A_1 does and the zero-knowledge simulators do as well. We claim that the adversary A_0 shows that the DC-signature scheme is non-transferable.

Denote by $\mathsf{Exp}^{\mathsf{non-trans-}1*}_{\mathcal{DCS},A_1,V}(\kappa)$ the experiment $\mathsf{Exp}^{\mathsf{non-trans-}1}_{\mathcal{DCS},A_1,V}(\kappa)$ except for the following modifications:

1. In the simulation of S the $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{s}}$-zero-knowledge simulator of π_v with the constant γ is invoked instead of the prover P_v.
2. In the simulation of C the $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}$-zero-knowledge simulator of π_c with the constant γ is invoked instead of the prover P_c.
3. On input (π_e, m, σ, spk) such that σ was computed by S in the simulation of C, the (T_{hs}, F_{hs})-zero-knowledge simulator of π_e with the constant γ is invoked instead of the prover P_e.
4. On other inputs of the form (π_e, m, σ, spk') in the simulation of C the (T_{cs}, F_{cs})-zero-knowledge simulator of π_e with the constant γ is invoked instead of the prover P_e. Note that spk' may equal spk if σ was not computed by S and that the simulator requires an additional advice string.

Define $\mathsf{Exp}^{\mathsf{non-trans-}0*}_{\mathcal{DCS},A_0,V}(\kappa)$ correspondingly. Then the claim below follows from Lemma 3.6 and the zero-knowledge properties of the protocols π_v, π_c, and π_e.

Claim 1. For every constant $\gamma' > 0$, we can choose the constant c such that we have $|\Pr[\mathsf{Exp}^{\mathsf{non-trans-}b*}_{\mathcal{DCS},A_b,V}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{non-trans-}b}_{\mathcal{DCS},A_b,V}(\kappa) = 1]| < \kappa^{-\gamma'}$.

Denote by T^0 the simulation of the experiment $\mathsf{Exp}^{\mathsf{non-trans-}0*}_{\mathcal{DCS},A_0,V}(\kappa)$. Denote by T^1 the simulation of the experiment $\mathsf{Exp}^{\mathsf{non-trans-}1*}_{\mathcal{DCS},A_1,V}(\kappa)$, except that whenever A_1 submits a message $(\mathsf{Sig}^{\mathsf{dc}}, m, pk)$ to S it stores the result (m, σ). Furthermore, whenever A_1 submits (π_e, m, σ, spk) to C and (m', σ) is stored for some m', it sets $c' = 0$ if $m' \neq m$ and $c' = 1$ otherwise, and replaces the answer c given by C, by c'.

Claim 2. $|\Pr[\mathsf{Exp}^{\mathsf{non-trans-}1*}_{\mathcal{DCS},A_1,V}(\kappa) = 1] - \Pr[T^1 = 1]|$ is negligible.

Proof. This follows from the collision-freeness of the signature scheme $\mathcal{SS}$. Note that the simulation T^1 can take the keys (spk, ssk) of S as input. The only way the difference in the claim can be non-negligible is if the adversary A_1 submits a message (π_e, m, σ, spk) to C, where (m', σ) is stored, $s = \mathsf{Con}^{\mathsf{dc}}_{sk}(\sigma, spk)$, and $\mathsf{Vf}_{spk}(m, s) = \mathsf{Vf}_{spk}(m', s) = 1$. However, if this happens with non-negligible probability the collision-freeness of $\mathcal{SS}$ is clearly broken. □

Suppose now that $|\Pr[\mathsf{Exp}^{\mathsf{non-trans-}1}_{\mathcal{DCS},A_1,V}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathsf{non-trans-}0}_{\mathcal{DCS},A_0,V}(\kappa) = 1]| \geq \kappa^{-\gamma''}$ for some constant γ'' and κ in an infinite index set $\mathcal{N}$. Then Claims 1 and 2 imply that $|\Pr[T^1 = 1] - \Pr[T^0 = 1]| \geq \frac{1}{2}\kappa^{-\gamma''}$ for a suitable choice of γ, and this is non-negligible. Assume without loss that A_1 asks for at most $t(\kappa)$ signatures from S, where $t(\kappa)$ is some polynomial. Denote by T_i^0 the simulation of T^0 except that for $l = 1, \ldots, i$, the lth query of the form $(\mathsf{Sig}^{\mathsf{dc}}, m, pk)$ of A_1 is treated exactly as it would have been treated in the simulation of T^1. This implies that the distribution of T_t^0 is identical to the distribution of T^1. Note that here it is *essential* that the simulator uses its oracle access in exactly the same way that A_1 would, since otherwise the scheduling tape would be different and the above claim would not hold. A hybrid argument then implies that there exists a fixed l such that $|\Pr[T_{l-1}^0 = 1] - \Pr[T_l^0 = 1]|$ is non-negligible, since $t(\kappa)$ is polynomial.

We are finally ready to construct an adversary A_{cca} to the Δ-CCA2-security of $\mathcal{CS}$. The adversary waits for a public key pk and then starts the simulation of T_{l-1}^0 using this key. Whenever a message must be decrypted it invokes its $\mathsf{Dec}_{sk}(\cdot,\cdot)$-oracle. This holds both when it must convert signatures and when it must provide a simulator with an additional advice string. The simulation is continued until A_1 submits its lth message of the form $(\mathsf{Sig}^{\mathsf{dc}}, m, pk)$. Then the adversary A_{cca} outputs $(r, (r_0, M_0), (r_1, M_1))$ with $M_0 = \perp$ and $M_1 = m$, where r is the randomness that was used to compute $(spk, ssk) = \mathsf{Kg}_r(1^\kappa)$, r_0 is the randomness used to compute $s_\perp$, and r_1 is fresh randomness. The experiment returns a ciphertext $\sigma = \mathsf{Enc}_{pk}(\Delta(r, (r_b, M_b)))$ for a random bit b (recall that $\Delta(r, (r_b, M_b)) = (spk, \mathsf{Sig}_{ssk, r_b}(M_b)))$. The adversary A_{cca} then continues the simulation of T_l^0 but uses σ instead of computing this, and it uses $s = \mathsf{Sig}^{\mathsf{dc}}_{ssk, r_1}(m)$ when A_0 submits $(\mathsf{Con}^{\mathsf{dc}}, \sigma, spk)$ to C. It follows that if $b = 0$, then the output of A_{cca} is distributed as T_{l-1}^0 and otherwise as T_l^0. This contradicts the Δ-CCA2-security of $\mathcal{CS}$, since by construction A_{cca} never queries the $\mathsf{Dec}_{sk}(\cdot,\cdot)$-oracle on the input (spk, σ).

We conclude that the scheme is non-transferable. □

□

5 Concrete Tools

In this section we present the tools we need to instantiate the generic DC-signature scheme with an efficient concrete scheme under standard complexity assumptions.

5.1 A Twin-Moduli Signature Scheme

To prove the existence of the scheme presented below we must assume that an arbitrary bit-string can be embedded into a prime in an efficient way. We assume that there is an efficient algorithm $\mathsf{Emb}_f^{f'}$ that given $n \in [0, 2^\kappa - 1]$ with overwhelming probability finds $s \in [2^{f(\kappa)-1}, 2^{f(\kappa)-1} + 2^{f(\kappa)-f'(\kappa)} - 1]$ such that $e = 2^{f(\kappa)}n + s$ is prime. We call this assumption the (f, f')-Embedding Assumption. In practice this is not a problem for reasonable f and f'.

Definition 5.1 ((f, f')-Embedding Assumption). The (f, f')-embedding assumption, for functions $f, f' : \mathbb{N} \to \mathbb{N}$, with $f(\kappa) > f'(\kappa)$, states that there exists $\mathsf{Emb}_f^{f'} \in$ PPT that on input 1^κ and $n \in [0, 2^\kappa - 1]$ with overwhelming probability outputs an integer $s \in [2^{f(\kappa)-1}, 2^{f(\kappa)-1} + 2^{f(\kappa)-f'(\kappa)} - 1]$ such that $e = 2^{f(\kappa)}n + s$ is prime.

The Signature Scheme. The twin-moduli signature scheme, $\mathcal{SS}^2 = (\mathsf{Kg}^2, \mathsf{Sig}^2, \mathsf{Vf}^2)$, is based on using two sets of RSA-parameters and the embedding algorithm $\mathsf{Emb}_f^{f'}$.

Denote by κ_r a security parameter such that $2^{-\kappa_r}$ is negligible in κ. On input 1^κ the key generator Kg^2 chooses $\kappa/2$-bit safe primes p_0, q_0, p_1, and q_1 randomly, defines $N_0 = p_0q_0$ and $N_1 = p_1q_1$, chooses $g_0 \in SQ_{N_0}$ and $g_1 \in SQ_{N_1}$ randomly, and outputs $((N_0, g_0, N_1, g_1), (p_0, q_0, g_0, p_1, q_1, g_1))$. Set $\kappa_p = f(2\kappa_r + \kappa_m + 1)$ and $\kappa_p' = f'(2\kappa_r + \kappa_m + 1)$. The signature algorithm Sig^2 takes as input a private key $(p_0, q_0, g_0, p_1, q_1, g_1)$ and a message $m \in [0, 2^{\kappa_m} - 1]$. It chooses $r \in [2^{2\kappa_r+\kappa_m}, 2^{2\kappa_r+\kappa_m} + 2^{\kappa_r+\kappa_m} - 1]$ randomly. Then it computes

$$\begin{aligned} s_0 &= \mathsf{Emb}_f^{f'}(r+m) & e_0 &= 2^{\kappa_p}(r+m) + s_0 & z_0 &= g_0^{1/e_0} \bmod N_0 \\ s_1 &= \mathsf{Emb}_f^{f'}(r) & e_1 &= 2^{\kappa_p}r + s_1 & z_1 &= g_1^{1/e_1} \bmod N_1 \ . \end{aligned}$$

Finally, it outputs (r, z_0, s_0, z_1, s_1). The verification algorithm Vf^2 takes as input a public key (N_0, g_0, N_1, g_1), a message $m \in \{0,1\}^{\kappa_m}$, and a candidate signature (r, z_0, s_0, z_1, s_1). It computes $e_0 = 2^{\kappa_p}(r+m) + s_0$ and $e_1 = 2^{\kappa_p}r + s_1$, and verifies that $r \in [1, 2^{2\kappa_r+\kappa_m+1} - 1]$, $s_0, s_1 \in [0, 2^{\kappa_p} - 1]$, that e_0 and e_1 are odd, and that $z_0^{e_0} = g_0 \bmod N_0$ and $z_1^{e_1} = g_1 \bmod N_1$. The basic idea of the scheme is similar to an idea of Cramer et al. [8], but the proposition below does not follow from their work. Using a collision-free hash function $H : \{0,1\}^* \to [0, 2^{\kappa_m} - 1]$ it can be used to sign messages of any length.

Remark 5.2. If the signature scheme is used as a standalone scheme, one can choose $r \in [0, 2^{\kappa_m+\kappa_r} - 1]$ instead. This scheme is of independent interest, since the verification of a signature is more efficient than [9, 15].

Proposition 5.3. *The scheme exists under the (f, f')-Embedding assumption and is CMA-secure and collision-free under the strong RSA-assumption.*

There are two key observations in the analysis: the signature oracle can be simulated without the factorization of both moduli, and the adversary can not anticipate for which of the two moduli the factorization is known. Thus, if the adversary breaks the scheme it can not avoid helping us to break the strong RSA-assumption.

Proof (Proposition 5.3). We first prove that the scheme is CMA-secure. Consider an arbitrary adversary A taking part in Experiment D.2. We build a machine A_b for $b \in \{0,1\}$ that simulates the CMA-experiment to A and show that $\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{SS}^2,A}(\kappa)$ must be negligible under the strong RSA-assumption.

Let $t = t(\kappa)$ be the number of queries asked by A. The adversary A_b accepts as input a pair (N, h), where $N = pq$ and $h \in SQ_N$ as input. It chooses t random values $\tau_1, \ldots, \tau_t \in [2^{2\kappa_r+\kappa_m}, 2^{2\kappa_r+\kappa_m} + 2^{\kappa_r+\kappa_m} - 1]$ and computes $\zeta_l = \mathsf{Emb}_f^{f'}(\tau_l)$. Then it defines $\rho_l = 2^{\kappa_p}\tau_l + \zeta_l$ and computes $g = h^{\prod_{l=1}^t \rho_l} \bmod N$. Finally, it sets $(N_b, g_b) = (N, g)$, defines $((N_{1-b}, g_{1-b}), (p_{1-b}, q_{1-b}, g_{1-b}))$ as in the key generation algorithm, and hands (N_0, g_0, N_1, g_1) to A. Denote by p_b and q_b the factors of N_b (that are unknown to A_b).

The machine A_b simulates the $\mathsf{Sig}_{(p_0,q_0,g_0,p_1,q_1,g_1)}(\cdot)$-oracle to A as follows. The adversary A asks at most t queries $m_1, \ldots, m_t \in [0, 2^{\kappa_m} - 1]$ due to its running time. The ith query m_i is answered by A_b as follows.

- If $b = 0$, then A_b knows (p_1, q_1, g_1) and computes $r_i = \tau_i - m_i$ and

$$\begin{aligned} s_{i,0} &= \zeta_i \ , & s_{i,1} &= \mathsf{Emb}_f^{f'}(r_i) \ , \\ e_{i,0} &= \rho_i \ , & e_{i,1} &= 2^{\kappa_p} r_i + s_{i,1} \ , \\ z_{i,0} &= h^{\frac{1}{\rho_i}\prod_{l=1}^t \rho_l} \bmod N_0 \ , & z_{i,1} &= g_1^{1/e_{i,1}} \bmod N_1 \ . \end{aligned}$$

- If $b = 1$, then A_b knows (p_0, q_0, g_0) and computes $r_i = \tau_i$ and

$$\begin{aligned} s_{i,0} &= \mathsf{Emb}_f^{f'}(m_i + r_i) \ , & s_{i,1} &= \zeta_i \ , \\ e_{i,0} &= 2^{\kappa_p}(m_i + r_i) + s_{i,0} \ , & e_{i,1} &= \rho_i \ , \\ z_{i,0} &= g_0^{1/e_{i,0}} \bmod N_0 \ , & z_{i,1} &= h^{\frac{1}{\rho_i}\prod_{l=1}^t \rho_l} \bmod N_1 \ . \end{aligned}$$

The machine A_b continues the simulation until A outputs a message m and a candidate signature (r, z_0, s_0, z_1, s_1) and this is also output by A_b.

Claim 3. The distribution of the output and queries of A in Experiment D.2 is statistically close to the distribution of its output and queries in the simulation of A_0, and A_1, respectively.

Proof. To see that the distribution of the output and queries of A in the simulation of A_1 is statistically close to the distribution of its output and queries in Experiment D.2 it suffices to note that in the simulation of A_1 each prime ρ_i divides $|SQ_N|$ with negligible probability, and when this is not the case the public key (N_0, g_0, N_1, g_1) and all answers from the $\mathsf{Sig}_{(p_0,q_0,g_0,p_1,q_1,g_1)}(\cdot)$-oracle are identically distributed as in the experiment.

The argument for the distribution of A_0 is similar except that one must also note that for every $m_i \in [0, 2^{\kappa_m} - 1]$ the distribution of $r_i = \tau_i - m_i$ with τ_i randomly chosen in $[2^{2\kappa_r+\kappa_m}, 2^{2\kappa_r+\kappa_m} + 2^{\kappa_r+\kappa_m} - 1]$ is statistically close to the uniform distribution on $[2^{2\kappa_r+\kappa_m}, 2^{2\kappa_r+\kappa_m} + 2^{\kappa_r+\kappa_m} - 1]$. □

We denote the variables occurring in an execution of Experiment D.2 as follows. The public keys are denoted by (N_0, g_0) and (N_1, g_1). The ith query to the signature oracle is denoted by m_i and the ith answer is denoted by $\sigma_i = (r_i, z_{i,0}, s_{i,0}, z_{i,1}, s_{i,1})$. For convenience we also define $e_{i,0} = 2^{\kappa_p}(r_i + m_i) + s_{i,0}$ and $e_{i,1} = 2^{\kappa_p} r_i + s_{i,1}$. The output of A is denoted by (m, σ) with $\sigma = (r, z_0, s_0, z_1, s_1)$, and we define $e_0 = 2^{\kappa_p}(m + r) + s_i$ and $e_1 = 2^{\kappa_p} r_i + s_1$. The corresponding variables in the simulation carried out by A_b are denoted by $(N_0^{(b)}, g_0^{(b)})$, $(N_1^{(b)}, g_1^{(b)})$, $m_i^{(b)}$, $\sigma_i^{(b)} = (r_i^{(b)}, z_{i,0}^{(b)}, s_{i,0}^{(b)}, z_{i,1}^{(b)}, s_{i,1}^{(b)})$, $e_{i,0}^{(b)}$, $e_{i,1}^{(b)}$, $(m^{(b)}, \sigma^{(b)})$, $\sigma^{(b)} = (r^{(b)}, z_0^{(b)}, s_0^{(b)}, z_1^{(b)}, s_1^{(b)})$, $e_0^{(b)}$, $e_1^{(b)}$. Let E be the event that $\mathsf{Vf}_{(N_0,g_0,N_1,g_1)}(m, \sigma) = 1$ and $m \neq m_i$ for $i = 1, \ldots, t$. In other words, if the event occurs the experiment outputs 1 and otherwise 0. The event $E^{(b)}$ is defined correspondingly for the simulations.

Claim 4. The probability $\Pr[E \wedge (\forall i : e_d \neq e_{i,d})]$ is negligible for $d \in \{0, 1\}$ under the strong RSA-assumption.

Proof. We only prove the case $d = 0$, since the other case is similar. First note, that for any valid signature both e_0 and e_1 are positive. If the claim is false we have from Claim 3 that

$$\Pr[E^{(0)} \wedge (\forall i : e_0^{(0)} \neq e_{i,0}^{(0)})]$$

is non-negligible. It may happen that the adversary A is not given all primes ρ_i as part of answers to the signature oracle, but each prime takes on any specific value with negligible probability. Thus, we conclude that

$$\Pr[E^{(0)} \wedge (\forall i : e_0^{(0)} \neq \rho_i)]$$

is non-negligible. Denote by A' the adversary defined as follows. It takes (N, h) as input, computes $(m^{(0)}, \sigma^{(0)}) = A_0(N, h)$, and computes integers a and a' such that $c = a \prod_{i=1}^{t} \rho_i + a' e_0^{(0)}$, where $c = \gcd(\prod_{i=1}^{t} \rho_i, e_0^{(0)})$. Then it outputs $(b, \eta) = ((z_0^{(0)})^a h^{a'}, e_0^{(0)}/c)$. Note that if $\mathsf{Vf}_{(N_0^{(0)}, g_0^{(0)}, N_1^{(0)}, g_1^{(0)})}(m^{(0)}, \sigma^{(0)}) = 1$ and $e_0^{(0)} \nmid \prod_{i=1}^{t} \rho_i$, then $e_0^{(0)}/c \neq 1$, and $(z_0^{(0)})^{e_0^{(0)}} = g_0 = h^{\prod_{i=1}^{t} \rho_i}$ which implies that

$$b^{\eta} = ((z_0^{(0)})^a h^{a'})^{e_0^{(0)}/c} = ((z_0^{(0)})^{a e_0^{(0)}} h^{a' e_0^{(0)}})^{1/c} = (h^{a \prod_{i=1}^{t} \rho_i} h^{a' e_0^{(0)}})^{1/c} = h \ .$$

Thus, the strong RSA-assumption is broken.

We now show that $e_0^{(0)} \nmid \prod_{i=1}^{t} \rho_i$. If $\mathsf{Vf}_{(N_0^{(0)}, g_0^{(0)}, N_1^{(0)}, g_1^{(0)})}(m^{(0)}, \sigma^{(0)}) = 1$, then $r^{(0)} \in [1, 2^{2\kappa_r + \kappa_m + 1} - 1]$, which implies that $|e_0^{(0)}| \leq 2^{\kappa_p + 2\kappa_r + \kappa_m}$. By construction $\rho_i = e_{i,0}^{(0)} > 2^{\kappa_p + 2\kappa_r + \kappa_m + 1}$, so $e_0^{(0)} < \rho_i \rho_j$ for $i, j \in \{1, \ldots, t(\kappa)\}$. On the other hand $r^{(0)} > 0$, which implies that $e_0^{(0)} > 1$. Thus, $e_0^{(0)}$ is bigger than one, smaller than any product of two or more of the primes ρ_i, and it is distinct from each prime ρ_i, and the claim follows. □

Claim 5. The probability $\Pr[E \wedge (\exists i : z_d \neq z_{i,d} \wedge e_d = e_{i,d})]$ is negligible for $d \in \{0, 1\}$.

Proof. Suppose the claim is false. Note that with overwhelming probability the prime $e_{i,d}$ does not divide $|SQ_{N_d}|$. Thus, the probability

$$\Pr[E \wedge (\exists i : z_d \neq z_{i,d} \wedge e_d = e_{i,d}) \wedge e_d \nmid |SQ_{N_d}|]$$

is non-negligible. Consider an output such that $\mathsf{Vf}_{(N_0, g_0, N_1, g_1)}(m, \sigma) = 1$, $z_d \neq z_{i,d}$, $e_d = e_{i,d}$, and $e_d \nmid |SQ_{N_d}|$. By definition $z_{i,d} \in SQ_{N_d}$ and $e_d = e_{i,d}$ is odd. Thus, if $z_d^{e_d} = z_{i,d}^{e_d}$, then $z_d \in SQ_{N_d}$. Since e_d is invertible modulo $|SQ_{N_d}|$ we conclude that $z_d = z_{i,d}$, which is a contradiction, and the claim holds. □

Claim 6. The probability $\Pr[E \wedge (\exists i, j : i \neq j \wedge (e_0, e_1) = (e_{i,0}, e_{j,1}))]$ is negligible.

Proof. Consider the probability $\delta(i, j, m')$ that the event E occurs, $(e_0, e_1) = (e_{i,0}, e_{j,1})$, and $m = m'$, for fixed $i, j \in \{1, \ldots, t(\kappa)\}$ and $m' \in [0, 2^{\kappa_m} - 1]$. Recall that $e_{i,0}$ and $e_{j,1}$ are on the form $e_{i,0} = 2^{\kappa_p}(r_i + m) + s_{i,0}$ and $e_{j,1} = 2^{\kappa_p} r_j + s_{j,1}$ respectively for some $s_{i,0}, s_{j,1} \in [0, 2^{\kappa_p} - 1]$ and positive r_i and r_j. Thus, r_i is uniquely defined by $e_{i,0}$ and m, and r_j is uniquely defined by $e_{j,1}$. The event E can only occur if $r_i = r_j$, which happen with probability $\delta(i, j, m') < 2^{-(\kappa_r + \kappa_m)}$. There are at most $t^2 2^{\kappa_m}$ triples (i, j, m'), so by the union bound the probability in the claim is bounded by $t^2 2^{-\kappa_r}$, which is negligible. □

Claim 7. There exists no $m, m' \in [0, 2^{\kappa_m} - 1]$, $s_0, s_1, s'_0, s'_1 \in [0, 2^{\kappa_p} - 1]$, and integers r and r' such that $m \neq m'$,

$$\begin{aligned} 2^{\kappa_p}(r + m) + s_0 = \quad & e_0 \quad = 2^{\kappa_p}(r' + m') + s'_0 \ , \text{ and} \\ 2^{\kappa_p} r + s_1 = \quad & e_1 \quad = 2^{\kappa_p} r' + s'_1 \ . \end{aligned}$$

Proof. It follows from the second equation that $r' = r$, since $s_1, s'_1 \in [0, 2^{\kappa_p} - 1]$. The first equation now implies that $m = m'$, since $s_0, s'_0 \in [0, 2^{\kappa_p} - 1]$ and $2^{\kappa_p}(r + m) - 2^{\kappa_p}(r' + m') = 2^{\kappa_p}(m - m')$. □

We now summarize what we have proved above. Claim 4, Claim 5, and the union bound show that the event

$$E \wedge [(\forall i : e_d \neq e_{i,d}) \vee (\exists i : z_d \neq z_{i,d} \wedge e_d = e_{i,d})]$$

occurs with negligible probability for $d \in \{0, 1\}$. Thus, if the adversary A outputs a valid signature of a new message with non-negligible probability, then

$$\Pr[E \wedge (\exists i, j : (z_0, z_1) = (z_{i,0}, z_{j,1}) \wedge (e_0, e_1) = (e_{i,0}, e_{j,1}))]$$

is non-negligible. Claim 6 and the union bound then shows that

$$\Pr[E \wedge (\exists i : (z_0, z_1) = (z_{i,0}, z_{i,1}) \wedge (e_0, e_1) = (e_{i,0}, e_{i,1}))]$$

is non-negligible. By definition of the event E this implies that

$$\Pr[E \wedge (\exists i : m \neq m_i \wedge (e_0, e_1) = (e_{i,0}, e_{i,1}))]$$

is non-negligible, but Claim 7 shows that this is impossible. Thus, we have reached a contradiction and the scheme is CMA-secure.

Collision-freeness follows immediately from Claim 7. □

5.2 Proofs of Knowledge of Equality Relations

There are various protocols in the literature [14, 2, 11, 4] for proving equality of integer exponents over groups of unknown order based on variations of Fujisaki-Okamoto commitments, under the strong RSA-assumption. These protocols are strictly speaking not proofs of knowledge, since extraction may fail with negligible probability over the choice of commitment parameters, but they can be used as proofs of knowledge provided that there are trusted commitment parameters present during their execution. Furthermore, they are usually expressed as honest verifier zero-knowledge protocols. A useful feature of these protocols is that they bound the bit-size of the exponents.

A Fujisaki-Okamoto commitment scheme [14] is generated by choosing two safe $\kappa/2$-bit primes $p = 2p' + 1$ and $q = 2q' + 1$, defining $N = pq$, and choosing $g, h \in SQ_N$ randomly. A commitment of an integer m is formed as $g^r h^m \bmod N$ for a random $r \in [0, N2^{\kappa_r}]$. Define $\mathcal{R}'_{srsa}$ to be the relation consisting of pairs $((N, g, h), (b, \eta_0, \eta_1, \eta_2))$ such that $b^{\eta_0} = g^{\eta_1} h^{\eta_2}$ and η_0 does not divide both η_1 and η_2, where $(\eta_1, \eta_2) \neq (0, 0)$. Define F'_{srsa} as the algorithm that on input (N, g, h) outputs (N, g, h) if h is in the subgroup generated by g modulo N and (N, g, g) otherwise.

Denote by $\mathcal{R}_{eq}$ the set of pairs

$$\Big[\big(k,\ (N, g, h),\ (n_1, \ldots, n_k),\ (((\kappa_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k},\ (((g_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k},\ ((u_{i,j})_{j=1}^{t_i})_{i=1}^{k}\big)\ ,\ (((x_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k}\Big]$$

where $g, h \in \mathbb{Z}_N^*$, $n_1, \ldots, n_k \in \mathbb{Z}$, for all j, l it holds that $g_{i,j,l}, u_{i,j} \in \mathbb{Z}_{n_i}^*$, $x_{i,j,l} \in [-2^{\kappa_{i,j,l} + \kappa_r} + 1, 2^{\kappa_{i,j,l} + \kappa_r} - 1]$, and for all i, j it holds that $u_{i,j} = \prod_{l=1}^{l_j} g_{i,j,l}^{x_{i,j,l}} \bmod n_i$.

Proposition 5.4. *There exists a proof of knowledge for the relation $\mathcal{R}_{eq} \vee \mathcal{R}'_{srsa}$ which is $(T_\emptyset \| T_\emptyset, F'_{srsa} \| F_{id})$-zero-knowledge for inputs with $x_{i,j,l} \in [-2^{\kappa_{i,j,l}}+1, 2^{\kappa_{i,j,l}}-1]$ such that the number of exponentiations computed by the prover and verifier is constant in the number of exponentiations needed to compute all $u_{i,j}$.*

The protocol used is based on known techniques, but for completeness we include it in Section B. Our efficiency analysis is based on this protocol.

5.3 Verifiable Generation of Hiding Commitment Scheme

The problem with the proofs of equal integer exponents in a two party setting is that it is difficult to generate the Fujisaki-Okamoto commitment parameters efficiently. Recall that the commitment parameters consist of a random RSA-modulus $N = pq$, where p and q are safe primes, a random $g \in SQ_N$, and $h = g^x$ for a random $x \in [0, N2^{\kappa_r}]$. A commitment C of $m \in \mathbb{Z}$ is formed as $C = g^r h^m$ for an $r \in [0, N2^{\kappa_r}]$. The problem is that if (N, g, h) are generated by the prover, then the commitments are not binding. On the other hand, if they are generated by the verifier, then h may not be of the form g^x, and then the commitments are not hiding. As far as we know there is no truly efficient solution to this problem.

We now sketch our solution to this problem. The prover generates a pair (N_r, g_r), where N_r is an RSA-modulus of two safe primes and $g_r \in SQ_{N_r}$. This is done only once and is part of the public key of the prover in our application. The verifier then generates (N, g, h) as above, except that it defines $h = g^x$ for a random $x \in [0, NN_r2^{\kappa_r}]$. It then computes a "commitment" $h_r = g_r^x$ of x and executes a Schnorr-like zero-knowledge "proof of knowledge" that the same integer x was used for both h_r and h. Extraction of x is possible with high probability provided that (N_r, g_r) are chosen correctly. This ensures that the parameters (N, g, h) can be used safely by a prover. On the other hand, provided $|SQ_{N_r}|$ and $|SQ_N|$ are relatively prime, h_r is essentially independently generated from h. This implies that the parameters (N, g, h) can be used safely by the verifier, since essentially no knowledge of x is leaked. To ensure that $|SQ_{N_r}|$ and $|SQ_N|$ are relatively prime with overwhelming probability we assume that N is generated independently from N_r. In practice this is very reasonable.

In Section 5.4 we show that the parameters (N, g, h) output by the protocol below can be used to execute the proof of equal exponents that assumes trusted commitment parameters. We call the two players in the proof the generator G and the receiver R to distinguish them from their roles in a larger protocol. Denote by $\pi_{pl} = (P_{pl}, V_{pl})$ the zero-knowledge proof of knowledge of a logarithm for prime order groups described by Cramer et al. [7].

Protocol 5.5 (Secure Generation of Integer Commitment Scheme).
Common Input: A κ-bit integer N_r and $g_r \in \mathbb{Z}^*_{N_r}$ to both parties.

1. The receiver chooses a safe κ-bit prime $P = 2Q + 1$, and $H \in G_Q$ randomly, where G_Q is the unique subgroup of order Q. Then it chooses $z \in \mathbb{Z}_Q$ randomly, computes $K = H^z \bmod P$, hands (P, H, K) to the generator, and executes π_{pl} as the prover on common input (P, H, K) and private input z. If the verifier rejects, then the generator hands $\perp$ to the receiver and halts.

2. The generator verifies that P is a safe prime and that $H, K \in G_Q$. Then it chooses an RSA-modulus N, $g \in SQ_N$, and $x \in [0, 2^{2\kappa+\kappa_r} - 1]$ randomly, and

computes $h = g^x \bmod N$ and $h_r = g_r^x \bmod N_r$. Then it chooses $c_g \in [0, 2^{\kappa_c} - 1]$, $r_g \in \mathbb{Z}_q$, and $r \in [0, 2^{2\kappa+2\kappa_r+\kappa_c} - 1]$ randomly, defines $w = H^{r_g} K^{c_g}$, $\alpha = g^r \bmod N$ and $\alpha_r = g_r^r \bmod N_r$, and hands $(h, h_r, w, \alpha, \alpha_r)$ to the receiver.

3. The receiver chooses $c_r \in [0, 2^{\kappa_c} - 1]$ randomly and hands c_r to the generator.

4. The generator computes $c = c_g \oplus c_r$ and $d = cx + r \bmod 2^{2\kappa+2\kappa_r+\kappa_c}$, hands (d, c_g, r_g) to the receiver, and outputs (N, g, h).

5. The receiver outputs (N, g, h) if $H^{r_g} K^{c_g} = w$, $h^c \alpha = g^d \bmod N$ and $h_r^c \alpha_r = g_r^d \bmod N_r$. Otherwise it outputs $\perp$.

We denote the above protocol by $\pi_{tp} = (G, R)$, where G is the generator and R the receiver.

Remark 5.6. Although the receiver can use the same prime P in every protocol instance, the generator must check that P is of the expected form to be confident that it can run the protocol π_{pl}, which is only sound if G_Q has prime order.

Checking for primality is expensive, i.e., it requires $O(\kappa)$ exponentiations. If one assumes that it is infeasible to find a specific safe prime P such that the discrete logarithm problem is feasible in G_Q, then any party can choose a prime P that is used in every protocol instance. Then each party performs the primality test only once. This is a natural assumption in practice, where one can use a prime from a cryptographic standard.

Proposition 5.7. *For every pair (N_r, g_r) with $N_r \in \mathbb{N}$ and $g_r \in \mathbb{Z}_{N_r}^*$ the probability $\Pr[\langle R, G\rangle(N_r, g_r) \neq \perp]$ is overwhelming.*

Proof. This follows from the completeness of the protocol π_{pl} and the fact that the receiver outputs (N, g, h) provided that there is no modular reduction in the computation of d. □

Proposition 5.8. *Let (N, g, h) be randomly distributed Fujisaki-Okamoto parameters. Define $[R^*, G](N_r, g_r)$ to be a pair consisting of the output of R^* and G.*

Then for every receiver $R^* \in \mathrm{PPT}$ and constant $\gamma > 0$ there exists a simulator $M \in \mathrm{PPT}$ such that for every pair (N_r, g_r) with $N_r \in \mathbb{N}$ and $g_r \in \mathbb{Z}_{N_r}^*$ the statistical distance between the distributions of $[R^*, G](N_r, g_r)$ and $M(N_r, g_r, N, g, h)$ is at most $\kappa^{-\gamma}$ and $M(N_r, g_r, N, g, h)$ is always on the form $(\cdot, out_G)$ with $out_G \in \{(N, g, h), \perp\}$.

Informally, this simply means that we can simulate the protocol in such a way that a particular set of parameters are used.

Remark 5.9. Since the generator does not have any secret input, it is not meaningful to say that the protocol is zero-knowledge. However, one may view the proposition as saying that the protocol leaks no knowledge to the receiver about the exponent x that is chosen by the generator within the protocol. In this sense the protocol is zero-knowledge.

Proof (Proposition 5.8). The simulator first simulates Step 1 of the protocol. If the generator halts with output $\perp$, it simply outputs $\perp$ and halts. Otherwise it invokes the extractor of the protocol π_{pl} to get z such that $K = H^z \bmod P$. Then the simulator simulates Step 2 of the protocol, with the following modifications. It uses the input (N, g, h) of instead of generating these parameters. It chooses $c \in [0, 2^{\kappa_c} - 1]$ and $d \in$

$[0, 2^{2\kappa+2\kappa_r+\kappa_c} - 1]$ randomly and defines $\alpha = g^d/h^c \bmod N$ and $\alpha_r = g_r^d/h_r^c \bmod N_r$. Finally, it simulates Step 4 except that it defines $c'_g = c_r \oplus c$ and $r'_g = r_g + (c_g - c'_g)z$.

First we argue that the distribution of the resulting view is statistically close to the distribution of the real view of the receiver. Consider the distribution of h and h_r in the real protocol. Denote by t the order of g_r modulo N_r. Note that when the receiver is malicious we have little control over t. However, when the generator is honest N is chosen as a product of two random $\kappa/2$-bit safe primes $p = 2p'+1$ and $q = 2q'+1$, and this is done independently of N_r. The probability that either p' or q' takes any specific prime as value is of course negligible. This implies that the probability that either p' or q' divides the order t of g_r modulo N_r is negligible. Thus, with overwhelming probability t and $p'q'$ are relatively prime.

This means that the group $X = \langle g_r \rangle \times \langle g \rangle$, where multiplication is taken elementwise modulo N_r and N respectively, is generated as a group by the pair (g_r, g). Furthermore, the order of the group X is $tp'q'$, which is an integer with less than 2κ bits. This implies that the distribution of $(g_r, g)^x = (g_r^x \bmod N_r, g^x \bmod N)$ for a randomly chosen $x \in [0, 2^{2\kappa+\kappa_r} - 1]$ is statistically close to the distribution of a pair (h, h_r), where $h \in \langle g \rangle$ is randomly chosen and $h_r = g_r^x$ with x distributed as before.

It follows from a standard argument that for every fixed h and h_r defined as in the protocol the distribution of the remainder of the simulated view is statistically close to view of the real execution.

Consider for any execution the probability $p(P, H, K)$ that the output of the receiver is not $\perp$, conditioned on a fixed choice of (P, H, K). By definition the extractor guaranteed by the proof of knowledge property of π_{pl} runs in expected time $T(\kappa)/(p(P, H, K) - \epsilon(\kappa))$ for some polynomial $T(\kappa)$ and negligible function $\epsilon(\kappa)$. We conclude that by truncating the simulation with an appropriate polynomial bound, and applying the union bound, the proposition follows. □

Denote by T_{srsa} the algorithm that on input 1^κ outputs (N_r, g_r), such that N_r is a product of two random safe $\kappa/2$-bit safe primes and g_r is randomly chosen in SQ_{N_r}.

Proposition 5.10. *Suppose that $(N_r, g_r) = T_{srsa}(1^\kappa)$. Then the probability that the receiver outputs (N, g, h), where h is not in the subgroup of $\mathbb{Z}_N^*$ generated by g, is negligible under the strong RSA-assumption and the discrete logarithm assumption.*

Proof (Proposition 5.10). To start with we show that a malicious generator can not open its Pedersen commitment in several ways.

Claim 8. Denote the list (P, H, K, h, h_r, w) by T, and denote by $S_{coll}(c)$ the set of T such that there exists $(c'_g, r'_g) \neq (c''_g, r''_g)$ such that $\Pr[(c_g, r_g) = (c'_g, r'_g) \mid T \in S_{coll}(c)], \Pr[(c_g, r_g) = (c''_g, r''_g) \mid T \in S_{coll}(c)] \geq \kappa^{-c}$. Then for every constant c and every generator $G^* \in \mathrm{PPT}$ the probability $\Pr[T \in S_{coll}(c)]$ is negligible.

Proof. This follows from the binding property of Pedersen commitments. Suppose the claim is false. Then there exists a constant c and a generator G^* such that $\Pr[T \in S_{coll}(c)]$ is non-negligible. Denote by A the algorithm that accepts (P, H, K) as input and simulates the receiver except that instead of executing the prover of the protocol π_{pl} it invokes the perfect zero-knowledge simulator. It continues the simulation until G^* outputs (c_g, r_g). Then it rewinds the simulation to the challenge step, Step 3, and then continues the simulation again until G^* outputs a new possibly different pair (c'_g, r'_g). If $(c_g, r_g) \neq (c'_g, r'_g)$ and $H^{r_g} K^{c_g} = w = H^{r'_g} K^{c'_g}$, then it outputs

$(r_g' - r_g)/(c_g - c_g') \bmod Q$, and otherwise $\perp$. It follows that A outputs $\log_H K$ with non-negligible probability. This contradicts the discrete logarithm assumption and the claim must be true. $\square$

We say that an output of R, (N, g, h), is bad if h is not in the subgroup generated by g, and denote this event by B. Suppose now that the proposition is false, i.e., there exists a generator G^*, a constant c, and an infinite index set $\mathcal{N}$ such that for $\kappa \in \mathcal{N}$, the probability that the honest receiver R outputs a bad (N, g, h) is at least κ^{-c} for $\kappa \in \mathcal{N}$.

Denote by A the algorithm that takes (N_r, g_r) as input and simulates an interaction between the honest receiver and the generator G^*. Then it rewinds back to Step 3 and completes the simulation once again. If either simulation is rejecting then it outputs $\perp$. Otherwise denote by $(h, h_r, w, \alpha, \alpha_r, c_r, d, c_g, r_g)$ and $(h, h_r, w, \alpha, \alpha_r, c_r', d', c_g', r_g')$ the relevant elements of the two simulations. This gives us $h^{c-c'} = g^{d-d'} \bmod N$ and $h_r^{c-c'} = g_r^{d-d'} \bmod N_r$. If $(c_g, r_g) \neq (c_g', r_g')$ or if $c - c'$ divides $d - d'$ or if $c_r = c_r'$, then it outputs $\perp$. Otherwise it computes a and b such that $e = a(c - c') + b(d - d')$ and $e = \gcd(c - c', d - d')$, and outputs $(b, \eta) = (h_r^b g_r^a, (c - c')/e)$. Note that $b^\eta = h_r^{b(c-c')/e} g_r^{a(c-c')/e} = g_r \bmod N_r$ with $\eta \neq \pm 1$.

Denote by E the event that the probability that R outputs a bad (N, g, h) is at least $\frac{1}{2}\kappa^{-c}$ conditioned on the simulation up to and including Step 2. Then it follows that $\kappa^{-c} \leq \Pr[B \wedge E] + \Pr[B \wedge \bar{E}]$, which gives $\frac{1}{2}\kappa^{-c} \leq \Pr[B \wedge E]$. This implies that $\Pr[E], \Pr[B \mid E] \geq \frac{1}{2}\kappa^{-c}$. Note now that if (N, g, h) is bad, then for any two lists $(h, h_r, w, \alpha, \alpha_r, c_r, d, c_g, r_g)$ and $(h, h_r, w, \alpha, \alpha_r, c_r', d', c_g', r_g')$ we must have $c = c'$ or $(c - c')$ does not divide $(d - d')$. The first event may happen in two ways: either $(c_g, r_g) \neq (c_g', r_g')$, or $(c_g, r_g) = (c_g', r_g')$ and $c_r = c_r'$. The probability of the first event is negligible by the claim above, and the probability of the second event is negligible as well since c_r and c_r' are chosen randomly from an exponentially large space. Thus, using the union bound the adversary A outputs a non-trivial root of g_r with probability at least $\Pr[E](\Pr[B \mid E] - \epsilon(\kappa))^2$ for some negligible function $\epsilon(\kappa)$. This contradicts the strong RSA-assumption and the proposition follows. $\square$

Remark 5.11. Even with the modification of Remark 5.6 the protocol requires a non-constant number of exponentiations, since the generator may have to generate a new RSA-modulus to ensure that its modulus is generated independently of N_r. If the reader find this annoying, please note that if the generator chooses an RSA-modulus N with at least $2\kappa + 2$ bits it can reuse the *same* modulus in any proof, since the orders of g and g_r are coprime. However, the size of the random exponents in the protocol above, and in all protocols that use the modulus must then be doubled, and this gives a far less efficient protocol in practice. Thus, we choose to present the protocol above.

5.4 Special Composition

Let $\pi = (P, V)$ be a protocol that requires Fujisaki-Okamoto commitment parameters as special input. Then we define the special composition $\pi_c = (P_c, V_c)$ of π_{tp} and π to be the following protocol. The prover takes common input (N_r, g_r) and x, private input w, and then executes the receiver R on input (N_r, g_r). If the result is of the form (N, g, h) it then executes P on special input (N, g, h), common input x and private input w. Otherwise the prover outputs $\perp$. The verifier takes common input (N_r, g_r)

and x, and then executes the generator G on input (N_r, g_r). Then it executes V on common input x, using the output of G, (N, g, h), as special input.

Lemma 5.12. *Let $\pi = (P, V)$ be a $(T_\emptyset \| T, F'_{srsa} \| F)$-zero-knowledge proof of knowledge for a relation $\mathcal{R}'_{srsa} \vee \mathcal{R}$. Then the special composition π_c of π_{tp} and π is a $(T_{srsa} \| T, F_\emptyset \| F)$-zero-knowledge proof of knowledge for the relation $\mathcal{R}$ under the strong RSA-assumption and the discrete logarithm assumption.*

Proof. We prove the zero-knowledge property first. Consider some verifier V_c^* and constant $\gamma > 0$. We define G^* to be the generator that simulates V_c^* except that it outputs its entire state when the receiver R of π_{tp} is able to form its output. This is a well defined statement, since it is defined only in terms of the number of messages read and written by V_c^*. We define V^* to be the verifier that takes the state of G^* as auxiliary input and then continues the simulation of V_c^*. Let M be the simulator corresponding to the verifier V^* and the constant γ that is guaranteed to exist by the $(T_\emptyset \| T, F'_{srsa} \| F)$-zero-knowledge property of π. We define M_c to be the simulator that executes the honest receiver R and then executes M of the output from R (and the common input of course). Suppose now that there exists an instance chooser I_c and a distinguisher D such that

$$|\Pr[\mathsf{Exp}_{\pi_c,\mathcal{R},I_c,V_c^*,D}^{(T_{srsa}\|T,F_\emptyset\|F)-\mathsf{zk}-0}(\kappa) = 1] - \Pr[\mathsf{Exp}_{\pi_c,\mathcal{R},I_c,V_c^*,D}^{(T_{srsa}\|T,F_\emptyset\|F)-\mathsf{zk}-1}(\kappa) = 1]| \geq \frac{1}{2}\kappa^{-\gamma} \enspace .$$

Denote by I the instance chooser that accepts an input t, computes $(N_r, g_r) = T_{srsa}(1^\kappa)$, and $(i, z) = I_c((N_r, g_r), t)$. Then it simulates an interaction between G^* and R on common input (N_r, g_r) until R gives an output. We may assume that the output of R is (N, g, h) and not $\perp$, since given that the output is $\perp$, the simulation of M_c is perfect. Finally, I outputs $(((N, g, h), t), z)$.

From Proposition 5.10 follows that the probability that h is not in the group generated by g is negligible. Thus, without loss we may assume that this is the case. Thus, the statistical distance between the distributions of $\mathsf{Exp}_{\pi,\mathcal{R},I,V^*,D}^{(T_\emptyset\|T,F'_{srsa}\|F)-\mathsf{zk}-b}(\kappa)$ and $\mathsf{Exp}_{\pi_c,\mathcal{R},I_c,V_c^*,D}^{(T_{srsa}\|T,F_\emptyset\|F)-\mathsf{zk}-b}(\kappa)$ is at most $\frac{1}{2}\kappa^{-\gamma}$. This means that

$$|\Pr[\mathsf{Exp}_{\pi,\mathcal{R},I,V^*,D}^{(T_\emptyset\|T,F'_{srsa}\|F)-\mathsf{zk}-0}(\kappa) = 1] - \Pr[\mathsf{Exp}_{\pi,\mathcal{R},I,V^*,D}^{(T_\emptyset\|T,F'_{srsa}\|F)-\mathsf{zk}-1}(\kappa) = 1]| \geq \kappa^{-\gamma} \enspace ,$$

which contradicts the $(T_\emptyset \| T, F'_{srsa} \| F)$-zero-knowledge of π. We conclude that π_c is $(T_{srsa} \| T, F_\emptyset \| F)$-zero-knowledge.

Next we consider knowledge extraction. Denote by X_c the extractor that proceeds as follows on common input x. It executes the generator G and then invokes the extractor X of the protocol π. If X outputs a witness w such that $\mathcal{R}(x, w) = 1$ then it outputs w. Otherwise it rewinds and executes G and the extractor X again. Clearly, if the extractor X_c halts it outputs a witness. If the extractor does not run in expected polynomial time, then by definition of a knowledge extractor this means that the extractor X with non-negligible probability outputs a witness $(b, \eta_0, \eta_1, \eta_2)$ such that $((N, g, h), (b, \eta_0, \eta_1, \eta_2)) \in \mathcal{R}'_{srsa}$. If this is the case we break the strong RSA-assumption as follows. Denote by A the algorithm that takes (N, g, h) as input, invokes the simulator guaranteed by Proposition 5.8, and then run the extractor X. It follows that A outputs $(b, \eta_0, \eta_1, \eta_2)$ such that $((N, g, h), (b, \eta_0, \eta_1, \eta_2)) \in \mathcal{R}'_{srsa}$ with non-negligible probability. From Lemma C.3 we know that this contradicts the strong RSA-assumption.

Note that for the above argument to go through it is necessary that the simulator receives the RSA-parameters as an input. Intuitively, the reason is that otherwise we have no control over what the protocol that generates these parameters leak about the secret exponent x. $\square$

5.5 The Cramer-Shoup Version of the Paillier Cryptosystem

The Cramer-Shoup version [10, 4] of the Paillier [25] cryptosystem can be described as follows. On input 1^κ the key generator first chooses the description of a collision-free hash function H. Then it chooses two random $\kappa/2$-bit safe primes $p = 2p' + 1$ and $q = 2q' + 1$ and defines $n = pq$. All multiplications below are then performed modulo n^2. It chooses $x, x_0, x_1 \in [0, 2^{\kappa+\kappa_r} - 1]$ and $\lambda \in \mathbb{Z}_{n^2}^*$ randomly, defines $k = \lambda^{2n}$, and computes $y = k^x$, $y_0 = k^{x_0}$, and $y_1 = k^{x_1}$. Finally, it defines $pk = (H, n, \lambda, y, y_0, y_1)$ and $sk = (H, n, x, x_0, x_1)$, and outputs the key pair (pk, sk) (we drop H and n from our notation when convenient). On input a public key pk, a message $m \in \mathbb{Z}_n$, and a label $L \in \{0,1\}^*$ the encryption algorithm chooses $t \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly and computes $u = k^t$ and $v = \eta^{m/2}(y^n)^t$, where $\eta = 1 + n$, and $m/2$ is computed modulo n. Then it computes $e = H(u^{2n}, v^2, L)$ and $w = y_0^t y_1^{te}$, and outputs (u, v, w). We consider this to be one of many encodings of the ciphertext $(u', v', w') = (u^{2n}, v^2, w^{2n})$. Any ciphertexts having the same decoding are considered *identical*. On input a secret key sk and a decoded ciphertext (u', v', w') the decryption algorithm first verifies that $w' = (u')^{x_0 + x_1 H(u', v', L)}$. If so it outputs $v'(u')^{-x}$ and otherwise $\perp$.

Our scheme differs slightly from the scheme in [4]: we use encodings to avoid "benign malleability", and we have abolished the verification that $v'(u')^{-x}$ is of the form η^m for some $m \in \mathbb{Z}_n$. It is not hard to see from the proof in [4] that the following proposition holds.

Proposition 5.13. *The above public key encryption scheme with labels is CCA2-secure under the decision composite residousity assumption.*

6 An Efficient Designated Confirmer Signature Scheme

We describe an efficient instantiation of the generic construction using the twin-moduli signature scheme and a variation of the Cramer-Shoup version of the Paillier cryptosystem. Suppose we use the twin-moduli signature scheme with security parameter $\kappa_{rsa} = \kappa$. To simplify the exposition we define a function

$$\begin{aligned} \mathsf{Pack} &: \mathbb{N} \times [0, 2^{\kappa_p} - 1]^2 \to \mathbb{N} \\ \mathsf{Pack} &: (r, s_0, s_1) \mapsto 2^{2\kappa_p} r + 2^{\kappa_p} s_0 + s_1 \ , \end{aligned}$$

where κ_p is defined as in the twin-moduli signature scheme, and note that the inverse of this function is easy to compute.

6.1 The Δ-CCA2-Secure Cryptosystem

We define a new cryptosystem $(\mathsf{CSKg}, \mathsf{Enc}, \mathsf{Dec})$ with labels as follows. Define $\kappa_{pa} = \kappa_{rsa} + 2\kappa_r + 3$. On input 1^κ the key generator CSKg invokes the original key generator on $1^{\kappa_{pa}}$ to generate a key pair (pk_*, sk_*), computes $(N_r, g_r) = T_{srsa}(1^{\kappa_{rsa}})$ and outputs

$$(pk, sk) = ((N_r, g_r, pk_*), sk_*) \ .$$

The encryption algorithm Enc takes as input pk, a label L interpreted as a twin-moduli public key (N_0, g_0, N_1, g_1), and a twin-moduli signature (r, z_0, s_0, z_1, s_1). It chooses $t_a, t_0, t_1 \in [0, 2^{\kappa_{pa}+\kappa_r} - 1]$ and $r_0, r_1 \in [0, 2^{\kappa_{rsa}+\kappa_r} - 1]$ randomly and computes

$$\begin{aligned}
a &= \mathsf{Pack}(r, s_0, s_1) & & \\
(u_a, v_a) &= (k^{t_a}, \eta^{a/2} y^{nt_a}) & w_a &= y_0^{t_a} y_1^{t_a e} \\
(u_0, v_0) &= (k^{t_0}, \eta^{r_0/2} y^{nt_0}) & w_0 &= y_0^{t_0} y_1^{t_0 e} \\
(u_1, v_1) &= (k^{t_1}, \eta^{r_1/2} y^{nt_1}) & w_1 &= y_0^{t_1} y_1^{t_1 e} \\
c_0' &= g_0^{r_0} z_0 \bmod N_0 & & \\
c_1' &= g_1^{r_1} z_1 \bmod N_1 & & \\
e &= H(L, u_a^{2n}, v_a^2, u_0^{2n}, v_0^2, c_0', u_1^{2n}, v_1^2, c_1') & &
\end{aligned}$$

Finally, it outputs the ciphertext $((u_a, v_a, w_a), (u_0, v_0, w_0), c_0', (u_1, v_1, w_1), c_1')$.

Decoding is done as when computing the hash value above, by squaring and taking $2n$th powers. The decryption algorithm takes a secret key sk, a label L interpreted as a twin-moduli public key (N_0, g_0, N_1, g_1), and a decoded ciphertext as input. Validity is checked by verifying that

$$w_a' = (u_a')^{x_0 + x_1 e} \ , \quad w_0' = (u_0')^{x_0 + x_1 e} \ , \quad \text{and} \quad w_1' = (u_1')^{x_0 + x_1 e} \ ,$$

where e is the hash value above. It is also verified that $u_a, v_a, w_a, u_0, v_0, w_0, u_1, v_1, w_1 \in \mathbb{Z}_{n^2}^*$ and that $c_0' \in \mathbb{Z}_{N_0}^*$ and $c_1' \in \mathbb{Z}_{N_1}^*$. Then the decryption algorithm computes

$$A = v_a'(u_a')^{-x} \ , \quad R_0 = v_0'(u_0')^{-x} \ , \quad \text{and} \quad R_1 = v_1'(u_1')^{-x} \ .$$

If $A = \eta^a$, $R_0 = \eta^{r_0}$ and $R_1 = \eta^{r_1}$, then it outputs

$$(\mathsf{Pack}^{-1}(a), c_0'/g_0^{r_0} \bmod N_0, c_1'/g_1^{r_1} \bmod N_1)$$

and otherwise it outputs $(\perp, A, R_0, R_1, c_0', c_1')$.

A problem with computing over a composite modulus is that if the holder of a public key is malicious there are no guarantees that the multiplicative group is of the expected form. Thus, it may be possible to choose the modulus and compute a ciphertext such that it can be decrypted both as valid and invalid, depending on which exponents x, x_0, x_1 are used, since the public key does *not* commit its owner to a fixed set of exponents. In our setting this means that a verifier can be given a DC-signature and a proof of validity, but when it hands the DC-signature to the confirmer, the confirmer can prove that the signature is invalid. Intuitively, this is clearly not acceptable, and formally it can not hold for a secure DC-signature scheme.

To solve this problem the confirmer uses the El Gamal cryptosystem as an unconditionally committing and computationally hiding commitment scheme, i.e., we redefine the key generator slightly. It defines $\kappa_{ddh} = \kappa_{pa} + 2\kappa_r + 3$, chooses a κ_{ddh}-bit safe prime $P = 2Q + 1$ randomly, and random elements $g, \beta_x, \beta_{x_0}, \beta_{x_1} \in G_Q$, and then chooses $r \in \mathbb{Z}_Q$ randomly and computes

$$(\mu, \nu_x, \nu_{x_0}, \nu_{x_1}) = (g^r, \beta_x^r g^x, \beta_{x_0}^r g^{x_0}, \beta_{x_1}^r g^{x_1}) \ .$$

The key generator CSKg is then redefined to output $(P, g, \beta_x, \beta_{x_0}, \beta_{x_1}, \mu, \nu_x, \nu_{x_0}, \nu_{x_1})$ as part of the public key (we drop P from our notation when convenient).

Corollary 6.1. *Let Δ be defined as in Section 4 and instantiated with the twin-moduli signature scheme $\mathcal{SS}^2$. Then the encryption scheme above is Δ-CCA2-secure under the decision composite residuosity assumption and the decision Diffie-Hellman assumption.*

Proof. Consider first the cryptosystem where $(\mu, \nu_x, \nu_{x_0}, \nu_{x_1})$ is replaced by a random list of elements from G_Q. Then security follows by a standard hybrid argument from Proposition 5.13, since for each part of the ciphertext (u_a, v_a, w_a), (u_0, v_0, w_0), and (u_1, v_1, w_1), we may view the the remainder of the ciphertext as a label. The problem that there may be several labels if this is done and that the label is not always a prefix is easily solved by defining a new hash function that permutes its bits using a fixed appropriate permutation. Then note that if using the original $(\mu, \nu_x, \nu_{x_0}, \nu_{x_1})$ makes the scheme insecure we can break the decision Diffie-Hellman assumption. □

6.2 Splitting the Confirmer Key Generator

The key generator $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c}}$ can be split in the following way. Define $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c},1}$ to output the tuple $(H, n, \lambda, P, g, \beta_x, \beta_{x_0}, \beta_{x_1})$, define $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c},2}$ to output (x, x_0, x_1, r), and let $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c},3}$ on input $(H, n, \lambda, P, g, \beta_x, \beta_{x_0}, \beta_{x_1})$ and (x, x_0, x_1, r) output $(y, y_0, y_1, \mu, \nu_x, \nu_{x_0}, \nu_{x_1})$ as defined above if $2 \in \mathbb{Z}_n^*$, P is a safe prime, $g, \beta_x, \beta_{x_0}, \beta_{x_1} \in G_Q$, and $x, x_0, x_1 \in [-2^{\kappa_{pa}+2\kappa_r} + 1, 2^{\kappa_{pa}+2\kappa_r} - 1]$. Otherwise it sets $y = y_0 = y_1 = 1$. The definition of a well-formed key then follows. The following shows that for any well-formed "public key" there exists a unique "secret key".

Lemma 6.2. *Let $pk_1 = (H, n, \lambda, P, g, \beta_x, \beta_{x_0}, \beta_{x_1})$, where H is any string, n is a κ_{pa}-bit integer, $\lambda \in \mathbb{Z}$, $P = 2Q+1$ is a κ_{ddh}-bit safe prime, and $g, \beta_x, \beta_{x_0}, \beta_{x_1} \in G_Q$. Then there does not exist $r, r' \in \mathbb{Z}$ and $x, x_0, x_1, x', x'_0, x'_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that $(x, x_0, x_1) \neq (x', x'_0, x'_1)$ and $\mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c},3}(pk_1, (x, x_0, x_1, r)) = \mathsf{Kg}^{\mathsf{dc}}_{\mathsf{c},3}(pk_1, (x', x'_0, x'_1, r'))$.*

Proof. This follows from the fact that there exists a unique $r \in \mathbb{Z}_Q$ such that $\mu = g^r$ and for that r there exists at most one set of x, x_0, and x_1 in $[-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that $\beta_x^r g^x = \nu_x$, $\beta_{x_0}^r g^{x_0} = \nu_{x_0}$, and $\beta_{x_1}^r g^{x_1} = \nu_{x_1}$, since Q is a $(\kappa_{pa} + 2\kappa_r + 2)$-bit integer. □

6.3 The CMA-Secure and Collision-Free Signature Scheme

We use the twin-moduli signature scheme with security parameter $\kappa_{rsa} = \kappa$ except that we assume that the public key is augmented with a pair $(N_r, g_r) = T_{srsa}(1^{\kappa_{rsa}})$, that is used in the protocols.

We need a particular form of the (f, f')-embedding assumption. We assume that the (f, f')-embedding assumption holds for some f and f' such that

$$\kappa'_p = f'(2\kappa_r + \kappa_m + 1) \geq \kappa_r + 2 \ , \quad \kappa_m + 2\kappa_r + 2\kappa_p + 4 < \kappa_{pa} - \kappa_r \ , \quad \text{and} \qquad (1)$$

$$\kappa_m + 2\kappa_r + \kappa_p + 2 < (\kappa_{rsa} - 1)/2 \qquad (2)$$

where $\kappa_p = f(2\kappa_r + \kappa_m + 1)$ as defined in the twin-moduli signature scheme. The first requirement ensures that there is sufficient slack between the bit-size of s_0 and s_1 and κ_p to allow a simple Schnorr-like protocol for proving knowledge of s_0 and s_1. The second requirement essentially ensures that the result a of running Pack on (r, s_0, s_1) can be uniquely encoded as an integer in $[0, n]$ with some additional slack that allows a simple Schnorr-like protocol for proving knowledge of a. To fully appreciate the value of our bit-fiddling, the reader may need to study our protocols and their analysis. The third requirement ensures that any integer e_0 or e_1 induced by a well-formed signature (r, z_0, s_0, z_1, s_1) is smaller than the order of any subgroup of SQ_{N_0} and SQ_{N_1}, when N_0 and N_1 are honestly generated. This is important in the construction of the (T_{hs}, F_{hs})-zero-knowledge simulator.

6.4 A Well-Formedness Function

We instantiate the well-formedness function wf as follows. First of all we have already defined a well-formed public key. We define wf to output 0 when $\perp$ is a prefix of its first input. We define $wf((r, z_0, s_0, z_1, s_1), pk, spk) = 1$ if pk is well-formed, $r \in [2^{2\kappa_r+\kappa_m}, 2^{2\kappa_r+\kappa_m} + 2^{\kappa_r+\kappa_m} - 1]$, $s_0, s_1 \in [2^{\kappa_p-1}, 2^{\kappa_p-1} + 2^{\kappa_p-\kappa_p'} - 1]$ are odd, $z_0, g_0 \in \mathbb{Z}_{N_0}^*$, $z_1, g_1 \in \mathbb{Z}_{N_1}^*$, and $z_1^{e_1} = g_1 \bmod N_1$. Note that the well-formedness of pk can be verified by anyone and that for any honestly generated spk and honestly computed signature this is always the case.

6.5 An Informal Description of Some Key Ideas

Our choice of cryptosystem and signature scheme allow efficient protocols that are described in detail in the next section. In this section we merely try to convey some key ideas.

The idea of the twin-moduli signature scheme is loosely speaking that all that is needed to verify a signature can be done "in the exponent". Recall that a verification involves multiplication by constants, adding, checking for interval-membership and oddity, and then checking the roots of the signature.

Let us write $C(m)$ for a Fujisaki-Okamoto commitment of the form $g^l h^m \bmod N$ for some random l, and simply write $\mathsf{Enc}^{\mathsf{pa}}(m)$ for a pair of the form $(k^r, \eta^m y^r)$, i.e., we ignore the encoding and the third component that guarantees CCA2-security.

Then the proof of validity of a signature can be explained as follows. A DC-signature essentially consists of a tuple

$$(E_a, E_{r_0}, c_0', E_{r_1}, c_1') = (\mathsf{Enc}^{\mathsf{pa}}(\mathsf{Pack}(r, s_0, s_1)), \mathsf{Enc}^{\mathsf{pa}}(r_0), g_0^{r_0} z_0, \mathsf{Enc}^{\mathsf{pa}}(r_1), g_1^{r_1} z_1) \ .$$

The prover forms commitments

$$\begin{aligned} C_r' &= C(r - 2^{2\kappa_r+\kappa_m}) \\ C_{s_0}' &= C((s_0 - 2^{\kappa_p-1} - 1)/2) \\ C_{s_1}' &= C((s_1 - 2^{\kappa_p-1} - 1)/2) \end{aligned}$$

and proves knowledge of the committed values. The protocol used to do this also implies that $r - 2^{2\kappa_r+\kappa_m} \in [-2^{2\kappa_r+\kappa_m} + 1, 2^{2\kappa_r+\kappa_m} - 1]$ and $(s_0 - 2^{\kappa_p-1} - 1)/2, (s_1 - 2^{\kappa_p-1} - 1)/2 \in [-2^{\kappa_p-2} + 1, 2^{\kappa_p-2} - 1]$. Then the verifier computes

$$\begin{aligned} C_r &= C_r' C(2^{2\kappa_r+\kappa_m}) \\ C_{s_0} &= (C_{s_0}')^2 C(2^{\kappa_p-1} + 1) \\ C_{s_1} &= (C_{s_1}')^2 C(2^{\kappa_p-1} + 1) \\ C_a &= C_r^{2^{2\kappa_p}} C_{s_0}^{2^{\kappa_p}} C_{s_1} \ , \end{aligned}$$

and the prover proves that the value committed to in C_a equal the value encrypted in E_a. Note that C_{s_0} and C_{s_1} are commitments to odd integers s_0 and s_1 in $[0, 2^{\kappa_p} - 1]$ and C_r is a commitment to an integer $r \in [1, 2^{2\kappa_r + \kappa_m + 1} - 1]$. Thus, part of the verification has already been executed.

To complete the verification the verifier computes commitments of the integers e_0 and e_1 induced by the values r, s_0, and s_1 by forming

$$C_{e_0} = (C_r C(m))^{2^{\kappa_p}} C_{s_0} \quad \text{and} \quad C_{e_1} = C_r^{2^{\kappa_p}} C_{s_1} \ .$$

All that then remains is to prove that if e_0 and e_1 are committed to in C_{e_0} and C_{e_1} and r_0 and r_1 are encrypted in E_{r_0} and E_{r_1}, then

$$(c_0')^{e_0} = g_0^{e_0 r_0} g_0 \bmod N_0 \quad \text{and} \quad (c_1')^{e_1} = g_1^{e_1 r_1} g_1 \bmod N_1 \ .$$

This shows that the encrypted signature is a valid twin-moduli signature of the message m.

The proof of invalidity for well-formed signatures is similar, but more complicated in that at some point the prover must show that $z_0^{e_0}/g_0 \neq 1$ without revealing this value. A standard trick to solve this problem is to randomize the result, i.e., revealing $(z_0^{e_0}/g_0)^l$ for a randomly chosen l. However, in general it may happen that $z_0^{e_0}/g_0$ is contained in some particular subgroup of $\mathbb{Z}_{N_0}^*$ and the simulator clearly does not know if this is the case.

When the public signature key is chosen honestly and the malicious verifier does not know the factorization of N_0, it is infeasible to find any element that generates a non-trivial subgroup of SQ_{N_0}. Thus, in this case the above idea works straightforwardly and there is no problem. In other words we have a (T_{hs}, F_{hs})-zero-knowledge simulator.

For maliciously generated N_0, g_0, z_0, and e_0 the above approach does not work at all, and it seems difficult to come up with an efficient approach that does work. Fortunately, we know that it suffices to have a simulator that is given the values z_0 and e_0 as an additional advice string, and given these it is obviously trivial to generate $(z_0^{e_0}/g_0)^l$ with the right distribution. In other words we have a (T_{cs}, F_{cs})-simulator.

6.6 The Protocols of the Construction

We are now ready to give the details of the protocols of our construction. It may seem that the proof of correct decryption in Camenisch and Shoup [4] can be used directly to construct the proof of correct conversion of a DC-signature, but this is not the case, since (1) we have modified the cryptosystem to allow encryption of twin-moduli signatures, and (2) they assume that the keys of the cryptosystem are *honestly* generated, whereas we do not. However, the reader will find no difficulty in recognizing some of the ideas presented in their paper (as well as in older work on which it is based) also in our proof of correct conversion.

Before we start we introduce some notational conventions. In each of the protocols the prover and verifier first decodes the signature input, but to simplify notation we assume that this has already been done before each protocol is executed. Whenever we for presentational convenience say that π_{eq} is invoked multiple times as a subprotocol, we assume that all these invocations are combined into a single invocation. We also assume that a suitable instance of π_{eq} is used. The protocols are not always strictly speaking zero-knowledge for the relations as stated. The problem is that we have not put any bound on the size of the witnesses, whereas the protocols assume that this is the case. Following common practice in the literature we ignore this, whenever it does not play any essential role. We assume that the verifier in each protocol performs all the self-evident checking that can be done without any secret keys, e.g., that elements that are expected to be invertible indeed are invertible and that elements are of the right bit-size. To avoid cluttering our notation we do not state explicitly the group in which computations take place.This follows from the groups in which the elements are contained. Finally, we abuse the notations $T_1\|T_2$ and $F_1\|F_2$ in that we assume that the concatenation puts the parameters in the right order.

6.6.1 Well-Formed Keys

Consider first the proof of a well-formed key π_{wf}, and recall that we use the splitting of CSKg above. Anybody can in fact verify that a public key is well-formed. However, a well-formed public key may have no corresponding secret key. Thus, the confirmer must prove knowledge of an integer r and integers $x, x_0, x_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that

$$(y, y_0, y_1) = (k^x, k^{x_0}, k^{x_1}) \tag{3}$$

$$(\mu, \nu_x, \nu_{x_0}, \nu_{x_1}) = (g^r, \beta_x^r g^x, \beta_{x_0}^r g^{x_0}, \beta_{x_1}^r g^{x_1}) \ . \tag{4}$$

Note that for honest confirmers we have $x, x_0, x_1 \in [0, 2^{\kappa_{pa}+\kappa_r}-1]$. Lemma 5.12 implies the following.

Corollary 6.3. *The special composition π_{wf} of π_{tp} and π_{eq} gives a $\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}}$-zero-knowledge proof of knowledge for the relation $\mathcal{R}_{wf}$ under the strong RSA-assumption and the discrete logarithm assumption.*

6.6.2 Correct Conversion

To prove that a label *spk* and ciphertext

$$((u_a, v_a, w_a), (u_0, v_0, w_0), c_0', (u_1, v_1, w_1), c_1')$$

are valid and decrypts to $(A, R_0, R_1, c_0', c_1') \in SQ_{n^2}^3 \times \mathbb{Z}_{N_0}^* \times \mathbb{Z}_{N_1}^*$, the confirmer must prove knowledge of an integer r and $x, x_0, x_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that Equations (3)-(4) are satisfied and

$$\begin{aligned} v_a &= A u_a^x & \qquad w_a &= u_a^{x_0}(u_a^e)^{x_1} \\ v_0 &= R_0 u_0^x & w_0 &= u_0^{x_0}(u_0^e)^{x_1} \\ v_1 &= R_1 u_1^x & w_1 &= u_1^{x_0}(u_1^e)^{x_1} \end{aligned}$$

where $e = H(spk, u_a, v_a, u_0, v_0, c_0', u_1, v_1, c_1')$. To prove that the ciphertext is valid and decrypts to $(\mathsf{Pack}^{-1}(a), z_0, z_1)$ the equations on the left are replaced by

$$\begin{aligned} v_a &= \eta^a u_a^x & & \\ v_0 &= \eta^{r_0} u_0^x & \qquad c_0' &= g_0^{r_0} z_0 \\ v_1 &= \eta^{r_1} u_1^x & c_1' &= g_1^{r_1} z_1 \end{aligned}$$

and the prover proves knowledge also of r_0, and r_1. Note that for honest provers we have $x, x_0, x_1 \in [0, 2^{\kappa_{pa}+\kappa_r}-1]$, $a \in [0, 2^{\kappa_m+2\kappa_r+2\kappa_p}-1]$ and $r_0, r_1 \in [0, 2^{\kappa_{rsa}+\kappa_r}-1]$ respectively. Again we have a corollary of Lemma 5.12.

Corollary 6.4. *The special composition π_c^v of π_{tp} and π_{eq} gives a $\mathsf{Kg}_\mathsf{c}^\mathsf{dc}$-zero-knowledge proof of knowledge for the relation $\mathcal{R}_c \cap \{((\sigma, s, pk, spk), sk) : \mathsf{Con}_{sk}^{\mathsf{dc}}(\sigma, spk) \neq \perp\}$ under the strong RSA-assumption and the discrete logarithm assumption.*

To prove that a ciphertext is invalid, the confirmer need only prove knowledge of an integer r and $x_0, x_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that Equations (3)-(4) are satisfied and that

$$w_a \neq u_a^{x_0}(u_a^e)^{x_1} \quad , \quad w_0 \neq u_0^{x_0}(u_0^e)^{x_1} \quad , \quad \text{or} \quad w_1 \neq u_1^{x_0}(u_1^e)^{x_1} \ .$$

Note that for an honest prover $x_0, x_1 \in [0, 2^{\kappa_{pa}+\kappa_r}-1]$. This is done using the protocol for trusted commitment parameters composed with the protocol below.

Protocol 6.5 (Invalid Ciphertext).
COMMON INPUT: An integer N and $g, h \in SQ_N$, a public key of the form $pk = (H, n, \lambda, y, y_0, y_1, P, g, \beta_x, \beta_{x_0}, \beta_{x_1}, \mu, \nu_x, \nu_{x_0}, \nu_{x_1})$, and a label spk and ciphertext of the form $c = ((u_a, v_a, w_a), (u_0, v_0, w_0), c_0', (u_1, v_1, w_1), c_1')$.
PRIVATE INPUT: A private key $sk = (x, x_0, x_1, r)$ corresponding to the public key pk.

1. The prover chooses $t_a, t_0, t_1, l_a, l_0, l_1 \in [0, 2^{\kappa_{rsa}+\kappa_r}-1]$ and $s_a, k_a, s_0, k_0, s_1, k_1 \in [0, 2^{\kappa_{pa}+\kappa_r}-1]$ randomly and computes

$$w' = (u_a^{x_0+x_1e}/w_a)^{k_a}(u_0^{x_0+x_1e}/w_0)^{k_0}(u_1^{x_0+x_1e}/w_1)^{k_1}$$

and

$$B_a = g^{t_a}h^{s_a} \quad B_a' = u_a^{s_a}u_a^{x_0}(u_a^e)^{x_1} \quad C_a = g^{l_a}B_a^{k_a} \quad C_a' = (B_a'/w_a)^{k_a} \tag{5}$$
$$B_0 = g^{t_0}h^{s_0} \quad B_0' = u_0^{s_0}u_0^{x_0}(u_0^e)^{x_1} \quad C_0 = g^{l_0}B_0^{k_0} \quad C_0' = (B_0'/w_0)^{k_0} \tag{6}$$
$$B_1 = g^{t_1}h^{s_1} \quad B_1' = u_1^{s_1}u_1^{x_0}(u_1^e)^{x_1} \quad C_1 = g^{l_1}B_1^{k_1} \quad C_1' = (B_1'/w_1)^{k_1} \tag{7}$$

and hands these values and w' to the verifier. Recall that w_a, w_0, w_1 are invertible, so this is always possible.

2. Then it proves knowledge of an integer r and $x, x_0, x_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$, and $t_a, t_0, t_1, l_a, l_0, l_1 \in [-2^{\kappa_{rsa}+2\kappa_r}+1, 2^{\kappa_{rsa}+2\kappa_r}-1]$ and $s_a, k_a, s_0, k_0, s_1, k_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$ such that Equations (3)-(4) are satisfied and such that the above holds, and it proves knowledge of $t_a', t_0', t_1' \in [-2^{\kappa_{rsa}+\kappa_{pa}+3\kappa_r}+1, 2^{\kappa_{rsa}+\kappa_{pa}+3\kappa_r}-1]$ and $s_a', s_0', s_1' \in [-2^{2\kappa_{pa}+3\kappa_r}+1, 2^{2\kappa_{pa}+3\kappa_r}-1]$ such that

$$C_a = g^{t_a'}h^{s_a'} \quad , \quad C_0 = g^{t_0'}h^{s_0'} \quad , \quad C_1 = g^{t_1'}h^{s_1'} \quad , \quad \text{and} \tag{8}$$
$$C_a'C_0'C_1' = u_a^{s_a'}u_0^{s_0'}u_1^{s_1'}w' \ . \tag{9}$$

In addition to verifying the proofs of knowledge the verifier checks that $w' \neq 1$.

Proposition 6.6. *Protocol 6.5 is a proof of knowledge for the relation $\mathcal{R}_{srsa}' \vee \mathcal{R}_c \cap \{((\sigma, s, pk, spk), sk) : \mathsf{Con}_{sk}^{\mathsf{dc}}(\sigma, spk) = \perp\}$ under the discrete logarithm assumption.*

Proof. We have from Proposition 5.4 that the subprotocol is a proof of knowledge of the exponents or of a witness $(b, \eta_0, \eta_1, \eta_2)$ such that $((N, g, h), (b, \eta_0, \eta_1, \eta_2)) \in \mathcal{R}_{srsa}'$. Thus, we assume that the extractor outputs exponents r and $x, x_0, x_1 \in$

$[-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$, and $t_a, t_0, t_1, l_a, l_0, l_1 \in [-2^{\kappa_{rsa}+2\kappa_r}+1, 2^{\kappa_{rsa}+2\kappa_r}-1]$, $s_a, k_a, s_0, k_0, s_1, k_1 \in [-2^{\kappa_{pa}+2\kappa_r}+1, 2^{\kappa_{pa}+2\kappa_r}-1]$, and $t'_a, t'_0, t'_1 \in [-2^{\kappa_{rsa}+\kappa_{pa}+3\kappa_r}+1, 2^{\kappa_{rsa}+\kappa_{pa}+3\kappa_r}-1]$ and $s'_a, s'_0, s'_1 \in [-2^{2\kappa_{pa}+3\kappa_r}+1, 2^{2\kappa_{pa}+3\kappa_r}-1]$ such that Equations (3)-(4), Equations (5)-(7), and Equations (8)-(9) are satisfied. From this we conclude that

$$\begin{aligned}
C_a &= g^{t_a k_a + l_a} h^{s_a k_a} & C'_a &= u_a^{s_a k_a} (u_a^{x_0} (u_a^{\mathsf{e}})^{x_1} / w_a)^{k_a} \\
C_0 &= g^{t_0 k_0 + l_0} h^{s_0 k_0} & C'_0 &= u_0^{s_0 k_0} (u_0^{x_0} (u_0^{\mathsf{e}})^{x_1} / w_0)^{k_0} \\
C_1 &= g^{t_1 k_1 + l_1} h^{s_1 k_1} & C'_1 &= u_1^{s_1 k_1} (u_1^{x_0} (u_1^{\mathsf{e}})^{x_1} / w_1)^{k_1}
\end{aligned}$$

If $(t'_a, s'_a) \neq (t_a k_a + l_a, s_a k_a)$, $(t'_0, s'_0) \neq (t_0 k_0 + l_0, s_0 k_0)$, or $(t'_1, s'_1) \neq (t_1 k_1 + l_1, s_1 k_1)$, then we have a non-trivial representation $g^{\eta_0} h^{\eta_1} = 1$ with (η_0, η_1) equal to $(t'_a - t_a k_a - l_a, s'_a - s_a k_a)$ or similarly for the other exponents. In other words a witness $(1, 0, \eta_1, \eta_2)$ such that $((N, g, h), (1, 0, \eta_1, \eta_2)) \in \mathcal{R}'_{srsa}$. Assume now that this is not the case. Then we may conclude that

$$\begin{aligned}
u_a^{s'_a} u_0^{s'_0} u_1^{s'_1} w' &= C'_a C'_0 C'_1 \\
&= u_a^{s'_a} u_0^{s'_0} u_1^{s'_1} ((u_a^{x_0} (u_a^{\mathsf{e}})^{x_1} / w_a)^{k_a} ((u_0^{x_0} (u_0^{\mathsf{e}})^{x_1} / w_0)^{k_0} ((u_1^{x_0} (u_1^{\mathsf{e}})^{x_1} / w_1)^{k_1}
\end{aligned}$$

and since $w' \neq 1$ and u_a, u_0, and u_1 are invertible modulo n^2, we conclude that we have a witness that the ciphertext is invalid. □

Define a function F by $F((pk, sk), (\sigma, spk)) = ((\sigma, \perp, pk, spk), sk)$, and denote by $(\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}})'$ the algorithm $\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}}$ except that the pair (N_r, g_r) is eliminated from the output.

Proposition 6.7. *The protocol is $(T_\emptyset \| (\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}})', F'_{srsa} \| F)$-zero-knowledge for the relation $\mathcal{R}'_{srsa} \vee \mathcal{R}_c \cap \{((\sigma, s, pk, spk), sk) : \mathsf{Con}_{sk}^{\mathsf{dc}}(\sigma, spk) = \perp\}$ under the strong RSA-assumption.*

Proof. The simulator chooses $B_a, B_0, B_1 \in SQ_N$, $w' \in SQ_n$, $B'_a \in \langle u_a \rangle$, $B'_0 \in \langle u_0 \rangle$, $B'_1 \in \langle u_1 \rangle$, randomly and computes C_a, C_0, C_1, C'_a, C'_0, and C'_1 honestly. Then it invokes the zero-knowledge simulators of the subprotocol.

The commitments are statistically hiding by the definition of F'_{srsa}, so we may assume without loss that they are distributed as in a real view. The zero-knowledge property then follows from the zero-knowledge property of the subprotocol, provided that w' is correctly distributed. Since the ciphertext is decoded we know that u_a, u_0, u_1, w_a, w_0, and w_1 are all $2n$th residues modulo n. The only way a real view can differ in distribution from the simulated view is if at least one of the elements $u_a^{x_0} (u_a^{\mathsf{e}})^{x_1} / w_a$, $u_0^{x_0} (u_0^{\mathsf{e}})^{x_1} / w_0$, and $u_1^{x_0} (u_1^{\mathsf{e}})^{x_1} / w_1$ have order different from $p'q'$ and 1, where $p = 2p' + 1$, $q = 2q' + 1$ and $n = pq$. However, finding any such element implies that the factors p and q can be recovered (see for example Mao and Lee [22]), and this clearly implies that the factoring assumption is broken, which implies that the strong RSA-assumption is broken.

More precisely, if there exists an instance chooser I such that the elements are anomalous as above with non-negligible probability, then we can define A to be the algorithm that takes n as input, and simulates $(\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}})'$ using this modulus. Then we execute the zero-knowledge experiment and output the anomalous element if the instance chooser outputs such an element. Note that it is essential that the factorization of n is *not* used by the decryption algorithm. □

Corollary 6.8. *The special composition $\pi_c^{\perp}$ of π_{tp} and Protocol 6.5 is a $\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}}$-zero-knowledge proof of knowledge for the relation $\mathcal{R}_c \cap \{((\sigma, s, pk, spk), sk) : \mathsf{Con}_{sk}^{\mathsf{dc}}(\sigma, spk) = \perp\}$ under the strong RSA-assumption and the discrete logarithm assumption.*

Corollary 6.9. *The pair $(\pi_c^{v}, \pi_c^{\perp})$ gives a $\mathsf{Kg}_{\mathsf{c}}^{\mathsf{dc}}$-zero-knowledge proof of knowledge for the relation $\mathcal{R}_c$ under the strong RSA-assumption and the discrete logarithm assumption.*

6.6.3 Validity/Invalidity of DC-signatures

Since the protocols π_v and π_c achieve similar results we define a joint protocol that behaves slightly differently depending on which type of input it is executed on. Below we ignore the case where $m = \perp$, since by definition no signature can be valid for this message.

Protocol 6.10 (Validity/Invalidity of Well-Formed Signature).
COMMON INPUT: An integer N and $g, h \in \mathbb{Z}_n^*$, a public key of the form
$pk = (H, n, \lambda, y, y_0, y_1, P, g, \beta_x, \beta_{x_0}, \beta_{x_1}, \mu, \nu_x, \nu_{x_0}, \nu_{x_1})$, a public signature key of the form $spk = (N_0, g_0, N_1, g_1)$, a message $m \in [0, 2^{\kappa_m} - 1]$, a candidate signature of the form
$((u_a, v_a, w_a), (u_0, v_0, w_0), c_0', (u_1, v_1, w_1), c_1')$, and a bit b.
PRIVATE INPUT: $r \in [2^{2\kappa_r + \kappa_m}, 2^{2\kappa_r + \kappa_m} + 2^{\kappa_r + \kappa_m} - 1]$, odd $s_0, s_1 \in [0, 2^{\kappa_p - \kappa_p'} - 1]$, from which we define

$$\begin{aligned} e &= H(spk, u_a, v_a, u_0, v_0, c_0', u_1, v_1, c_1') \\ a &= \mathsf{Pack}(r, s_0, s_1) \end{aligned}$$

and either (1) $r_0, r_1 \in [0, 2^{\kappa_{rsa} + \kappa_r} - 1]$, $t_a, t_0, t_1 \in [0, 2^{\kappa_{pa} + \kappa_r} - 1]$, and $b = 1$, such that

$$(u_a, v_a, w_a) = (k^{t_a}, \eta^{a} y^{t_a}, (y_0 y_1^{e})^{t_a}) \quad (10)$$
$$(u_0, v_0, w_0) = (k^{t_0}, \eta^{r_0} y^{t_0}, (y_0 y_1^{e})^{t_0}) \quad (11)$$
$$(u_1, v_1, w_1) = (k^{t_1}, \eta^{r_1} y^{t_1}, (y_0 y_1^{e})^{t_1}) \quad (12)$$

or (2) $sk = (x, x_1, x_2, r)$ with $x, x_1, x_2 \in [0, 2^{\kappa_{pa} + \kappa_r} - 1]$ and $r \in \mathbb{Z}_Q$ such that Equations (3)-(4) are satisfied and

$$(v_a, w_a) = (\eta^{a} u_a^{x}, u_a^{x_0} (u_a^{e})^{x_1}) \quad (13)$$
$$(v_0, w_0) = (\eta^{r_0} u_0^{x}, u_0^{x_0} (u_0^{e})^{x_1}) \quad (14)$$
$$(v_1, w_1) = (\eta^{r_1} u_1^{x}, u_1^{x_0} (u_1^{e})^{x_1}) \ . \quad (15)$$

In both cases it also holds that

$$\begin{aligned} (c_0', c_1') &= (g_0^{r_0} z_0, g_1^{r_1} z_1) \\ wf((r, z_0, s_0, z_1, s_1), pk, spk) &= 1 \\ \mathsf{Vf}_{spk}(m, (r, z_0, s_0, z_1, s_1)) &= b \ . \end{aligned}$$

1. The prover forms commitments holding the different parts of the signature (r, z_0, s_0, z_1, s_1). More precisely, the prover defines $r' = r - 2^{2\kappa_r + \kappa_m}$, $s_0' =$

$(s_0 - 2^{\kappa_p - 1} - 1)/2$ and $s_1' = (s_1 - 2^{\kappa_p - 1} - 1)/2$, chooses $l_r, l_{s_0}, l_{s_1}, l_{r_0}, l_{r_1} \in [0, 2^{\kappa_{rsa}+\kappa_r} - 1]$ randomly, computes

$$(C_r', C_{s_0}', C_{s_1}', C_{r_0}, C_{r_1}) = (g^{l_r} h^{r'}, g^{l_{s_0}} h^{s_0'}, g^{l_{s_1}} h^{s_1'}, g^{l_{r_0}} h^{r_0}, g^{l_{r_1}} h^{r_1}) \ , \tag{16}$$

and hands $(C_r', C_{s_0}', C_{s_1}', C_{r_0}, C_{r_1})$ to the verifier.

2. The prover and verifier compute

$$C_{s_0} = (C_{s_0}')^2 h^{2^{\kappa_p - 1}+1} \tag{17}$$
$$C_{s_1} = (C_{s_1}')^2 h^{2^{\kappa_p - 1}+1} \tag{18}$$
$$C_r = C_r' h^{2^{2\kappa_r + \kappa_m}} \tag{19}$$
$$C_a = C_r^{2^{2\kappa_p}} C_{s_0}^{2^{\kappa_p}} C_{s_1} \tag{20}$$
$$C_{e_0} = (C_r h^m)^{2^{\kappa_p}} C_{s_0} \tag{21}$$
$$C_{e_1} = C_r^{2^{\kappa_p}} C_{s_1} \ . \tag{22}$$

The prover computes $a = \mathsf{Pack}(r, s_0, s_1)$, $l_a = 2^{2\kappa_p} l_r + 2^{\kappa_p} l_{s_0} + 2 l_{s_1}$, $e_0 = 2^{\kappa_p}(r + m) + s_0$, $l_{e_0} = 2^{\kappa_p + 1} l_r + 2 l_{s_0}$, $e_1 = 2^{\kappa_p} r + s_1$, and $l_{e_1} = 2^{\kappa_p+1} l_r + 2 l_{s_1}$.

Note that C_a is a commitment of a, C_{e_0} is a commitment of e_0, and C_{e_1} is a commitment of e_1.

3. For all types of inputs the prover proves knowledge of a and l_a such that

$$C_a = g^{l_a} h^a \ . \tag{23}$$

In case (1) the prover proves knowledge of exponents such that Equations (10)-(12) and Equation (16) are satisfied simultaneously, i.e., exponents are equal when appropriate. In case (2) the prover proves knowledge of exponents such that Equations (3)-(4), Equations (13)-(15), and Equation (16) are satisfied simultaneously, i.e., exponents are equal when appropriate. The use of π_{eq} implies that the exponents satisfy $r' \in [-2^{\kappa_m + 2\kappa_r} + 1, 2^{\kappa_m + 2\kappa_r} - 1]$ and $s_0', s_1' \in [-2^{\kappa_p - \kappa_p' + \kappa_r} + 1, 2^{\kappa_p - \kappa_p' + \kappa_r} - 1]$.

These proofs show that the committed values correspond to the encrypted signature.

4. The prover chooses $l_0, l_0', j_0, l_1, l_1', j_1 \in [0, 2^{\kappa_{rsa}+\kappa_r} - 1]$ randomly, computes

$$(C_0, C_0', C_0'') = (g^{l_0} C_{r_0}^{e_0}, g^{l_0'} h^{j_0}, (c_0')^{j_0} (c_0')^{e_0}) \tag{24}$$
$$(C_1, C_1', C_1'') = (g^{l_1} C_{r_1}^{e_1}, g^{l_1'} h^{j_1}, (c_1')^{j_1} (c_1')^{e_1}) \tag{25}$$

and hands $(C_0, C_0', C_0'', C_1, C_1', C_1'')$ to the verifier. Then the prover proves knowledge of exponents such that Equations (24)-(25) are satisfied and

$$(C_{e_1}, C_1, C_1''/g_1) = (g^{l_{e_1}} h^{e_1}, g^{l_1''} h^{j_1'}, (c_1')^{j_1} g_1^{j_1'}) \tag{26}$$

for some l_1'' and j_1'.

Recall that for well-formed signatures this can always be proved.

5. If $b = 1$, i.e., the input was a valid signature, the prover also proves knowledge of exponents such that

$$(C_{e_0}, C_0, C''_0/g_0) \quad = \quad (g^{l_{e_0}} h^{e_0}, g^{l''_0} h^{j'_0}, (c'_0)^{j_0} g_0^{j'_0}) \ , \tag{27}$$

for some l''_0 and j'_0.

6. If $b = 0$ then the prover chooses $i_0, i'_0, i''_0 \in [0, 2^{\kappa_{rsa}+\kappa_r} - 1]$ randomly, computes

$$(B_0, B'_0, B''_0) \quad = \quad (g^{i_0} C_0^{i''_0}, g^{i'_0} (C'_0)^{i''_0}, (C''_0/g_0)^{i''_0}) \tag{28}$$

and proves in addition to knowledge of the above exponents also knowledge of k_0, k'_0, f_0, f'_0 such that

$$(B_0, B'_0, B''_0) \quad = \quad (g^{k_0} h^{f_0}, g^{k'_0} h^{f'_0}, (c'_0)^{f'_0} g_0^{f_0} d_0) \tag{29}$$

for some $d_0 \neq 1$ that is sent to the verifier.

Proposition 6.11. *Protocol 6.10 is a proof of knowledge for the relation $\mathcal{R}'_{srsa} \vee \mathcal{R}_v$ and $\mathcal{R}'_{srsa} \vee \mathcal{R}_e$ for Case (1) and Case (2) inputs respectively under the discrete logarithm assumption.*

Proof. The extractor simply invokes the extractor of the subprotocol π_{eq}. The proposition allows us to assume that it outputs exponents l_r, l_{s_0}, l_{s_1}, l_{r_0}, l_{r_1}, r_0, r_1, a, l_a, $r' \in [-2^{\kappa_m+2\kappa_r} + 1, 2^{\kappa_m+2\kappa_r} - 1]$, and $s'_0, s'_1 \in [-2^{\kappa_p-\kappa'_p+\kappa_r} + 1, 2^{\kappa_p-\kappa'_p+\kappa_r} - 1]$ such that

- Case (1): Equations (10)-(12), Equation (16), and Equation (23) are satisfied for some exponents t_a, t_0, t_1, or
- Case (2): Equations (13)-(15), Equation (16), and Equation (23) are satisfied for some exponents x, x_0, x_1 and r.

For the unique *sk* such that (*pk*, *sk*) is well-formed it also holds that

$$v_a u_a^{-x} = \eta^a \ , \quad v_0 u_0^{-x} = \eta^{r_0} \ , \quad \text{and} \quad v_1 u_1^{-x} = \eta^{r_1} \ .$$

In other words the committed values are equal to the encrypted values. In addition to this we also have an exponent l_a^* such that

$$C_a = g^{l_a^*} h^{\mathsf{Pack}(r, s_0, s_1)} \ ,$$

where $s_0 = 2s'_0 + 2^{\kappa_p-1} + 1$ and $s_1 = 2s'_1 + 2^{\kappa_p-1} + 1$. We may assume that $(a, l_a) = (\mathsf{Pack}(r, s_0, s_1), l_a^*)$, since otherwise we can define $\eta_1 = l_a^* - l_a$ and $\eta_2 = \mathsf{Pack}(r, s_0, s_1) - a$ and have $((N, g, h), (1, 0, \eta_1, \eta_2)) \in \mathcal{R}'_{srsa}$.

From Equations (19)-(22) we get exponents $r = r' + 2^{2\kappa_r+\kappa_m}$, $e_0 = 2^{\kappa_p}(r+m) + s_0$, l_{e_0}, $e_1 = 2^{\kappa_p} r + s_1$, and l_{e_1} such that

$$(C_{e_0}, C_{e_1}) = (g^{l_{e_0}} h^{e_0}, g^{l_{e_1}} h^{e_1}) \ .$$

Note that $r \in [1, 2^{2\kappa_r+\kappa_m+1} - 1]$ and the exponents s_0 and s_1 are odd by construction and contained in $[0, 2^{\kappa_p} - 1]$, since $\kappa'_p \geq \kappa_r + 2$ by Equation (1). Furthermore, e_0

and e_1 are formed from r, m, s_0, and s_1 as defined by the twin-moduli signature scheme. Thus, all that remains of the verification of the encrypted candidate twin-moduli signature is to consider how e_0 and e_1 relate to $z_0 = c'_0/g_0^{r_0}$ and $z_1 = c'_1/g_0^{r_1}$. Recall that for any well-formed twin-moduli signature both g_0 and g_1 are invertible and we have $z_1^{e_1} = g_1$.

From Equations (24) and (25) we have exponents e_1^*, $l_{e_1}^*$, l_1, l'_1, and j_1 such that

$$\begin{aligned}(C_{e_0}, C_0, C'_0, C''_0) &= (g^{l_{e_0}^*} h^{e_0^*}, g^{l_0} C_{r_0}^{e_0^*}, g^{l'_0} h^{j_0}, (c'_0)^{j_0} (c'_0)^{e_0^*}) \\ (C_{e_1}, C_1, C'_1, C''_1) &= (g^{l_{e_1}^*} h^{e_1^*}, g^{l_1} C_{r_1}^{e_1^*}, g^{l'_1} h^{j_1}, (c'_1)^{j_1} (c'_1)^{e_1^*}) \ .\end{aligned}$$

Similarly to the above, we may assume that $(l_{e_0}^*, e_0^*) = (l_{e_0}, e_0)$ and $(l_{e_1}^*, e_1^*) = (l_{e_1}, e_1)$. From Equation (26) we get additional exponents l''_1 and j'_1 such that

$$(C_1, C''_1/g_1) = (g^{l''_1} h^{j'_1}, (c'_1)^{j_1} g_1^{j'_1}) \ . \tag{30}$$

Thus, we have $g^{l_{r_1} e_1 + l_1} h^{r_1 e_1} = C_1 = g^{j''_1} h^{j'_1}$, and similarly to the above we may assume that $j'_1 = r_1 e_1$. This then implies that $(c'_1)^{j_1} (c'_1)^{e_1} = C''_1 = (c'_1)^{j_1} g_1^{r_1 e_1} g_1$. By assumption both c'_1 and g_1 are invertible modulo N_1. Thus, we have $(c'_1/g_1^{r_1})^{e_1} = g_1$ as expected.

If $b = 1$, i.e., we expect to extract a valid signature, then from Equation (27) we get additional exponents l''_0 and j'_0 such that

$$(C_0, C''_0/g_0) = (g^{l''_0} h^{j'_0}, (c'_0)^{j_0} g_0^{j'_0}) \ . \tag{31}$$

Similarly to the above it follows that $(c'_0/g_0^{r_0})^{e_0} = g_0$ as expected. To summarize, the ciphertext contains a valid, and well-formed, twin-moduli signature (r, z_0, s_0, z_1, s_1) of the message m relative the public key *spk*.

If $b = 0$ we instead have exponents i_0, i'_0, i''_0, k_0, f_0, k'_0, and f'_0 such that

$$\begin{aligned}(B_0, B'_0, B''_0) &= (g^{i_0} C_0^{i''_0}, g^{i'_0} (C'_0)^{i''_0}, (C''_0/g_0)^{i''_0}) \ , \quad \text{and} \\ (B_0, B'_0, B''_0) &= (g^{k_0} h^{f_0}, g^{k'_0} h^{f'_0}, (c'_0)^{f'_0} g_0^{f_0} d_0) \ .\end{aligned}$$

Thus, we have $g_0^{i_0} C_0^{i''_0} = g^{l_{r_0} e_0 i''_0 + l_0 i''_0 + i_0} h^{r_0 e_0 i''_0} = B_0 = g^{k_0} h^{f_0}$ and $g^{i'_0} (C'_0)^{i''_0} = g^{l'_0 i''_0 + i'_0} h^{j_0 i''_0} = B'_0 = g^{k'_0} h^{f'_0}$. We conclude that $f_0 = r_0 e_0 i''_0$ and $f'_0 = j_0 i''_0$. We also have

$$(C''_0/g_0)^{i''_0} = (c'_0)^{j_0 i''_0} ((c'_0)^{e_0}/g_0)^{i''_0} = B''_0 = (c'_0)^{f'_0} g_0^{f_0} d_0 \ .$$

Since both c'_0 and g_0 are invertible modulo N_0, this implies that

$$((c'_0)^{e_0}/g_0)^{i''_0} = g_0^{r_0 e_0 i''_0} ((c'_0/g_0^{r_0})^{e_0}/g_0)^{i''_0} = g_0^{f_0} d_0 \ ,$$

i.e., $((c'_0/g_0^{r_0})^{e_0}/g_0)^{i''_0} = d_0$. Finally, we conclude that $(c'_0/g_0^{r_0})^{e_0}/g_0 \neq 1$ as expected, since $d_0 \neq 1$. To summarize, the ciphertext contains an invalid, but well-formed, twin-moduli signature (r, z_0, s_0, z_1, s_1) of the message m relative the public key *spk*. □

Let T'_{hs} be identical to T_{hs} except that the parameters (N_r, g_r) are excluded from the output, and let F'_{hs} be defined as F_{hs} except that (N_r, g_r) is not in its input. Define T'_{cs} and F'_{cs} correspondingly. We prove the following two propositions jointly.

Proposition 6.12. *Protocol 6.10 is* $(T_\emptyset \| T_\emptyset, F'_{srsa} \| \mathsf{Kg}_\mathsf{s}^{\mathsf{dc}})$*-zero-knowledge for Case (1) inputs under the discrete logarithm assumption.*

Proposition 6.13. *Protocol 6.10 is* $(T_\emptyset \| T'_{hs}, F'_{srsa} \| F'_{hs})$*-zero-knowledge as well as* $(T_\emptyset \| T'_{cs}, F'_{srsa} \| F'_{cs})$*-zero-knowledge for Case (2) inputs under the discrete logarithm assumption.*

Proof (Propositions 6.12 and 6.13). Consider first an input with $b = 1$, i.e., the DC-signature is valid. The simulator chooses $C'_r, C'_{s_0}, C'_{s_1}, C_{r_0}, C_{r_1}, C_0, C'_0, C_1, C'_1 \in SQ_N$, $C''_0 \in \langle c'_0 \rangle$ and $C''_1 \in \langle c'_1 \rangle$ randomly and then invokes the zero-knowledge simulator of π_{eq}. Since all commitments are statistically hiding it follows from the zero-knowledge property of π_{eq} that the resulting simulation is acceptable.

Consider now an input with $b = 0$, i.e., the DC-signature is invalid. In this case the simulator chooses the commitments above as before, but also computes B_0, B'_0, B''_0 as in the protocol. Provided that the simulator can form a correctly distributed d_0, it can then invoke the zero-knowledge simulator of π_{eq}, and we may conclude that the protocol is zero-knowledge. Unfortunately, it seems difficult to generate a correctly distributed d_0 for an arbitrary *spk* that may be maliciously generated. The problem is that N_0, z_0, and e_0 may be chosen maliciously, and if they are it may be the case that $z_0^{e_0}/g_0$ generates some particular subgroup of $\mathbb{Z}^*_{N_0}$. The distinguisher could then tell the difference between a simulation and a real execution using the factorization of N_0. Fortunately, we do not need a zero-knowledge simulator.

The (T'_{cs}, F'_{cs})-zero-knowledge simulator can of course use its advice to compute d_0 exactly as is done in the protocol. Thus, the distribution of the simulated d_0 is identical to that in a real execution.

The (T'_{hs}, F'_{hs})-zero-knowledge simulator simply chooses $d_0 \in SQ_{N_0}$ randomly. This works, since for the instance the protocol is executed, the signature is constructed honestly. More precisely, we know that $z_0^{e'_0} = g_0$ for some $e'_0 \neq e_0$. Thus, z_0 generates SQ_{N_0} and we may write $z_0^{e_0}/g_0 = z_0^{e_0 - e'_0}$. By construction $e_0 = 2^{\kappa_p}(r + m) + s_0$ with $r \in [1, 2^{\kappa_m + 2\kappa_r} - 1]$ and $m \in [0, 2^{\kappa_m} - 1]$, so $|e_0 - e'_0|$ is smaller than the orders of both non-trivial subgroups of SQ_{N_0} for every $m \in [0, 2^{\kappa_m} - 1]$ due to Equation (2). This implies that $z_0^{e_0 - e'_0}$ generates SQ_{N_0}.

Thus, we conclude that the proposition is true. □

We now have the following corollaries from Lemma 5.12.

Corollary 6.14. *The special composition* π_v *of* π_{tp} *and Protocol 6.10 for Case (1) is a* $\mathsf{Kg}_\mathsf{s}^{\mathsf{dc}}$*-zero-knowledge proof of knowledge for the relation* $\mathcal{R}_v$ *under the strong RSA-assumption and the discrete logarithm assumption.*

Corollary 6.15. *The special composition* π_e *of* π_{tp} *and Protocol 6.10 for Case (2) is a* (T_{hs}, F_{hs})*-zero-knowledge and* (T_{cs}, F_{cs})*-zero-knowledge proof of knowledge for the relation* $\mathcal{R}_v$ *under the strong RSA-assumption and the discrete logarithm assumption.*

6.7 Complexity Analysis

For simplicity we estimate the efficiency of the scheme without any non-trivial optimizations. For each composite modulus used in the protocols either the prover or the verifier knows the factorization and can exponentiate using Chinese remaindering.

There are also a number of well known techniques to improve the efficiency of schemes such us ours, e.g., off-line precomputation, combining commitments, and speeding up computing multiple exponentiations and fixed based exponentiations. We expect these techniques to reduce the complexity of our protocols substantially.

We use the following assumptions in our analysis. Exponentiation modulo a $(\kappa+\kappa')$-integer is computationally equivalent to $(\frac{\kappa+\kappa'}{\kappa})^3$ exponentiations modulo a κ-bit integer. Exponentiation modulo a κ-bit integer, where the exponent has $\kappa+\kappa'$ bits is equivalent to $1+\frac{\kappa'}{\kappa}$ exponentiations modulo a κ-bit integer.

We assume that the (f, f')-embedding assumption holds with $\kappa_p = f(\kappa_r+\kappa_m+1) = 2\kappa_r$ and $f'(\kappa_r+\kappa_m+1) = \kappa_r+2$. For our estimates below we assume that on average the embedding algorithm $\mathsf{Emb}_f^{f'}$ requires the equivalent of $2\kappa'(\frac{\kappa'}{\kappa})^3$ κ-bit exponentiations, where $\kappa' = \kappa_p + \kappa_m + 2\kappa_r$ to find an embedding. In other words we assume that given $r \in [0, 2^{\kappa_m+2\kappa_r} - 1]$ the embedding algorithm tries random odd s until $e = 2^{\kappa_p} r + s$ is prime, and we assume that each such odd integer is prime with probability roughly $\delta = \frac{2}{\kappa' \ln 2}$ (this is reasonable since a random κ'-bit integer is prime with probability roughly $1/\kappa \ln 2$). We also assume that in each iteration the Miller-Rabin test identifies a composite as such with probability 3/4, and that the output probable prime passes κ_c iterations. Thus, our estimate follows from estimating the extected number of exponentiations to $\sum_{i=0}^{\infty} \delta(1-\delta)^i \frac{4}{3} i + \kappa_c \leq 2\kappa'$, for reasonable choices of κ_r and κ_c.

Recall that the Cramer et al. [7] protocol π_{pl} for proving knowledge of a logarithm requires 4 exponentiations for the prover and 6 exponentiations for the verifier over the prime order group in which the protocol is executed.

By tedious book-keeping we have written a program that computes the complexity of all operations and protocols taking into account the various bit-sizes of moduli and exponents. Table 1 summarizes the results for a set of practical parameters. We stress that these values do not include the cost for checking primality and for generating RSA-parameters, see Remark 5.6 and 5.11 for a discussion on this. The size of a signature is $18\kappa_{pa} + 2\kappa_{rsa}$ bits, which for the choice of parameters in Table 1 amounts to about 22kB.

Operation	Alg./Prot.	Signer	Confirmer	Verifier
Signing	$\mathsf{Sig}^{\mathsf{dc}}$	140		
Converting	$\mathsf{Con}^{\mathsf{dc}}$		66	
Verifying	$\mathsf{Vf}^{\mathsf{dc}}$			1
Well-Formedness	π_{wf}		61	59
Correctness of conversion	π_c		327	227
Validity/Invalidity	π_e		189	169
Validity	π_v	166		151

Table 1: The estimated average complexity of the algorithms and the protocols in terms of κ-bit exponentiations when $\kappa = 1024$, $\kappa_r = \kappa_c = 50$, and $\kappa_m = 160$.

7 Acknowledgments

I thank Ronald Cramer and Ivan Damgård for answering my questions about their work, I thank Dominik Raub and Stefano Tessaro for discussions on non-transferability, and I thank the anonymous reviewers of TCC 2007 for helpful comments.

References

[1] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature scheme. In *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer Verlag, 1997.

[2] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology – Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[3] J. Camenisch and Markus Michels. Confirmer signature schemes secure against adaptive adversaries. In *Advances in Cryptology – Eurocrypt 2000*, Lecture Notes in Computer Science, pages 243–258. Springer Verlag, 2000.

[4] J. Camensisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology – Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer Verlag, 2003.

[5] D. Chaum. Designated confirmer signatures. In *Advances in Cryptology – Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 86–91. Springer Verlag, 1994.

[6] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Advances in Cryptology – Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer Verlag, 1990.

[7] R. Cramer, I. Damgård, and P. D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *Public Key Cryptography – PKC 2000*, volume 1751, pages 354–372. Springer Verlag, 2000.

[8] R Cramer, I. Damgård, and T. P. Pedersen. Efficient and provable security amplifications. In *Security Protocols, International Workshop, Cambridge, United Kingdom, April 10-12, 1996, Proceedings*, volume 1189 of *Lecture Notes in Computer Science*, pages 101–109. Springer, 1996.

[9] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *6th ACM Conference on Computer and Communications Security (CCS)*, pages 46–51. ACM Press, 1999.

[10] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. `http://homepages.cwi.nl/~cramer/`, June 1999.

[11] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology – Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer Verlag, 2002.

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

[13] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd ACM Symposium on the Theory of Computing (STOC)*, pages 542–552. ACM Press, 1991.

[14] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 1997.

[15] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer Verlag, 1999.

[16] C. Gentry, D. Molnar, and Z. Ramzan. Efficient designated confirmer signatures without random oracles or zero-knowledge proofs (extended abstract). In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 662–681. Springer Verlag, 2005.

[17] O. Goldreich. A uniform-complexity treatment of encryption and zeroknowledge. *Journal of Cryptology*, 6(1):21–53, 1993.

[18] O. Goldreich. On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits. In *4th Theory of Cryptography Conference (TCC)*, volume 4392 of *Lecture Notes in Computer Science*, pages 174–193. Springer Verlag, 2007.

[19] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[20] S. Goldwasser and E. Waisbard. Transformation of digital signature schemes into designated confirmer signatures. In *1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 77–100. Springer Verlag, 2004.

[21] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology – Eurocrypt '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer Verlag, 1996.

[22] W. Mao and C.H. Lim. Cryptanalysis of subgroups of $\mathbb{Z}_n^*$ advances in cryptology. In *Advances in Cryptology – Asiacrypt '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 214–216. Springer Verlag, 1998.

[23] M. Michels and M. Stadler. Generic constructions for secure and efficient confirmer signature schemes. In *Advances in Cryptology – Eurocrypt 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 406–421. Springer Verlag, 1998.

[24] T. Okamoto. Designated confirmer signatures and public key encryption are equivalent. In *Advances in Cryptology – Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 61–74. Springer Verlag, 1994.

[25] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Verlag, 1999.

[26] R. Pass and A. Rosen. New and improved constructions of non-malleable cryptographic protocols. In *37th ACM Symposium on the Theory of Computing (STOC)*, pages 533–542. ACM Press, 2005.

[27] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology – Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 1998.

[28] D. Wikström. On the security of mix-nets and hierarchical group signatures. Doctoral thesis, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, TRITA NA 05-38, ISSN 0348-2952, ISRN KTH/NA/R--05/38--SE, ISBN 91-7283-717-9, December 2005, `http://www.kth.se`.

[29] Douglas Wikström. Designated confirmer signatures revisited. In *4th Theory of Cryptography Conference (TCC)*, volume 4392 of *Lecture Notes in Computer Science*, pages 342–361. Springer Verlag, 2007.

A Detailed Descriptions of Problems With Some Previous Definitions

In this section we describe problems we see with some previous definitions.

A.1 Camenisch and Michels [3]

For the discussion in this section we assume familiarity with [3]. Consider "Security for the signer" on page 246 in [3]. First the key generators for the signer and confirmer are executed and the adversary is given the public key y_S of the signer and the public key y_C and secret key x_C of the confirmer. Then the adversary is allowed oracle access to the signer, and it must output a signature of a message it did not send as a query to the signer. For every adversary its success probability should be negligible, where the probability is taken over the key generators. The definition says nothing about the success probability when the adversary can *choose* y_C, i.e., unforgeability is not guaranteed against malicious confirmers.

Consider the definition of non-transferability on page 247 in [3]. The definition states that there must exist a "simulator" such that Game 1 and Game 2 are indistinguishable. The notion of a simulator is not defined, but we assume that it is an expected or strict polynomial machine. Game 2 is defined in such a way that the simulator must execute an interactive proof with the adversary *without using a witness*, and *without rewinding*. This is of course impossible, since the existence of such a simulator contradicts the soundness of the interactive proof. Most likely the authors meant to formalize an experiment such that the simulator can be rewound.

A.2 Goldwasser and Waisbard [20]

For the discussion in this section we assume familiarity with [20]. Consider the definition of unforgeability in [20], i.e., Definition 1, point 6(a). The forging algorithm is given the signers public key PK_s and the confirmers public key PK_c and secret key SK_c. Then the forger is allowed to request signatures of arbitrary messages and must come up with a valid signature of a new message. A scheme is unforgeable if no adversary can do this with non-negligible probability, where the probability is taken over the randomness of the keys PK_s, PK_c, and SK_c. This means that the definition says nothing about the probability of forging a valid signature of a new message when the adversary is allowed to *choose* PK_c, i.e., unforgeability is not guaranteed against malicious confirmers.

Goldwasser and Waisbard [20] claim that it suffices to use witness hiding protocols. However, Definition 1 point 6(b) in [20], that deals with non-transferability only requires that the adversary can not convince a third party using the confirmation protocol $ConfirmedSign$ of the designated signature scheme. It does not rule out the existence of another protocol *different* from the confirmation protocol for proving the validity of a signature. More precisely, there could exist a sound protocol that is *not* a proof of knowledge, with which the adversary can convince a distrustful verifier. The witness-hiding property of the protocols does not rule out that the adversary by interacting with the signer and confirmer gains the ability to convince a distrustful verifier using such a protocol.

Going back to the job offer scenario, this means that nothing prevents a candidate from convincing a third party that it was given a job offer from a certain employer.

Another problem with the definition in [20] is that the adversary is not given access to a conversion oracle. Thus, the definition says nothing about the ability of the adversary to forge signatures or transfer knowledge of the validity of signatures, when it sees some converted signatures.

A.3 Gentry, Molnar, and Ramzan [16]

For the discussion in this section we assume familiarity with [16]. Consider the definition of experiment “Exp-No-FoolVerifier” on page 72 in [16]. It clearly states that adversary is given the public key $\mathsf{SGk}_{\mathcal{S}}$ of the signer and secret key $\mathsf{Sk}_{\mathcal{C}}$ of the confirmer. However, these keys are honestly generated. Thus, the definition says nothing about the success probability of the adversary when it is allowed to choose the confirmer key, i.e., unforgeability is not ensured for malicious confirmers.

As for transferability the authors of [16] admit (see page 72) that they do not even aim for a solution that is non-transferable. Indeed, the holder of a DC-signature using their definition can always prove that the signer was involved in the computation of the signature.

B Proof of Equal Exponents

Denote by $\pi_{pl} = (P_{pl}, V_{pl})$ the zero-knowledge proof of knowledge of a logarithm for prime order groups described by Cramer et al. [7].

Protocol B.1 (General Equality Relations).
Common Input: An integer N, and $g, h \in \mathbb{Z}_N^*$. Positive integers $n_1, \ldots, n_k$, positive

integer bit-sizes $(((\kappa_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k}$, elements $(((g_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k}$ and $((u_{i,j})_{j=1}^{t_i})_{i=1}^{k})$, where for all j, l it holds that $g_{i,j,l}, u_{i,j} \in \mathbb{Z}_{n_i}^*$.
PRIVATE INPUT: Exponents $(((x_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k}$, in the intervals $x_{i,j,l} \in [-2^{\kappa_{i,j,l}} + 1, 2^{\kappa_{i,j,l}} - 1]$ such that for all i, j it holds that $u_{i,j} = \prod_{l=1}^{l_j} g_{i,j,l}^{x_{i,j,l}} \bmod n_i$.

1. The verifier chooses a safe κ-bit prime $P = 2Q + 1$, and $H \in G_Q$ randomly. Then it chooses $x \in \mathbb{Z}_Q$ randomly, computes $K = H^x$, hands (P, H, K) to the generator, and executes π_{pl} as the prover on common input (P, H, K) and private input x. If the verifier of π_{pl} rejects the prover hands $\perp$ to the verifier and halts.

2. The prover chooses $r \in [0, 2^{\kappa+\kappa_r} - 1]$, $c_p \in [0, 2^{\kappa_c - 1} - 1]$, and $z_{i,j,l} \in [0, 2^{\kappa+\kappa_r} - 1]$ randomly, computes $w = K^r H^{c_p}$, $v_{i,j,l} = g^{z_{i,j,l}} h^{x_{i,j,l}}$, and hands the tuple $(w, (((v_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k})$ to the verifier.

3. The prover checks that P is a safe prime. Then it chooses $r_{i,j,l} \in [0, 2^{\kappa_{i,j,l}+\kappa_c+\kappa_r} - 1]$ and $r'_{i,j,l} \in [0, 2^{\kappa+\kappa_c+2\kappa_r} - 1]$ randomly, computes $\alpha_{i,j} = \prod_{l=1}^{s_j} g_{i,j,l}^{r_{i,j,l}} \bmod n_i$, and $\beta_{i,j,l} = g^{r'_{i,j,l}} h^{r_{i,j,l}} \bmod N$, and hands $((\alpha_{i,j}, (\beta_{i,j,l}, \gamma_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k})$ to the verifier.

4. The verifier chooses $c_v \in [2^{\kappa_c - 1}, 2^{\kappa} - 1]$ randomly and hands c_v to the prover.

5. The prover computes $c = c_p \oplus c_v$, $d_{i,j,l} = x_{i,j,l} c + r_{i,j,l} \bmod 2^{\kappa_{i,j,l}+\kappa_c+\kappa_r}$, and $e_{i,j,l} = z_{i,j,l} c + r'_{i,j,l} \bmod 2^{\kappa+\kappa_c+2\kappa_r}$ and hands $(r, c_p, (((d_{i,j,l}, e_{i,j,l})_{l=1}^{s_j})_{j=1}^{t_i})_{i=1}^{k})$ to the verifier.

6. The verifier checks that $c_p \in [0, 2^{\kappa_c - 1} - 1]$, $w = H^r K^{c_p}$, $u_{i,j}^c \alpha_{i,j} = \prod_{l=1}^{s_j} g_{i,j,l}^{d_{i,j,l}} \bmod n_i$, and that $v_{i,j,l}^c \beta_{i,j,l} = g^{e_{i,j,l}} h^{d_{i,j,l}} \bmod N$.

It is easy to see that a verifier can reuse the same (P, H, K) in every invokation of the protocol, and these parameters can be generated in advance. We state the protocol as above to avoid unnecessary cluttering of our notation. We denote the protocol above by $\pi_{eq} = (P_{eq}, V_{eq})$.

Remark B.2. For exponents $x_{i,j,1}$ and $x_{i,j,2}$ such that $n_i = N$ and $u_{i,j} = g^{x_{i,j,1}} h^{x_{i,j,2}}$ there is no need to construct separate commitments $v_{i,j,1}$ and $v_{i,j,2}$. We use this observation in our complexity estimates.

C Cryptographic Assumptions

C.1 Safe Prime Assumption

Recall that a prime p is said to be safe if $(p - 1)/2$ is prime. It is not even known if there are infinitely many safe primes, but in practice the probability that a random prime p is safe is roughly $1/\log p$. Formally, we make an assumption.

Definition C.1 (Safe Prime Assumption). There exists an integer $c > 0$ such that a random κ-bit prime p is safe with probability at least κ^{-c}.

C.2 Strong RSA-Assumption

The strong RSA-assumption says that it is infeasible to compute any non-trivial root of a random element in SQ_N where N is an RSA-modulus, even if allowed to select which root to compute.

This assumption was first considered by Barić and Pfitzmann [1] and differs from the standard RSA-assumption in that the root to compute is not predetermined. Currently the fastest known method to solve this problem is to factor N, but it is not known if the strong RSA-assumption is equivalent to the factoring assumption.

Definition C.2 (Strong RSA-Assumption). Let p and q be randomly chosen $\kappa/2$-bit safe primes, define $N = pq$, and let $g \in SQ_N$ be randomly chosen. Then for every $A \in \mathrm{PPT}$ the probability $\Pr[A(N, g) = (b, e) \wedge e \neq \pm 1 \wedge b^e = g \bmod N]$ is negligible.

In contrast to the original assumption made in [1] we assume that the factors of N are safe primes, but it is easy to see that our assumption is equivalent to the original if the safe prime assumption holds.

To simplify our proofs use use the following lemma. A weak version of the lemma can be found in Damgård and Fujisaki [11], but our lemma is slightly stronger. In their analysis it is essential that the bit-size of η_0 is smaller than $\kappa/2$. In [28] it is shown that this restriction is not necessary.

Lemma C.3 (Variants of Strong RSA-Assumption). *Assume that the strong RSA-assumption is true. Let p and q be randomly chosen $\kappa/2$-bit safe primes, define $N = pq$, and let $g, h \in SQ_N$ be random. Then for all adversaries $A \in \mathrm{PPT}$ the probabilities*

$$\begin{aligned}&\Pr[A(N, g, h) = (b, \eta_0, \eta_1, \eta_2) \wedge \eta_0 \neq 0 \wedge (\eta_0 \nmid \eta_1 \vee \eta_0 \nmid \eta_2)\\ &\qquad \wedge b^{\eta_0} = g^{\eta_1} h^{\eta_2} \bmod N]\\ &\Pr[A(N, g, h) = (b, \eta_1, \eta_2) \wedge (\eta_1, \eta_2) \neq (0, 0) \wedge g^{\eta_1} = h^{\eta_2} \bmod N]\end{aligned}$$

are negligible.

C.2.1 A Simplifying Convention

Consider any computation involving an RSA-modulus N where the inverse of an element $a \in \mathbb{Z}_N$ must be computed. In principle, it could happen that a is not a unit in $\mathbb{Z}_N$. However, if such an element is encountered with non-negligible probability in a computation where the factorization of N is not known, we have of course found one of the factors of N and the strong RSA-assumption is broken.

To simplify the exposition of the protocols and their analysis we assume, without loss, that all elements in $\mathbb{Z}_N$ that appear in the simulations in the security analyses always can be inverted.

C.3 Composite Residuosity Class Assumptions

Assumptions about the hardness of computing and deciding composite residuosity classes were first considered by Paillier [25] to prove the security of the Paillier cryptosystem.

Let p and q be distinct primes with the same number of bits and define $n = pq$ and $\eta = n+1$. Define also $\mathcal{E}_\eta : \mathbb{Z}_n \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^2}^*$, $\mathcal{E}_\eta : (m, r) \mapsto \eta^m r^n \bmod n^2$. Then Lemma 3 in [25] states that $\mathcal{E}_\eta$ is a bijection.

Definition C.4 (Residue Class). The n-th residue class $[u]_\eta$ of u with respect to η is the unique $m \in \mathbb{Z}_n$ such that there exists an $r \in \mathbb{Z}_n^*$ such that $\mathcal{E}_\eta(m, r) = u$.

Definition C.5 (Composite Residuosity Assumption). Let p and q be randomly chosen $\kappa/2$-bit safe primes and define $n = pq$. Let $u \in \mathbb{Z}_{n^2}^*$ be randomly chosen. The composite residuosity (CR) assumption states that for all adversaries A the probability $\Pr[A(n, u) = [u]_\eta]$ is negligible.

Definition C.6 (Decision Composite Residuosity Assumption). Let p and q be randomly chosen $\kappa/2$-bit safe primes and define $n = pq$. Let $u \in \mathbb{Z}_{n^2}^*$ be randomly chosen. The decision composite residuosity (DCR) assumption states that for all adversaries $A \in \mathrm{PPT}$ the absolute value $|\Pr[A(n, u) = 1] - \Pr[A(n, u^n \bmod n^2) = 1]|$ is negligible.

In contrast to Paillier, we assume that the factors of n are safe primes, but it is easy to see that our assumption follows from the original under the safe prime assumption.

C.4 Discrete Logarithm Assumption

There seems to be no consensus on a formal definition of a "standard discrete logarithm assumption" in a subgroup of the multiplicative group modulo a prime. Thus, we take the liberty to simply call the specific assumption we define below "the discrete logarithm assumption" without any qualifier.

Definition C.7 (Discrete Logarithm Assumption). If $P = 2Q + 1$ is a random safe prime, and $g, \beta \in G_Q$ are random, where G_Q is the group of squares in $\mathbb{Z}_P^*$, then for every adversary $A \in \mathrm{PPT}$ the probability $\Pr[A(P, g, \beta) = \log_g \beta]$ is negligible.

C.5 Decision Diffie-Hellman Assumption

Similarly to the discrete logarithm problem, the decision Diffie-Hellman problem can be considered over many different groups and there seems to be no consensus on a standard group. Again we use no qualifier.

Definition C.8 (Decision Diffie-Hellman Assumption). Let $P = 2Q + 1$ be a random safe prime, let G_Q be the group of squares of $\mathbb{Z}_P^*$, let g generate G_Q, and let $a, b, c \in \mathbb{Z}_Q$ be randomly chosen. Then for every adversary $A \in \mathrm{PPT}$ the absolute value $|\Pr[A(P, g, g^a, g^b, g^{ab}) = 1] - \Pr[A(P, g, g^a, g^b, g^c) = 1]|$ is negligible.

A hybrid argument shows that a more slightly more general statement holds. Namely, if $b', b'', c', c'' \in \mathbb{Z}_Q$ are chosen randomly as well, then the absolute value above can be replaced by $|\Pr[A(P, g, g^a, g^b, g^{ab}, g^{b'}, g^{ab'}, g^{b''}, g^{ab''}) = 1]$
$- \Pr[A(P, g, g^a, g^b, g^c, g^{b'}, g^{c'}, g^{b''}, g^{c''}) = 1]|$.

D Standard Definitions

Definition D.1 (Signature Scheme). A signature scheme $\mathcal{SS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ consists of three polynomial-time algorithms

1. A probabilistic key generation algorithm Kg that on input 1^κ outputs a public key *spk* and a private key *ssk*.
2. A probabilistic signature algorithm Sig that on input a private key *ssk* and a message m outputs a signature s.
3. A probabilistic verification algorithm Vf that on input a public key *spk*, a message m, and a signature s outputs a bit $b \in \{0, 1\}$.

Furthermore, for each output (ssk, spk) of $\mathsf{Kg}(1^\kappa)$ and each m it must hold that $\mathsf{Vf}_{spk}(m, \mathsf{Sig}_{ssk}(m)) = 1$.

Experiment D.2 (CMA-Security, $\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa)$).

$$\begin{aligned} (spk, ssk) &\leftarrow \mathsf{Kg}(1^\kappa) \\ (m, s) &\leftarrow A^{\mathsf{Sig}_{ssk}(\cdot)}(\mathsf{guess}, spk) \end{aligned}$$

If $\mathsf{Vf}_{spk}(m, s) = 1$ *and A did not ask for a signature of m return* 1*, else return* 0*.*

Definition D.3 (CMA-Security). A signature scheme $\mathcal{SS}$ is CMA-secure if for every adversary $A \in \mathrm{PPT}$ the probability $\Pr[\mathsf{Exp}^{\mathsf{cma}}_{\mathcal{SS},A}(\kappa) = 1]$ is negligible in κ.

E Program Used To Estimate the Complexity

```
; (load "efficiency.scm")
; (complexity 1024 50 50 160 50)

(define (complexity secpRSA secpR secpC secpM secpPPad)
  (let ((secpDL secpRSA)
        (secpPA (+ secpRSA (* 2 secpR) 3)))
    (let ((secpDDH (+ secpPA (* 2 secpR) 3))
          (secpP (+ secpR secpPPad)))

    ; The cost of various types of exponentiations.
    (define (RSAexp secp)                                         ; Our estimate is in terms of RSA-exponentiations.
      (/ secp secpRSA))
    (define (DLexp secp)                                          ; DL-prime has same size as RSA-modulus.
      (/ secp secpDL))
    (define (PAexp secp)                                          ; PA-modulus has same size as RSA-modulus,
      (* 8 (/ secpPA secpRSA) (/ secpPA secpRSA) (/ secpPA secpRSA)      ; but we compute modulo the square of the modulus.
         (/ secp (* 2 secpPA))))
    (define (DDHexp secp)                                         ; DDH-prime is larger than RSA-modulus.
      (* (/ secpDDH secpRSA) (/ secpDDH secpRSA) (/ secpDDH secpRSA)
         (/ secp secpDDH)))
    (define (OneExp secp)                                         ; Cost of a single secp-bit exponentiation,
      (* (/ secp secpRSA) (/ secp secpRSA) (/ secp secpRSA)))     ; which is used in primality testing.

    ; Computing a twin-moduli signature.
    (define (Sig)
      (+ (* 4 (RSAexp secpRSA))                                   ; Inversion in exponent and exponentiation.
         (* 4 (+ secpM secpR secpR secpP)                         ; Embedding string into prime.
            (OneExp (+ secpM secpR secpR secpP)))))

    ; Computing a DC-signature.
    (define (desSig)
      (+ (Sig)                                                    ; compute signature
         (* 9 (PAexp (+ secpPA secpR)))                           ; t_a
         (* 2 (RSAexp (+ secpRSA secpR)))                         ; r_0, r_1
         (* 3 (PAexp secpPA))                                     ; 2n decoding in hash
         (PAexp secpRSA)))                                        ; e

    ; Converting a signature.
    (define (desConv)
      (+ (* 3 (PAexp secpPA))                                     ; 2n decoding
         (* 3 (PAexp (+ secpPA secpR secpRSA)))                   ; validity
         (* 3 (PAexp (+ secpPA secpR)))                           ; x decryption
```

```
     (* 2 (RSAexp (+ secpRSA secpR)))))                               ; (r_0,r_1)-decoding

; Verifying a converted signature.
(define (desVf)
  (* 2 (RSAexp (+ secpM secpR secpR secpP))))                         ; e_0, e_1 exponentiations

; Cramer et al. zero-knowledge proof of knowledge of logarithm.
(define (PpiPL) (* 4 (DLexp secpDL)))                                 ; Prover exponentiations
(define (VpiPL) (* 6 (DLexp secpDL)))                                 ; Verifier exponentiations

; Protocol for generating Fujisaki-Okamoto parameters.
(define (GpiTP)
  (+ (VpiPL)                                                          ; Cramer et al. PoK exponent, verifier
     (* 2 (RSAexp (+ (* 2 secpRSA) secpR)))                           ; x exponentiations
     (+ (DLexp secpC)                                                 ; c_g
        (DLexp secpDL))                                               ; r_g
     (* 2 (RSAexp (+ (* 2 (+ secpRSA secpR)) secpC)))))               ; r exponentiations
(define (RpiTP)
  (+ (PpiPL)                                                          ; Cramer et al. PoK exponent, prover
     (+ (DLexp secpC)                                                 ; c_g
        (DLexp secpDL))                                               ; r_g
     (* 2 (+ (RSAexp (+ (* 2 (+ secpRSA secpR)) secpC))               ; d
             (RSAexp secpC)))))                                       ; c

; The overhead cost of the general equality protocol that is independent
; of the particular relation that is considered.
(define (PpiEQoverhead)
  (+ (RpiTP)                                                          ; Generate commitment parameters
     (VpiPL)                                                          ; Cramer et al. PoK exponent, verifier
     (DLexp secpDL)                                                   ; r
     (DLexp secpC)))                                                  ; c_p
(define (VpiEQoverhead)
  (+ (GpiTP)                                                          ; Generate commitment parameters
     (PpiPL)                                                          ; Cramer et al. PoK exponent, prover
     (DLexp secpDL)                                                   ; r
     (DLexp secpC)))                                                  ; c_p

; The overhead cost for forming a commitment of a secp-bit value
; that is not committed to otherwise.
(define (PpiEQ-form secp)
  (+ (RSAexp secp)                                                    ; The value to be committed
     (RSAexp (+ secpRSA secpR))))                                     ; Random value exponentiation

; The overhead cost in the general equality proof for each unique exponent
; that is not committed to otherwise.
(define (PpiEQ-comm secp)
  (+ (RSAexp (+ secpRSA (* 2 secpR) secpC))                           ; Proof about random value
     (RSAexp (+ secp secpR secpC))))                                  ; Proof about committed value
(define (VpiEQ-comm secp)
  (+ (RSAexp (+ secpRSA (* 2 secpR) secpC))                           ; Reply exponentiations
     (RSAexp (+ secp secpR secpC))                                    ;
     (RSAexp secpC)))                                                 ; Challenge exponentiation

; The general equality proof for a single exponent with secp
; bits for various moduli.
;
; Provers
(define (PpiEQ-RSA secp) (RSAexp (+ secp secpR secpC)))
(define (PpiEQ-PA secp) (PAexp (+ secp secpR secpC)))
(define (PpiEQ-DDH secp) (DDHexp (+ secp secpR secpC)))
;
; Verifiers
(define (VpiEQ-RSA secp) (+ (RSAexp (+ secp secpR secpC))
                            (RSAexp secpC)))
(define (VpiEQ-PA secp) (+ (PAexp (+ secp secpR secpC))
                           (PAexp secpC)))
(define (VpiEQ-DDH secp) (+ (DDHexp (+ secp secpR secpC))
                           (DDHexp secpC)))

; Proof of knowledge of well-formedness.
(define (PpiVF)
  (+ (PpiEQoverhead)                                                  ; EQ-prot overhead
     (* 3 (PpiEQ-form (+ secpPA secpR)))                              ; Form commitments of x, x_0, x_1
     (* 3 (PpiEQ-comm (+ secpPA secpR)))                              ; Proof overhead from these commitments
     (* 3 (PpiEQ-PA (+ secpPA secpR)))                                ; Proofs about x, x_0, x_1
     (* 3 (PpiEQ-DDH (+ secpPA secpR)))                               ; Proofs about x, x_0, x_1
     (* 4 (DDHexp secpDDH))))                                         ; r
(define (VpiVF)
  (+ (VpiEQoverhead)                                                  ; EQ-prot overhead
     (* 3 (VpiEQ-comm (+ secpPA secpR)))                              ; Proof overhead from commitments of x, x_0, x_1
     (* 3 (VpiEQ-PA (+ secpPA secpR)))                                ; Proofs about x, x_0, x_1
     (* 3 (VpiEQ-DDH (+ secpPA secpR)))                               ; Proofs about x, x_0, x_1
     (* 4 (DDHexp (+ secpDDH secpC)))))                               ; r

; Proof of knowledge of correct conversion of valid cryptotext.
(define (PpiXvalid)
  (+ (PpiVF)                                                          ; The key is valid
     (* 3 (PAexp secpRSA))                                            ; Exponentiation by e
     (* 9 (PpiEQ-PA (+ secpPA secpR)))                                ; The x, x_0, x_1 exponents
     (* 2 (+ (PpiEQ-form (+ secpRSA secpR))                           ; Form commitments of r_0, r_1
             (PpiEQ-comm (+ secpRSA secpR))))                         ; Overhead for proofs about r_0, r_1
     (* 4 (PpiEQ-PA secpPA))))                                        ; The r_0, r_1 exponents
(define (VpiXvalid)
  (+ (VpiVF)                                                          ; The key is valid
     (* 3 (PAexp secpRSA))                                            ; Exponentiation by e
     (* 9 (PpiEQ-PA (+ secpPA secpR)))                                ; The x, x_0, x_1 exponents
     (* 2 (PpiEQ-comm (+ secpRSA secpR)))                             ; Overhead for proofs about r_0, r_1
     (* 4 (PpiEQ-PA secpPA))))                                        ; The r_0, r_1 exponents

; Proof of knowledge of correct conversion of invalid cryptotext.
(define (PpiXinvalid)
  (+ (PpiVF)                                                          ; The key is valid
     (* 3 (PAexp (+ secpPA secpR)))                                   ; Exponentiation by e
     (* 3 (PAexp secpPA))                                             ; Inverting w_a,w_0,w_1
     (* 3 (PAexp (+ secpPA secpR)))                                   ; Form w', k_a, k_0, k_1
     (* 3 (+ (* 2 (RSAexp (+ secpRSA secpR)))                         ; Form commitments, t_a, l_a
             (* 2 (RSAexp (+ secpPA secpR)))                          ; s_a, k_a over RSA-modulus
             (* 4 (PAexp (+ secpPA secpR)))))                         ; s_a, k_a, x_0, x_1 over PA-modulus
     (* 3 (+ (PpiEQ-RSA (+ secpRSA secpR))                            ; t_a
             (PpiEQ-RSA (+ secpPA secpR))                             ; s_a
             (PpiEQ-PA (+ secpPA secpR))                              ; s_a
             (PpiEQ-PA (+ secpPA secpR))                              ; x_0
             (PpiEQ-PA (+ secpPA secpR))                              ; x_1
             (PpiEQ-RSA (+ secpRSA secpR))                            ; l_a
             (PpiEQ-RSA (+ secpPA secpR))                             ; k_a
             (PpiEQ-PA (+ secpPA secpR))))                            ; k_a
     (* 3 (+ (PpiEQ-RSA (+ secpRSA secpPA secpR secpR secpR))         ; t_a',t_0',t_1'
             (PpiEQ-RSA (+ secpPA secpPA secpR secpR secpR))          ; s_a',s_0',s_1'
             (PpiEQ-PA (+ secpPA secpPA secpR secpR secpR))))))       ; s_a'
(define (VpiXinvalid)
  (+ (VpiVF)                                                          ; The key is valid
     (* 3 (PAexp secpRSA))                                            ; Exponentiation by e
     (* 3 (PAexp secpPA))                                             ; Inverting w_a,w_0,w_1
     (* 3 (+ (PpiEQ-RSA (+ secpRSA secpR))                            ; t_a
             (PpiEQ-RSA (+ secpPA secpR))                             ; s_a
             (PpiEQ-PA (+ secpPA secpR))                              ; s_a
             (PpiEQ-PA (+ secpPA secpR))                              ; x_0
             (PpiEQ-PA (+ secpPA secpR))                              ; x_1
             (PpiEQ-RSA (+ secpRSA secpR))                            ; l_a
             (PpiEQ-RSA (+ secpPA secpR))                             ; k_a
             (PpiEQ-PA (+ secpPA secpR))))                            ; k_a
     (* 3 (+ (PpiEQ-RSA (+ secpRSA secpPA secpR secpR secpR))         ; t_a',t_0',t_1'
             (PpiEQ-RSA (+ secpPA secpPA secpR secpR secpR))          ; s_a',s_0',s_1'
             (PpiEQ-PA (+ secpPA secpPA secpR secpR secpR))))))       ; s_a'

; Proof of correct conversion.
(define (PpiX)
  (max (PpiXvalid) (PpiXinvalid)))                                    ; Worst case of valid/invalid
(define (VpiX)
  (max (VpiXvalid) (VpiXinvalid)))                                    ; Worst case of valid/invalid

; Proof of validity/invalidity (general method).
(define (PpiVIgeneral thebit thecase)
  (+ (* 5 (RSAexp (+ secpRSA secpR)))                                 ; Form commitments the l
     (RSAexp (+ secpM secpR secpR))                                   ; r'
     (* 2 (RSAexp (- secpP secpR)))                                   ; s_0', s_1'
     (* 2 (RSAexp (+ secpRSA secpR)))                                 ; r_0, r_1
     (RSAexp (+ secpP secpP secpP))                                   ; C_a
     (RSAexp secpP)                                                   ; C_{e_0}
     (RSAexp secpP)                                                   ; C_{e_1}
     (* (- 1 thecase)                                                 ; Signer executes prover
        (+ (PpiEQoverhead)                                            ; Overhead of EQ-protocol
           (RSAexp secpRSA)                                           ; Exponentiation by e
           (* 3 (+ (PpiEQ-form (+ secpPA secpR))                      ; Form commitments of t_a,t_0,t_1
                   (* 3 (PpiEQ-comm (+ secpPA secpR)))                ; Proof about t_a,t_0,t_1
                   (* 3 (PpiEQ-PA (+ secpPA secpR)))))                ; Proof about t_a,t_0,t_1
           (* 2 (PpiEQ-PA (+ secpRSA secpR)))                         ; Proof about r_0 and r_1 (RSA-part below)
           (PpiEQ-PA (+ secpM secpR secpR secpP secpP))))             ; a in the input
     (* thecase
        (+ (PpiVF)                                                    ; this includes EQoverhead
           (* 9 (PpiEQ-PA (+ secpPA secpR)))                          ; Proofs about x,x_0,x_1
           (* 3 (PpiEQ-RSA (+ secpRSA secpR)))))                      ; Proofs about a,r_0,r_1
     (* 7 (PpiEQ-RSA (+ secpRSA secpR)))                              ; all l and r_0 r_1
     (PpiEQ-RSA (+ secpM secpR secpR))                                ; r
     (* 2 (PpiEQ-RSA (- secpP secpR)))                                ; s_0', s_1'
     (* 2 (+ (* 2 (RSAexp (+ secpRSA secpR)))                         ; Form commitments, l_0,l_0'
             (RSAexp (+ secpM secpR secpR secpP))                     ; e_0
             (RSAexp (+ secpM secpR secpR secpP))                     ; e_0
             (RSAexp (+ secpRSA secpR))                               ; j_0
             (RSAexp (+ secpRSA secpR))))                             ; j_0
     (* 2 (+ (PpiEQ-RSA (+ secpRSA secpR))                            ; Proofs, l_0
             (PpiEQ-RSA (+ secpM secpR secpR secpP))                  ; e_0
             (PpiEQ-RSA (+ secpRSA secpR))                            ; l_0'
             (PpiEQ-RSA (+ secpRSA secpR))                            ; j_0
             (PpiEQ-RSA (+ secpRSA secpR))                            ; j_0
             (PpiEQ-RSA (+ secpM secpR secpR secpP))))                ; e_0
     (PpiEQ-RSA (+ secpRSA secpR secpP))                              ; Proofs, l_{e_1}
     (PpiEQ-RSA (+ secpM secpR secpR secpP))                          ; e_1
     (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; l_1''
     (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; j_1'
     (PpiEQ-RSA (+ secpRSA secpR))                                    ; j_1
     (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; j_1'
     (* thebit
        (+ (PpiEQ-RSA (+ secpRSA secpR secpP))                        ; Proofs, l_{e_0}
           (PpiEQ-RSA (+ secpM secpR secpR secpP))                    ; e_0
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))      ; l_0''
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))      ; j_0'
           (PpiEQ-RSA (+ secpRSA secpR))                              ; j_0
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))))    ; j_0'
     (* (- 1 thebit)
        (+ (* 5 (RSAexp (+ secpRSA secpR)))                           ; Form commitments, i_0,i_0',i_0''
           (* 5 (PpiEQ-RSA (+ secpRSA secpR)))                        ; Proofs, i_0,i_0',i_0''
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR                    ; Proofs
                         secpR secpP secpRSA secpR))                  ; k_0
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR
                         secpR secpP secpRSA secpR))                  ; f_0
           (PpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; k_0'
           (PpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; f_0'
           (PpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; f_0'
           (PpiEQ-RSA (+ secpRSA secpR secpM secpR
                         secpR secpP secpRSA secpR))))                ; f_0
     ))
(define (VpiVIgeneral thebit thecase)
  (+ (RSAexp (+ secpP secpP secpP))                                   ; C_a
     (RSAexp secpP)                                                   ; C_{e_0}
     (RSAexp secpP)                                                   ; C_{e_1}
     (* (- 1 thecase)                                                 ; Signer executes prover
        (+ (VpiEQoverhead)                                            ; Overhead of EQ-protocol
           (RSAexp secpRSA)                                           ; Exponentiation by e
           (* 3 (+ (* 3 (VpiEQ-comm (+ secpPA secpR)))                ; Proof about t_a,t_0,t_1
                   (* 3 (VpiEQ-PA (+ secpPA secpR)))))                ; Proof about t_a,t_0,t_1
           (* 2 (VpiEQ-PA (+ secpRSA secpR)))                         ; Proof about r_0 and r_1 (RSA-part below)
           (VpiEQ-PA (+ secpM secpR secpR secpP secpP))))             ; a in the input
     (* thecase
        (+ (VpiVF)                                                    ; this includes EQoverhead
           (* 9 (VpiEQ-PA (+ secpPA secpR)))                          ; Proofs about x,x_0,x_1
           (* 3 (VpiEQ-RSA (+ secpRSA secpR)))))                      ; Proofs about a,r_0,r_1
     (* 7 (VpiEQ-RSA (+ secpRSA secpR)))                              ; all l and r_0 r_1
     (VpiEQ-RSA (+ secpM secpR secpR))                                ; r
     (* 2 (VpiEQ-RSA (- secpP secpR)))                                ; s_0', s_1'
     (* 2 (+ (VpiEQ-RSA (+ secpRSA secpR))                            ; Proofs, l_0
             (VpiEQ-RSA (+ secpM secpR secpR secpP))                  ; e_0
             (VpiEQ-RSA (+ secpRSA secpR))                            ; l_0'
             (VpiEQ-RSA (+ secpRSA secpR))                            ; j_0
             (VpiEQ-RSA (+ secpRSA secpR))                            ; j_0
             (VpiEQ-RSA (+ secpM secpR secpR secpP))))                ; e_0
     (VpiEQ-RSA (+ secpRSA secpR secpP))                              ; Proofs, l_{e_1}
     (VpiEQ-RSA (+ secpM secpR secpR secpP))                          ; e_1
     (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; l_1''
     (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; j_1'
     (VpiEQ-RSA (+ secpRSA secpR))                                    ; j_1
     (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))            ; j_1'
     (* thebit
        (+ (VpiEQ-RSA (+ secpRSA secpR secpP))                        ; Proofs, l_{e_0}
           (VpiEQ-RSA (+ secpM secpR secpR secpP))                    ; e_0
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))      ; l_0''
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))      ; j_0'
           (VpiEQ-RSA (+ secpRSA secpR))                              ; j_0
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR secpR secpP))))    ; j_0'
     (* (- 1 thebit)
        (+ (* 5 (VpiEQ-RSA (+ secpRSA secpR)))                        ; Proofs, i_0,i_0',i_0''
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR                    ; Proofs
                         secpR secpP secpRSA secpR))                  ; k_0
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR
                         secpR secpP secpRSA secpR))                  ; f_0
           (VpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; k_0'
           (VpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; f_0'
           (VpiEQ-RSA (* 2 (+ secpRSA secpR)))                        ; f_0'
           (VpiEQ-RSA (+ secpRSA secpR secpM secpR
                         secpR secpP secpRSA secpR))))                ; f_0
     ))

; Proof of validity/invalidity of confirmer.
(define (PpiDE)
  (max (PpiVIgeneral 0 1) (PpiVIgeneral 1 1)))                        ; Worst case
(define (VpiDE)
  (max (VpiVIgeneral 0 1) (VpiVIgeneral 1 1)))                        ; Worst case

; Proof of validity signer.
(define (PpiEE) (PpiVIgeneral 1 0))
(define (VpiEE) (VpiVIgeneral 1 0))

; Utility function.
(define (spacepad value)
  (cond ((< value 10)
         (display "  ")
         (display value))
        ((and (< value 100) (> value 9))
         (display " ")
         (display value))
        (else (display value))))

; Pretty print all complexities.
(define (all)
  (newline)
  (newline)
  (display "                            Signer Confirmer Verifier")
  (newline)
  (display "Signing                     ")
  (spacepad (round (desSig)))
  (newline)
  (display "Converting                         ")
  (spacepad (round (desConv)))
  (newline)
  (display "Verifying                                    ")
  (spacepad (round (desVf)))
  (newline)
  (display "Well-Formedness                    ")
  (spacepad (round (PpiVF)))
  (display "       ")
  (spacepad (round (VpiVF)))
  (newline)
  (display "Correctness of conversion          ")
  (spacepad (round (PpiX)))
  (display "       ")
  (spacepad (round (VpiX)))
  (newline)
  (display "Validity/Invalidity                ")
  (spacepad (round (PpiDE)))
  (display "       ")
  (spacepad (round (VpiDE)))
  (newline)
  (display "Validity                    ")
  (spacepad (round (PpiEE)))
  (display "              ")
  (spacepad (round (VpiEE)))
  (newline)
  (newline))

(all))))
```