Int J Inf Secur (2004) / Digital Object Identifier (DOI) 10.1007/s10207-004-0041-0

Two-party generation of DSA signatures

Philip MacKenzie[1], **Michael K. Reiter**[2]

[1] Bell Labs, Lucent Technologies, Murray Hill, NJ, USA
e-mail: philmac@research.bell-labs.com
[2] Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: reiter@cmu.edu

Published online: 21 July 2004 –

Abstract. We describe a means of sharing the DSA signature function, so that two parties can efficiently generate a DSA signature with respect to a given public key but neither can alone. We focus on a certain instantiation that allows a proof of security for concurrent execution in the random oracle model and that is very practical. We also briefly outline a variation that requires more rounds of communication but that allows a proof of security for sequential execution without random oracles.

Keywords: Cryptography – Digital signature – Multiparty computation

1 Introduction

In this paper we present an efficient and provably secure protocol by which alice and bob, each holding a share of a DSA [34] private key, can (and must) interact to generate a DSA signature on a given message with respect to the corresponding public key. As noted in previous work on multiparty DSA signature generation (e.g., [10, 22, 35]), a shared generation of DSA signatures tends to be more complicated than a shared generation of many other types of ElGamal-based signatures [15] because (i) a shared secret must be inverted, and (ii) a multiplication must be performed on two shared secrets. One can see this difference by comparing a Harn signature [29] with a DSA signature, say, over parameters $<g, p, q>$, with public/secret key pair $<y(= g^x \bmod p), x>$ and ephemeral public/secret key pair $<r(= g^k \bmod p), k>$. In a Harn signature, one computes

$$s \leftarrow x(\mathsf{h}(m)) - kr \bmod q$$

and returns a signature $<r, s>$, while for a DSA signature, one computes

$$s \leftarrow k^{-1}(\mathsf{h}(m) + xr) \bmod q$$

and returns a signature $<r \bmod q, s>$. Obviously, to compute the DSA signature the ephemeral secret key must be inverted, and the resulting secret value must be multiplied by the secret key. For security, all of these secret values must be shared, and thus inversion and multiplication on shared secrets must be performed. Protocols to perform these operations have tended to be much more complicated than protocols for adding shared secrets.

Of course, protocols for generic secure two-party computation (e.g., [49]) could be used to perform two-party DSA signature generation, but here we explore a more efficient protocol to solve this particular problem. To our knowledge, the protocol we present here is the first practical and provably secure protocol for two-party DSA signature generation. As building blocks it uses a public key encryption scheme with certain useful properties (for which several examples exist) and efficient special-purpose zero-knowledge proofs. The assumptions under which these building blocks are secure are the assumptions required for security of our protocol. For example, by instantiating our protocol with particular constructions, we can achieve a protocol that is provably secure under the decision composite residuosity assumption (DCRA) [41] and the strong RSA assumption [2] when executed sequentially, or one that is provably secure in the random oracle model [7] under the DCRA and strong RSA assumption, even when arbitrarily many instances of the protocol are run concurrently. The former protocol requires eight messages, while the latter protocol requires only four messages (counting initialization in neither case).

Extended abstract appears in *Advances in Cryptology – CRYPTO 2001*, August 2001.

Our interest in two-party DSA signature generation stems from our broader research into techniques by which a device that performs private key operations (signatures or decryptions) in networked applications, and whose local private key is activated with a password or PIN, can be immunized against offline dictionary attacks in case the device is captured [36, 37]. Briefly, we achieve this by involving a remote server in the device's private key computations, essentially sharing the cryptographic computation between the device and the server. Our original work showed how to accomplish this for the case of RSA functions or certain discrete-log-based functions other than DSA, using known techniques for sharing those functions between two parties. The important case of DSA signatures is enabled by the techniques of this paper. Given our practical goals, in this paper we focus on the most efficient (four message, random oracle) version of our protocol, which is quite suitable for use in the context of our system.

2 Related work

Two-party generation of DSA signatures falls into the category of threshold signatures or, more broadly, threshold cryptography. Early work in the field is due to Boyd [6], Desmedt [12], Croft and Harris [8], Frankel [18], and Desmedt and Frankel [14]. Work in threshold cryptography for discrete-log based cryptosystems other than DSA is due to Desmedt and Frankel [14], Hwang [31], Pedersen [43], Harn [29], Park and Kurosawa [42], Herzberg et al. [30], Frankel et al. [19], and Jarecki and Lysyanskaya [32].

Several works have developed techniques directly for shared generation of DSA signatures. Langford [35] presents threshold DSA schemes ensuring unforgeability against one corrupt player out of $n \geq 3$, of t corrupt players out of n for arbitrary $t < n$ under certain restrictions (see below), and of t corrupt players out of $n \geq t^2 + t + 1$. Cerecedo et al. [10] and Gennaro et al. [22] present threshold schemes that prevent t corrupt players out of $n \geq 2t + 1$ from forging, and thus require a majority of correct players. Both of these works further develop *robust* solutions, in which the t corrupted players cannot interfere with the other $n - t$ signing a message, provided that stronger conditions on n and t are met (at least $n \geq 3t + 1$). However, since we consider the two-party case only, robustness is not a goal here.

The only previous proposal that can implement two-party generation of DSA signatures is due to Langford [35, Sect. 5.1], which ensures unforgeability against t corrupt players out of n for an arbitrary $t < n$. This is achieved, however, by using a trusted center to precompute the ephemeral secret key k for each signature and to share $k^{-1} \bmod q$ and $k^{-1}x \bmod q$ among the n parties. That is, this solution circumvents the primary difficulties of sharing DSA signatures – inverting a shared secret and multiplying shared secrets, as discussed in Sect. 1 – by using a trusted center. Recognizing the significant drawbacks of a trusted center, Langford extends this solution by replacing the trusted center with three centers (that protect k^{-1} and $k^{-1}x$ from any one) [35, Sect. 5.2], thereby precluding this solution from being used in the two-party case.

Though our motivating application naturally admits a trusted party for initializing the system (see [36]), our presentation here includes a distributed initialization protocol involving only alice and bob, and no trusted center. Since we are using random oracles in the signature protocol, we will describe an initialization protocol instantiated using random oracles. To achieve provable security, this initialization must be executed in a sequential manner before any signature protocols are executed, even though the signature protocols themselves may be executed concurrently with respect to each other.

3 Preliminaries

Security parameters. Let κ be the main cryptographic security parameter used for, e.g., hash functions and discrete log group orders; a reasonable value today may be $\kappa = 160$. We will also use $\kappa' > \kappa$ as a secondary security parameter for public key modulus size; reasonable values today may be $\kappa' = 1024$ or $\kappa' = 2048$. The value κ' is dependent on κ and is set so that known attacks on public key systems with modulus size κ' are at least as hard as known attacks on hash functions and other brute-force attacks on systems with main security parameter κ. We assume that an appropriate κ' can be computed from κ efficiently.

Signature schemes. A *digital signature scheme* is a triple $(G_{\mathcal{S}}, S, V)$ of algorithms, the first two being probabilistic, and all running in expected polynomial time. $G_{\mathcal{S}}$ takes as input 1^{κ} and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_{\mathcal{S}}(1^{\kappa})$. S takes a message m and a secret key sk as input and outputs a signature σ for m, i.e., $\sigma \leftarrow S_{sk}(m)$. V takes a message m, a public key pk, and a candidate signature σ' for m and returns the bit $b = 1$ if σ' is a valid signature for m, and otherwise returns the bit $b = 0$. That is, $b \leftarrow V_{pk}(m, \sigma')$. Naturally, if $\sigma \leftarrow S_{sk}(m)$, then $V_{pk}(m, \sigma) = 1$.

DSA. The Digital Signature Algorithm [34] was proposed by NIST in April 1991 and in May 1994 was adopted as a standard digital signature scheme in the U.S. [17]. It is a variant of the ElGamal signature scheme [15] and is defined as follows, with $\kappa = 160$, κ' set to a multiple of 64 between 512 and 1024, inclusive, and hash function h defined as SHA-1 [16].[1] Let "$z \stackrel{R}{\leftarrow} S$" denote the assignment to z of an element of S

[1] Although κ is fixed for the DSA standard, we will still use it as if it were a varying security parameter.

selected uniformly at random. Let $\equiv_q$ denote equivalence modulo q.

$G_{\mathrm{DSA}}(1^\kappa)$: Generate a κ-bit prime q and κ'-bit prime p such that q divides $p-1$. Then generate an element g of order q in $\mathbb{Z}_p^*$. The triple $<g,p,q>$ is public. Finally, generate $x \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and $y \leftarrow g^x \bmod p$, and let $<g,p,q,x>$ and $<g,p,q,y>$ be the secret and public keys, respectively.

$S_{<g,p,q,x>}(m)$: Generate an ephemeral secret key $k \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and ephemeral public key $r \leftarrow g^k \bmod p$. Compute $s \leftarrow k^{-1}(\mathsf{h}(m) + xr) \bmod q$. Return $<r \bmod q, s>$ as the signature of m.

$V_{<g,p,q,y>}(m, <r', s>)$: Return 1 if $0 < r' < q$, $0 < s < q$, and $r' \equiv_q (g^{\mathsf{h}(m)s^{-1}} y^{r's^{-1}} \bmod p)$, where s^{-1} is computed modulo q. Otherwise, return 0.

Zero-knowledge proofs. Our protocols employ a variety of noninteractive zero-knowledge (NIZK) proofs [4]. Here we define their security under the random oracle assumption. Our definitions for NIZK proofs are based on definitions from [3] and [13]. Notice, however, that we only require standard soundness, rather than simulation soundness [13].

For a relation R, let $L_R = \{w : (w,v) \in R\}$ be the *language* defined by the relation, and for all $w \in L_R$, let $W_R(w) = \{v : (w,v) \in R\}$ be the *witness set* for w. For any NP language L, note that there is a natural *witness relation* R containing pairs (w,v), where v is the witness for the membership of w in L, and that $L_R = L$. Recall that κ is a security parameter. Let $\mathcal{H}$ be the set of all hash functions with κ-bit outputs.

Let $\mathcal{X} = \{X_\kappa\}_{\kappa \geq 1}$ and $\mathcal{Y} = \{Y_\kappa\}_{\kappa \geq 1}$ be two probability distribution ensembles. We define the *distinguishing probability* of $\mathcal{X}$ and $\mathcal{Y}$ as $\epsilon(\kappa) = \sum_\alpha |\Pr(X_\kappa = \alpha) - \Pr(Y_\kappa = \alpha)|$.

We denote by $\mathcal{H}$ the set of all functions hash from $\{0,1\}^*$ to $\{0,1\}^\infty$.

– **Zero-knowledge proofs**: A noninteractive zero-knowledge proof system with initialization (NIZKPI system) Ψ for an NP language L, with witness relation R, is a tuple $(\mathcal{I}, \mathcal{P}, \mathcal{V}, \mathsf{Sim})$, where $\mathcal{I}$ and $\mathcal{P}$ are probabilistic polynomial-time algorithms, $\mathcal{V}$ is a deterministic polynomial-time algorithm, and Sim is a probabilistic polynomial-time protocol for performing initialization, answering hash queries and answering $\mathcal{P}$ queries, denoted by $\mathsf{Siminit}$, $\mathsf{Simhash}$, and $\mathsf{Simprove}$,[2] respectively, satisfying:

1. Completeness: For all $(w,v) \in R$, for all $\mathsf{hash} \in \mathcal{H}$, for all $I \leftarrow \mathcal{I}(1^\kappa)$, $\mathcal{V}^{\mathsf{hash}}(I, w, \mathcal{P}^{\mathsf{hash}}(I,w,v))$ returns TRUE.
2. Soundness: There is a function $\textsc{Serr}(\kappa, t, n_{\mathrm{ro}})$ (*soundness error*) such that for all probabilistic adversaries $\mathcal{A}$ that run in time t, and make at most n_{ro} hash queries, $\Pr[\mathsf{Expt}_{\mathcal{A},\Psi}(\kappa)] \leq \textsc{Serr}(\kappa, t, n_{\mathrm{ro}})$, where experiment $\mathsf{Expt}_{\mathcal{A},\Psi}(\kappa)$ is defined as follows:

$\mathsf{Expt}_{\mathcal{A},\Psi}(\kappa)$:
$\mathsf{hash} \stackrel{R}{\leftarrow} \mathcal{H}$
$I \stackrel{R}{\leftarrow} \mathcal{I}(1^\kappa)$
$(w,\sigma) \leftarrow \mathcal{A}^{\mathsf{hash}}(I)$
Return TRUE iff $(w \notin L \wedge \mathcal{V}^{\mathsf{hash}}(I,w,\sigma) = \mathrm{TRUE})$

If this experiment returns TRUE for a certain σ, we call σ a *fraudulent proof*.

3. Unbounded statistical zero-knowledge: There is a function $\textsc{Simerr}(\kappa, n_{\mathrm{ro}}, n_{\mathrm{pr}})$ (*simulation error*) such that

$$\max_{\mathcal{A}} |\Pr[\mathsf{Expt}'_{\mathcal{A},\Psi}(\kappa) = 1] - \Pr[\mathsf{Expt}''_{\mathcal{A},\Psi}(\kappa) = 1]| \leq \textsc{Simerr}(\kappa, n_{\mathrm{ro}}, n_{\mathrm{pr}}),$$

where the maximum is over all (unbounded time) adversaries $\mathcal{A}$ that make at most n_{ro} hash queries and n_{pr} $\mathcal{P}$ queries, and where experiments $\mathsf{Expt}'_{\mathcal{A},\Psi}(\kappa)$ and $\mathsf{Expt}''_{\mathcal{A},\Psi}(\kappa)$ are defined as follows:

$\mathsf{Expt}'_{\mathcal{A},\Psi}(\kappa)$:	$\mathsf{Expt}''_{\mathcal{A},\Psi}(\kappa)$:
$\mathsf{hash} \stackrel{R}{\leftarrow} \mathcal{H}$	$I \stackrel{R}{\leftarrow} \mathsf{Siminit}(1^\kappa)$
$I \stackrel{R}{\leftarrow} \mathcal{I}(1^\kappa)$	Return $\mathcal{A}^{\mathsf{Simhash},\mathsf{Sim}'(I,\cdot,\cdot)}(I)$
Return $\mathcal{A}^{\mathsf{hash},\mathcal{P}^{\mathsf{hash}}(I,\cdot,\cdot)}(I)$	

where $\mathsf{Sim}'(I,w,v) \stackrel{\mathrm{def}}{=} \mathsf{Simprove}(I,w)$ for $(w,v) \in R$. [If $(w,v) \notin R$, we may assume that both $\mathcal{P}^{\mathsf{hash}}(I,w,v)$ and $\mathsf{Sim}'(I,w,v)$ abort, though we do require that they halt in polynomial time irrespective of whether $(w,v) \in R$.]

In our protocols, we denote a zero-knowledge proof that a predicate P holds on a given input w (i.e., that w is in the language of elements satisfying P) by $\mathsf{zkp}\,[P]$.[3]

Encryption schemes. An *encryption scheme* $\mathcal{E}$ is a triple $(G_\mathcal{E}, E, D)$ of algorithms, the first two being probabilistic, and all running in expected polynomial time. $G_\mathcal{E}$ takes as input 1^κ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_\mathcal{E}(1^\kappa)$. E takes a public key pk and a message m as input and outputs an encryption c for m; we denote this $c \leftarrow E_{pk}(m)$. D takes a ciphertext c and a secret key sk as input and returns either a message m such that c is a valid encryption of m under the corresponding public key, if such an m exists, and otherwise returns an arbitrary value.

We require the encryption scheme we use to be *matchable* in the following sense: There exists an efficiently

[2] We may assume that $\mathsf{Simhash}$ is given a polynomial-size input and a polynomial-size output length, since it obviously could not output an infinite number of bits in polynomial time.

[3] The input w will be implicit in the definition of P and thus is not included separately in this notation.

computable predicate $M(pk, sk)$ that returns 1 if and only if (pk, sk) could possibly be output from $G_{\mathcal{E}}(1^{\kappa})$. $M(pk, sk)$ implies not only that sk is the matching private key for pk, but also that both are well formed according to the key generation algorithm. For the example cryptosystem we adopt here (described in Sect. 6.1), we describe how to implement a zero-knowledge proof of knowledge of sk such that $M(pk, sk)$ holds for a public pk. This proof is required in the initialization protocol for our system (Sect. 4.1).

Our protocol additionally requires that encryption be semantically secure and have a certain additive homomorphic property. For any public key pk output from the $G_{\mathcal{E}}$ function, let M_{pk} be the space of possible inputs to E_{pk} and C_{pk} the space of possible outputs of E_{pk}. Then we require that there exist an efficient implementation of an additional function $+_{pk} : C_{pk} \times C_{pk} \to C_{pk}$ such that (written as an infix operator)

$$m_1, m_2, m_1 + m_2 \in M_{pk} \Rightarrow \\ D_{sk}(E_{pk}(m_1) +_{pk} E_{pk}(m_2)) = m_1 + m_2 \,. \quad (1)$$

Examples of cryptosystems for which the function $+_{pk}$ exists (with $M_{pk} = [-v, v]$ for a certain value v) are due to Naccache and Stern [38], Okamoto and Uchiyama [40], and Paillier [41].[4] Note that Eq. (1) further implies the existence of an efficient function $\times_{pk} : C_{pk} \times M_{pk} \to C_{pk}$ such that

$$m_1, m_2, m_1 m_2 \in M_{pk} \Rightarrow D_{sk}(E_{pk}(m_1) \times_{pk} m_2) = m_1 m_2 \,. \quad (2)$$

In addition, in our protocol, a party may be required to generate a noninteractive zero-knowledge proof of a certain predicate P involving decryptions of elements of C_{pk}, among other things. In Sect. 6.1, we show how these proofs can be accomplished if the Paillier cryptosystem is in use. We emphasize, however, that our use of the Paillier cryptosystem is only exemplary; the other cryptosystems cited above could equally well be used with our protocol.

System model. Our system includes two parties, alice and bob. Each must execute an initialization protocol (in a sequential manner) before any signature protocols are executed. After initialization, communication between alice and bob occurs in *sessions* (or signature protocol runs), one per message that they sign together. alice plays the role of session initiator in our signature protocol. We presume that each message is implicitly labeled with an identifier for the session to which it belongs. Multiple signing sessions can be executed concurrently.

The adversary in our protocol controls the network, inserting and manipulating communication as it chooses. In addition, it takes one of two forms: an alice-compromising adversary that has read access to the private storage of alice and a bob-compromising adversary that has read access to the private storage of bob. Without loss of generality, we assume that an alice-compromising adversary takes the place of alice in interactions with bob, and does so starting from the beginning of system execution. We make the analogous assumption for a bob-compromising adversary.

Informally, the goal of an alice-compromising adversary is to generate a signature on a message that bob did not cooperate to sign, either because he was not asked or because he refused. The goal of a bob-compromising adversary is analagous. Our protocol is secure if it ensures that any correctly signed message was generated with the cooperation of both alice and bob for that message. Our security goals do not include fairness (e.g., if one obtains a signature, then the other must as well) or robustness (e.g., if one misbehaves, then the other can prove this is the case to others).

We note that a proof of security in this two-party system extends to a proof of security in an n-party system in a natural way, assuming the adversary decides which parties to compromise before any session begins. The basic idea is to guess for which pair of parties the adversary forges a signature and focus the simulation proof on those two parties, running all other parties as in the real protocol. The only consequence is a factor of roughly n^2 lost in the reduction argument from the security of the signature scheme.

4 S-DSA system

In this section we present a new system called S-DSA by which alice and bob can jointly create DSA signatures on messages.

4.1 Initialization

Our signature protocol of Sect. 4.2 requires an initialization in which the following properties are achieved:

I1. A DSA public key pair is generated ($<g, p, q, y>$, $<g, p, q, x>$).

I2. The private key x is multiplicatively shared between alice and bob, so that alice holds a random private value $x_1 \in \mathbb{Z}_q$ and bob holds a random private value $x_2 \in \mathbb{Z}_q$ such that $x \equiv_q x_1 x_2$. Along with y, $y_1 = g^{x_1} \bmod p$ and $y_2 = g^{x_2} \bmod p$ are public.

I3. alice holds the private key sk corresponding to a public encryption key pk, and there is another public encryption key pk' for which alice does not know the corresponding sk'.

Here we assume this initialization is performed by a trusted third party. However, since avoiding a trusted third party is preferable, in Appendix A we describe an initialization protocol for achieving the properties above.

Two comments about properties I2 and I3 are in order. Regarding I2, we use a multiplicative sharing of x

[4] The cryptosystem of Benaloh [1] also has this additive homomorphic property and thus could also be used in our protocol. However, it would be less efficient for our purposes.

to achieve greater efficiency than can be achieved by using either polynomial sharing or additive sharing. With multiplicative sharing of keys, inversion and multiplication of shared keys becomes trivial, but addition of shared keys becomes more complicated. For DSA, however, this approach seems to allow a much more efficient two-party protocol. Regarding I3, it is necessary for our particular zero-knowledge proof constructions described in Sect. 6

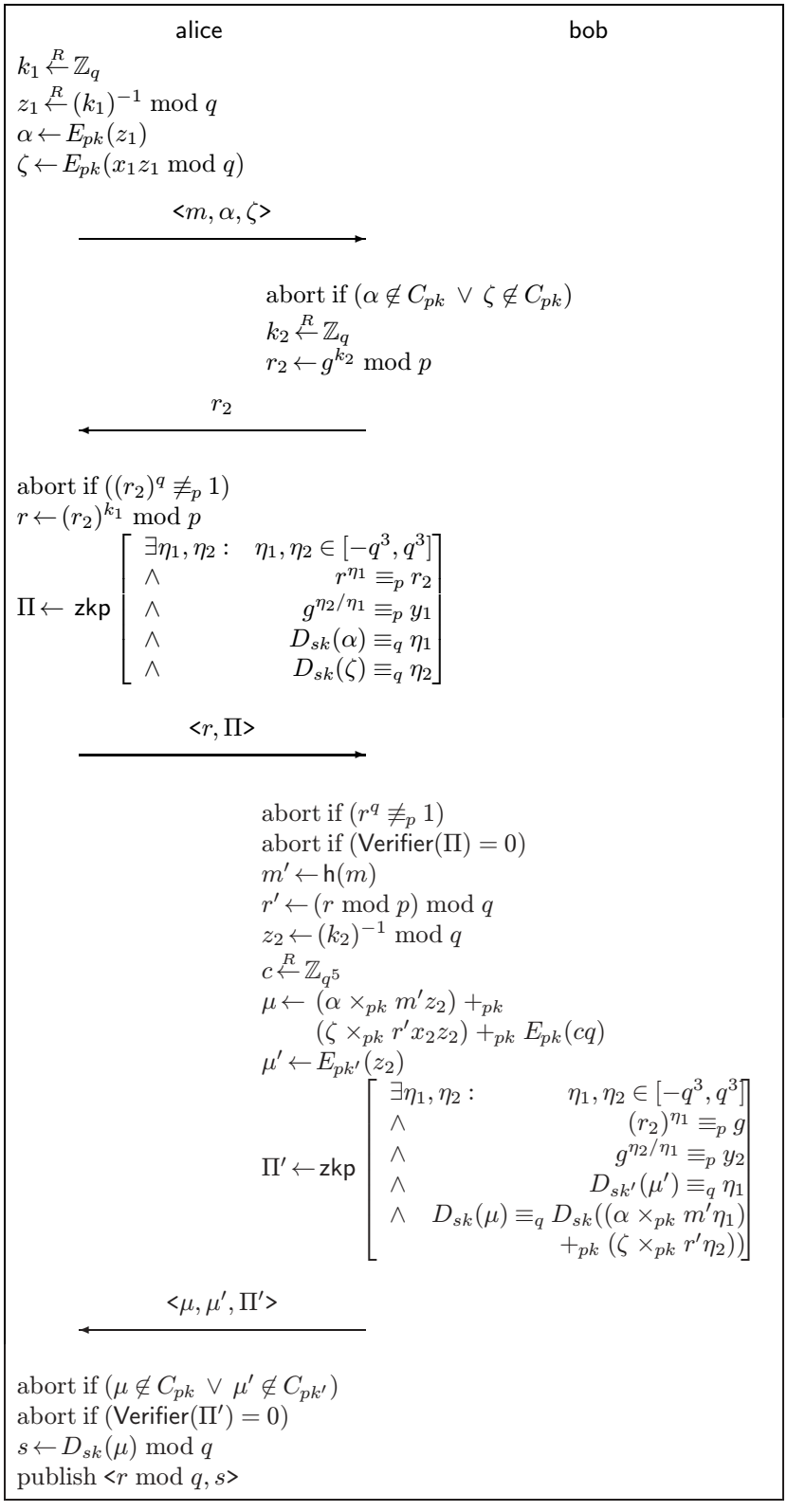

Fig. 1. S-DSA shared signature protocol

that the range of M_{pk} be at least $[-q^8, q^8]$ and the range of $M_{pk'}$ be at least $[-q^6, q^6]$, although we believe a slightly tighter analysis would allow both to have a range of $[-q^6, q^6]$.

4.2 Signing protocol

The protocol by which alice and bob cooperate to generate signatures with respect to the public key $<g, p, q, y>$ is shown in Fig. 1. As input to this protocol alice receives the message m to be signed. bob receives no input (but receives m from alice in the first message).

At a high level, the protocol is broken into two "phases", each consisting of one message in each direction. The goal of the first phase is to establish an ephemeral private key $k \in \mathbb{Z}_q$ and public key $r = g^k \bmod p$, where k is shared as $k = k_1 k_2 \bmod q$, with k_1 a secret known by alice and k_2 a secret known to bob. In addition, alice commits to $(k_1)^{-1} \bmod q$ by sending its encryption α under pk and to $(k_1)^{-1} x_1 \bmod q$ by sending its encryption ζ under pk. In the second phase, bob uses these commitments together with $+_{pk}$ and $\times_{pk}$ to form the encryption of the s component of the signature under pk (without ever decrypting α or ζ, which it cannot do).

More specifically, the protocol works as follows. Upon receiving m to sign, alice first computes its share k_1 of the ephemeral private key for this signature, computes $z_1 = (k_1)^{-1} \bmod q$, and encrypts both z_1 and $x_1 z_1 \bmod q$ under pk. alice's first message to bob consists of m and these ciphertexts, α and ζ. bob performs some simple consistency checks on α and ζ (though he cannot decrypt them, since he does not have sk), generates his share k_2 of the ephemeral private key, and returns his share $r_2 = g^{k_2} \bmod p$ of the ephemeral public key.

Once alice has received r_2 from bob and performed simple consistency checks on it (e.g., to determine it has order q in $\mathbb{Z}_p^*$), she is able to compute the ephemeral public key $r = (r_2)^{k_1} \bmod p$, which she sends to bob in the third message of the protocol. alice also sends a noninteractive zero-knowledge proof Π that there are values η_1 $(= z_1)$ and η_2 $(= x_1 z_1 \bmod q)$ that are consistent with r, r_2, y_1, α, and ζ and that are in the range $[-q^3, q^3]$. This last fact is necessary so that bob's subsequent formation of (a ciphertext of) s does not leak information about his private values.

Upon receiving $<r, \Pi>$, bob verifies Π and performs additional consistency checks on r. If these pass, then he proceeds to compute a ciphertext μ of the value s (modulo q) for the signature, using the ciphertexts α and ζ received in the first message from alice; the values $\mathsf{h}(m)$, $z_2 = (k_2)^{-1} \bmod q$, $r \bmod q$, and x_2; and the special $\times_{pk}$ and $+_{pk}$ operators of the encryption scheme. In addition, bob uses $+_{pk}$ to "blind" the plaintext value with a random, large multiple of q. So when alice later decrypts μ, she statistically gains no information about bob's private values. In addition to returning μ, bob computes and returns $\mu' \leftarrow E_{pk'}(z_2)$ and a noninteractive zero-knowledge proof Π' that there are values η_1 $(= z_2)$ and η_2 $(= x_2 z_2 \bmod p)$ consistent with r_2, y_2, μ, and μ' and that are in the range $[-q^3, q^3]$. After receiving and checking these values, alice recovers s from μ to complete the signature.

The noninteractive zero-knowledge proofs Π and Π' are assumed to satisfy the completeness, soundness, and zero-knowledge properties as defined in Sect. 3. The implementations of Π and Π' in Sect. 6 enforce these properties under reasonable assumptions. To instantiate this protocol without random oracles, Π and Π' would need to become interactive zero-knowledge protocols. It is not too difficult to construct four-move protocols for Π and Π', and by overlapping some messages, one can reduce the total number of moves in this instantiation of the S-DSA signature protocol to eight. For brevity, we omit the full description of this instantiation.

When the zero-knowledge proofs are implemented using random oracles, we can show that our protocol is secure even when multiple signing instances are executed concurrently. Perhaps the key technical aspect is that we only require proofs of language membership, which can be implemented using random oracles without requiring rewinding in the simulation proof. In particular, we avoid the need for any proofs of knowledge that would require rewinding in knowledge extractors for the simulation proof, even if random oracles are used. The need for rewinding (and, particularly, nested rewinding) causes many proofs of security to fail in the concurrent setting (e.g., [33]).

5 Security for S-DSA

In this section we provide a formal proof of security for the S-DSA system. We begin by defining security for signatures and encryption in Sect. 5.1 and for S-DSA in Sect. 5.2. We then state our theorems and proofs in Sect. 5.3.

5.1 Security for DSA and encryption

Security for signature schemes. We specify existential unforgeability versus chosen message attacks [28] for a signature scheme $\mathcal{S} = (G_{\mathcal{S}}, S, V)$. A forger $\mathcal{F}$ is given pk, where $(pk, sk) \leftarrow G_{\mathcal{S}}(1^\kappa)$, and tries to forge signatures with respect to pk. It is allowed to query a signature oracle (with respect to sk) on messages of its choice. It succeeds if after this it can output a valid forgery (m, σ) such that $V_{pk}(m, \sigma) = 1$, where m was not one of the messages signed by the signature oracle. We say $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(\mathcal{F}) = \Pr(\mathcal{F} \text{ succeeds})$, and $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(t, u) = \max_{\mathcal{F}} \left\{ \mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(\mathcal{F}) \right\}$, where the maximum is taken over all forgers of time complexity t that make u queries to the signature oracle.

Security for encryption schemes. We specify semantic security [27]. An attacker $\mathcal{A}$ is given pk, where

$(pk, sk) \leftarrow G_{\mathcal{E}}(1^\kappa)$. $\mathcal{A}$ generates two equal-length strings X_0 and X_1 and sends these to a test oracle, which chooses $b \stackrel{R}{\leftarrow} \{0,1\}$, and returns $Y = E_{pk}(X_b)$. Finally, $\mathcal{A}$ outputs b' and succeeds if $b' = b$. Let $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{A}) = 2 \cdot \Pr(\mathcal{A} \text{ succeeds}) - 1$. Note that this implies $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{A}) = \Pr(\mathcal{A} \text{ guesses } 1 | b = 1) - \Pr(\mathcal{A} \text{ guesses } 1 | b = 0)$. Let $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{A}) \right\}$, where the maximum is taken over all adversaries of time complexity t.

5.2 Security for S-DSA

Our security definition for S-DSA is similar to our security definition for signature schemes above, except that the signature oracle is replaced by an alice or bob oracle, as described below.

A forger $\mathcal{F}$ begins with DSA parameters $<g, p, q>$, the ability to invoke queries on alice or bob "oracles", and, as described in Sect. 3, entire control of the network between alice and bob. $\mathcal{F}$ also receives the public key y, the public shares y_1 of alice and the public share y_2 of bob, and the public keys pk and pk' belonging to alice and bob, respectively. An alice-compromising forger also receives x_1 and sk and thus has the ability to faithfully execute the protocol of alice, while a bob-compromising forger also receives x_2 and sk' and thus has the ability to faithfully execute the protocol of bob. The goal of $\mathcal{F}$ is to forge a signature with respect to $<g, p, q, y>$. Instead of a signature oracle, there is an alice oracle or a bob oracle.

A bob-compromising $\mathcal{F}$ may query alice by invoking $aInv1(m)$, $aInv2(r_2)$, or $aInv3(<\mu, \mu', \Pi'>)$ for input parameters of $\mathcal{F}$'s choosing. The queries $aInv1(m)$, $aInv2(r_2)$, and $aInv3(<\mu, \mu', \Pi'>)$ correspond to a request to initiate the protocol of Fig. 1 for message m and the first and second messages received ostensibly from bob in this protocol, respectively. (We assume that these invocations are also accompanied by a session identifier, which is left implicit.) These return outputs of the form $<m, \alpha, \zeta>$, $<r, \Pi>$ or a signature for the message m from the previous $aInv1$ query in the same session, respectively, or abort.

The queries $bInv1(<m, \alpha, \zeta>)$ and $bInv2(<r, \Pi>)$ are defined analogously for the bob oracle and can be invoked by an alice-compromising forger.

An alice-compromising forger $\mathcal{F}$ *succeeds* if after invoking the bob oracle as it chooses it can output (m, σ), where $V_{<g,p,q,y>}(m, \sigma) = 1$ and m is not one of the messages sent to bob in a $bInv1$ query. Similarly, a bob-compromising forger $\mathcal{F}$ *succeeds* if after invoking the alice oracle as it chooses it can output (m, σ), where $V_{<g,p,q,y>}(m, \sigma) = 1$ and m is not one of the messages sent to alice in an $aInv1$ query.

Let q_{alice} be the number of $aInv1$ queries to alice, which we take to be zero for an alice-compromising forger. Let q_{bob} be the number of $bInv1$ queries; similarly, this is zero for a bob-compromising forger. Let q_{hash_Π} denote the number of queries to the random oracle associated with Π, and let $q_{\mathsf{hash}_{\Pi'}}$ denote the number of queries to the random oracle associated with Π'. We say $\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA},\kappa}(\mathcal{F}) = \Pr(\mathcal{F} \text{ succeeds})$.

5.3 Theorems

Here we state theorems and provide proofs that relate the probability with which a forger can break the S-DSA system to the probability that either DSA, the underlying encryption scheme, or the zero-knowledge proofs used in S-DSA can be broken. This implies that if DSA, the underlying encryption scheme, and the zero-knowledge proofs are secure, our system will be secure.

The idea behind each proof is to construct a series of systems S-DSA$_0$, S-DSA$_1$, ..., related to S-DSA, with S-DSA$_0$ = S-DSA, and such that we eventually come to a system S-DSA$_i$ such that breaking S-DSA$_i$ implies breaking the original DSA signature scheme. We then show that for any attacker, the difference in the advantage of the attacker in breaking S-DSA$_{i-1}$ and S-DSA$_i$ is related to the probability of breaking the underlying encryption scheme or breaking the soundness of the zero-knowledge proof.

In the theorem statement below, let t_{exp} denote the time required for a modular exponentiation with an exponent and modulus of κ' bits.

Theorem 1. *Fix an* alice*-compromising forger $\mathcal{F}$ that runs in time t. Then for $t' = O(t + q_{bob} t_{exp})$*

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA},\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{DSA},\kappa}(t', q_{\mathsf{bob}}) + \textsc{Serr}_{\Pi}(\kappa, t', q_{\mathsf{hash}_\Pi}) + \textsc{Simerr}_{\Pi'}(\kappa, q_{\mathsf{hash}_{\Pi'}}, q_{\mathsf{bob}}) + q_{\mathsf{bob}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t') + 8(2^{-\kappa}).$$

Proof. Let S-DSA$_1$ be the S-DSA$_0$ system, except that in response to a $bInv2(<r, \Pi>)$ query, run Sim to produce a simulated Π'. Then

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_0,\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_1,\kappa}(\mathcal{F}) + \textsc{Simerr}_{\Pi'}(\kappa, q_{\mathsf{hash}_{\Pi'}}, q_{\mathsf{bob}}).$$

Let S-DSA$_2$ be the S-DSA$_1$ system, except that in response to a query $bInv2(<r, \Pi>)$, set $\mu' \leftarrow E_{pk'}(0)$. Note that we still run Sim to produce a simulated Π'. Then

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_1,\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) + q_{\mathsf{bob}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t').$$

To see this, let $\mathcal{D}$ be an algorithm that takes a public key pk' and a test oracle as input, chooses $j \stackrel{R}{\leftarrow} \{1, \ldots, q_{\mathsf{bob}}\}$, and runs S-DSA$_1$ using pk' as the public encryption key of bob, with the following modifications. $\mathcal{D}$ computes the first $j-1$ ciphertexts by bob under the key pk' as normal, i.e., as $E_{pk'}(z_2)$. $\mathcal{D}$ computes the j-th ciphertext using the response from the test oracle with inputs $X_0 = 0$ and $X_1 = z_2$. $\mathcal{D}$ computes the remaining ciphertexts as $E_{pk'}(0)$. When the simulation completes, $\mathcal{D}$ outputs 1 if $\mathcal{F}$ produces a forgery and 0 otherwise. Note that the case $j = 1$ with the test oracle bit equal to 0 corresponds to

S-DSA$_2$ and the case $j = q_{\mathsf{bob}}$ with the test oracle bit equal to 1 corresponds to S-DSA$_1$. Let $\mathcal{D}_j$ denote $\mathcal{D}$ conditioned on the choice of j and $\mathcal{D}_{j,b}$ denote $\mathcal{D}_j$ conditioned on the bit choice b of the test oracle. Let $\overline{\mathcal{D}}_{j,b}$ denote the simulation run by $\mathcal{D}_{j,b}$. Then, noting that for $1 \le j \le q_{\mathsf{bob}} - 1$, $\overline{\mathcal{D}}_{j,1}$ is perfectly indistinguishable from $\overline{\mathcal{D}}_{j+1,0}$, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t') &\ge \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{D}) \\
&= \frac{1}{q_{\mathsf{bob}}} \sum_{j=1}^{q_{\mathsf{bob}}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{D}_j) \\
&= \frac{1}{q_{\mathsf{bob}}} \sum_{j=1}^{q_{\mathsf{bob}}} (\Pr(\mathcal{D}_j \text{ outputs } 1 | b = 1) \\
&\qquad - \Pr(\mathcal{D}_j \text{ outputs } 1 | b = 0)) \\
&= \frac{1}{q_{\mathsf{bob}}} \sum_{j=1}^{q_{\mathsf{bob}}} \left(\mathsf{Succ}^{\text{eu-cma}}_{\overline{\mathcal{D}}_{j,1},\kappa}(\mathcal{F}) - \mathsf{Succ}^{\text{eu-cma}}_{\overline{\mathcal{D}}_{j,0},\kappa}(\mathcal{F}) \right) \\
&= \frac{1}{q_{\mathsf{bob}}} \left(\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_1,\kappa}(\mathcal{F}) - \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) \right).
\end{aligned}$$

Let S-DSA$_3$ be the S-DSA$_2$ system, except that in the initialization, the secret key sk corresponding to public key pk is recorded, and in response to a $bInv2(\langle r, \Pi \rangle)$ query, if Π is a fraudulent proof (i.e., a valid proof for a string not satisfying the predicate), S-DSA$_3$ aborts. (Note that we assume simulating Π' has no effect on the soundness of Π. In our instantiations of these protocols, this will be true due to the fact that Π and Π' use different random oracles.) Then

$$\begin{aligned}
&\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) \\
&\quad \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) + \textsc{Serr}_{\Pi}(\kappa, t', q_{\mathsf{hash}_\Pi}).
\end{aligned}$$

Now we show that

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{DSA},\kappa}(t', q_{\mathsf{bob}}) + 8(2^{-\kappa}).$$

To see this, let $\mathcal{D}$ be an algorithm that takes a DSA public key $\langle g, p, q, y \rangle$ generated by $G_{\mathrm{DSA}}(1^\kappa)$ and its corresponding signature oracle and runs S-DSA$_3$ with the following modifications. In the initialization, $\mathcal{D}$ chooses $x_1 \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and computes $y_1 \leftarrow g^{x_1} \bmod p$ and $y_2 \leftarrow y^{1/x_1} \bmod p$. In response to a $bInv1(\langle m, \alpha, \zeta \rangle)$ query, $\mathcal{D}$ computes $z_1 \leftarrow D_{sk}(\alpha)$. (Recall that $\mathcal{D}$ has stored the secret key sk from initialization in S-DSA$_3$.) Then $\mathcal{D}$ queries the DSA signature oracle with m to get a signature $\langle \hat{r}, \hat{s} \rangle$, and computes $r \leftarrow g^{\mathsf{h}(m)\hat{s}^{-1}} y^{\hat{r}\hat{s}^{-1}} \bmod p$, where $\hat{s}^{-1}$ is computed modulo q. Finally, $\mathcal{D}$ computes $r_2 \leftarrow r^{z_1} \bmod p$ and returns r_2. In response to a $bInv2(\langle r, \Pi \rangle)$ query, $\mathcal{D}$ computes μ by choosing $c \stackrel{R}{\leftarrow} \mathbb{Z}_{q^5}$ and then setting $\mu \leftarrow E_{pk}(\hat{s} + qc)$. Note that the distinguishing probability of $\mathcal{D}$ and S-DSA$_3$ is bounded by $\frac{4}{q}$ due to the different way μ is computed. While the plaintext would be equivalent modulo q in either case, the multiple of q comes from a slightly different range. In S-DSA$_3$, μ would be an encryption of $\hat{s} + qc' + qc$ for some $c' \in [-2q^4, 2q^4]$ and random $c \in \mathbb{Z}_{q^5}$. Thus the distinguishing probability is bounded by $\frac{2(2q^4)}{q^5} = \frac{4}{q} \le 8(2^{-\kappa})$.

Theorem 2. *Fix a* bob*-compromising forger $\mathcal{F}$ that runs in time t. Then for $t' = O(t + q_{\mathsf{alice}} t_{\mathsf{exp}})$*

$$\begin{aligned}
&\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA},\kappa}(\mathcal{F}) \\
&\quad \le \mathsf{Succ}^{\text{eu-cma}}_{\text{DSA},\kappa}(t', q_{\mathsf{alice}}) + \textsc{Serr}_{\Pi'}(\kappa, t', q_{\mathsf{hash}_{\Pi'}}) + \\
&\qquad \textsc{Simerr}_{\Pi}(\kappa, q_{\mathsf{hash}_\Pi}, q_{\mathsf{alice}}) + 2q_{\mathsf{alice}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t').
\end{aligned}$$

Proof. Let S-DSA$_1$ be the S-DSA$_0$ system, except that in response to a $aInv2(r_2)$ query, run Sim to produce a simulated Π. Then

$$\begin{aligned}
&\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_0,\kappa}(\mathcal{F}) \\
&\quad \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_1,\kappa}(\mathcal{F}) + \textsc{Simerr}_{\Pi}(\kappa, q_{\mathsf{hash}_\Pi}, q_{\mathsf{alice}}).
\end{aligned}$$

Let S-DSA$_2$ be the S-DSA$_1$ system, except that in the initialization, the secret key sk' (corresponding to public key pk') and the DSA secret key $\langle g, p, q, x \rangle$ are recorded. In response to an $aInv3(\mu, \mu', \Pi')$ query corresponding to an $aInv1(m)$ query, compute $z_2 \leftarrow D_{sk'}(\mu')$, $k_2 \leftarrow (z_2)^{-1} \bmod q$, and $k \leftarrow k_1 k_2 \bmod q$. Then return $\langle r \bmod q, s \rangle$, where $s = k^{-1}(h(m) + xr) \bmod q$. As long as Π' is not a fraudulent proof (i.e., a valid proof for a string not satisfying the predicate), this response in S-DSA$_2$ is exactly the same as the response would be in S-DSA$_1$. (Note that we assume simulating Π has no effect on the soundness of Π'. In our instantiations of these protocols, this will be true due to the fact that Π and Π' use different random oracles.) Then

$$\begin{aligned}
&\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) \\
&\quad \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) + \textsc{Serr}_{\Pi'}(\kappa, t', q_{\mathsf{hash}_{\Pi'}}).
\end{aligned}$$

Let S-DSA$_3$ be the S-DSA$_2$ system, except that, in response to an $aInv1(m)$ query, set $\alpha \leftarrow E_{pk}(0)$ and $\zeta \leftarrow E_{pk}(0)$. Note that we still run Sim to produce a simulated Π. Then

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) + 2q_{\mathsf{alice}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t').$$

To see this, let $\mathcal{D}$ be an algorithm that takes a public key pk and a test oracle as input, chooses $j \stackrel{R}{\leftarrow} \{1, \ldots, 2q_{\mathsf{alice}}\}$, and runs S-DSA$_2$ using pk as the public encryption key of alice, with the following modifications. $\mathcal{D}$ computes the first $j-1$ ciphertexts by alice under the key pk as normal, i.e., as $E_{pk}(z_1)$ and $E_{pk}(x_1 z_1 \bmod q)$. $\mathcal{D}$ computes the j-th ciphertext using the response from the test oracle with inputs $X_0 = 0$ and either $X_1 = z_1$ if j is odd or $X_1 = x_1 z_1 \bmod q$ if j is even. $\mathcal{D}$ computes the remaining ciphertexts as $E_{pk}(0)$. When the simulation completes, $\mathcal{D}$ outputs 1 if $\mathcal{F}$ produces a forgery and 0 otherwise. Note that the case $j = 1$ with the test oracle bit equal to 0 corresponds to S-DSA$_3$ and the case $j = 2q_{\mathsf{alice}}$ with the test oracle bit equal to 1 corresponds to S-DSA$_2$. Let $\mathcal{D}_j$ denote $\mathcal{D}$ conditioned on the choice of j and $\mathcal{D}_{j,b}$ denote $\mathcal{D}_j$

conditioned on the bit choice b of the test oracle. Let $\overline{\mathcal{D}}_{j,b}$ denote the simulation run by $\mathcal{D}_{j,b}$. Then noting that, for $1 \le j \le 2q_{\mathsf{alice}} - 1$, $\overline{\mathcal{D}}_{j,1}$ is perfectly indistinguishable from $\overline{\mathcal{D}}_{j+1,0}$, we have

$$\begin{aligned}
&\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t') \\
&\quad \ge \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{D}) \\
&\quad = \frac{1}{2q_{\mathsf{alice}}} \sum_{j=1}^{2q_{\mathsf{alice}}} \mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(\mathcal{D}_j) \\
&\quad = \frac{1}{2q_{\mathsf{alice}}} \sum_{j=1}^{2q_{\mathsf{alice}}} (\Pr(\mathcal{D}_j \text{ outputs } 1 | b = 1) \\
&\qquad\qquad - \Pr(\mathcal{D}_j \text{ outputs } 1 | b = 0)) \\
&\quad = \frac{1}{2q_{\mathsf{alice}}} \sum_{j=1}^{2q_{\mathsf{alice}}} \left(\mathsf{Succ}^{\text{eu-cma}}_{\overline{\mathcal{D}}_{j,1},\kappa}(\mathcal{F}) - \mathsf{Succ}^{\text{eu-cma}}_{\overline{\mathcal{D}}_{j,0},\kappa}(\mathcal{F}) \right) \\
&\quad = \frac{1}{2q_{\mathsf{alice}}} \left(\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_2,\kappa}(\mathcal{F}) - \mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) \right).
\end{aligned}$$

Now we show that

$$\mathsf{Succ}^{\text{eu-cma}}_{\text{S-DSA}_3,\kappa}(\mathcal{F}) \le \mathsf{Succ}^{\text{eu-cma}}_{\text{DSA},\kappa}(t', q_{\mathsf{alice}}).$$

To see this, let $\mathcal{D}$ be an algorithm that takes a DSA public key $\langle g, p, q, y \rangle$ generated by $G_{\mathrm{DSA}}(1^\kappa)$ and its corresponding signature oracle and runs S-DSA$_3$ with the following modifications. In the initialization, $\mathcal{D}$ chooses $x_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and computes $y_2 \leftarrow g^{x_2} \bmod p$ and $y_1 \leftarrow y^{1/x_2} \bmod p$. In response to an $aInv2(r_2)$ query (after an $aInv1(m)$ query), $\mathcal{D}$ queries the DSA signature oracle with m to get a signature $\langle \hat{r}, \hat{s} \rangle$ and computes $r \leftarrow g^{\mathsf{h}(m)\hat{s}^{-1}} y^{\hat{r}\hat{s}^{-1}} \bmod p$, where $\hat{s}^{-1}$ is computed modulo q. Then in response to an $aInv3(\langle \mu, \mu', \Pi' \rangle)$ query, $\mathcal{D}$ returns $\langle \hat{r}, \hat{s} \rangle$ (assuming alice would not abort). Note that $\mathcal{D}$ produces a view that is perfectly indistinguishable from S-DSA$_3$ as long as no Π' is fraudulent, and we have already added the probability of this to the success probability of the adversary.

6 Proofs Π and Π'

In this section we provide an example of how alice and bob can efficiently construct and verify the noninteractive zero-knowledge proofs Π and Π' of Fig. 1. The form of these proofs naturally depends on the encryption scheme $(G_{\mathcal{E}}, E, D)$, and the particular encryption scheme for which we detail Π and Π' here is that due to Paillier [41]. We reiterate, however, that our use of Paillier is merely exemplary, and similar proofs Π and Π' can be constructed with other cryptosystems satisfying the required properties (Sect. 3).

We caution the reader that from this point forward, our use of variables is not necessarily consistent with their prior use in the paper; rather, it is necessary to replace certain variables or reuse them for different purposes.

6.1 The Paillier cryptosystem

A specific example of a cryptosystem that has the homomorphic properties required for our protocol is the first cryptosystem presented in [41]. It uses the facts that $w^{\lambda(N)} \equiv_N 1$ and $w^{N\lambda(N)} \equiv_{N^2} 1$ for any $w \in \mathbb{Z}^*_{N^2}$, where $\lambda(N)$ is the Carmichael function of N. Let L be a function that takes input elements from the set $\{u < N^2 | u \equiv 1 \bmod N\}$ and returns $L(u) = \frac{u-1}{N}$. We then define the Paillier encryption scheme (G_{Pai}, E, D) as follows. This definition differs from that in [41] only in that we define the message space M_{pk} for public key $pk = \langle N, g \rangle$ as $M_{\langle N,g \rangle} = [-(N-1)/2, (N-1)/2]$ (versus $\mathbb{Z}_N$ in [41]).

$G_{\mathrm{Pai}}(1^\kappa)$: Compute κ', choose random $\kappa'/2$-bit primes P, Q, set $N = PQ$, and choose an element $g \in \mathbb{Z}^*_{N^2}$ such that the order of g is a multiple of N ([41]). Return the public key $\langle N, g \rangle$ and the private key $\langle N, g, \lambda(N) \rangle$.

$E_{\langle N,g \rangle}(m)$: Select a random $x \in \mathbb{Z}^*_N$ and return $c = g^m x^N \bmod N^2$.

$D_{\langle N,g,\lambda(N) \rangle}(c)$: Compute $m = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$. Return m if $m \le (N-1)/2$, otherwise return $m - N$.

$c_1 +_{\langle N,g \rangle} c_2$: Return $c_1 c_2 \bmod N^2$.

$c \times_{\langle N,g \rangle} m$: Return $c^m \bmod N^2$.

Paillier [41] shows that both $c^{\lambda(N)} \bmod N^2$ and $g^{\lambda(N)} \bmod N^2$ are elements of the form $(1+N)^d \equiv_{N^2} 1 + dN$, and thus the L function can be easily computed for decryption. The security of this cryptosystem relies on the *Decision Composite Residuosity Assumption*, DCRA.

Note that we must include the initialization of the Paillier keys in the initialization of S-DSA. However, for the purposes of this section, these keys, along with the other parameters p, q in our system, are assumed to be public and fixed, and thus the language L is fixed. That is, the initialization of these parameters is *not* considered part of the initialization of Π and Π', but simply part of the definition of Π and Π'.

6.2 Strong RSA

Both Π and Π' rely on the Strong RSA problem, defined here. Let G_{RSA} be an RSA modulus generator, i.e., a probabilistic polynomial-time algorithm that takes as input 1^κ, computes κ', and produces a value $N = PQ$, where $P = 2P' + 1$ and $Q = 2Q' + 1$ are safe primes of length $\kappa'/2$. Let

$$\begin{aligned}
&\mathsf{Succ}_{\text{M-RSA},\kappa}(\mathcal{A}) \\
&\quad = \Pr[(e \ge 2) \wedge (y \equiv_N x^e) : N \leftarrow G_{\mathrm{RSA}}(1^\kappa); \\
&\qquad\qquad\qquad\qquad y \stackrel{R}{\leftarrow} \mathbb{Z}^*_N; \\
&\qquad\qquad\qquad\qquad (x, e) \leftarrow \mathcal{A}(N, y)]
\end{aligned}$$

and let

$$\mathsf{Succ}_{\text{M-RSA},\kappa}(t) = \max_{\mathcal{A}} \left\{ \mathsf{Succ}_{\text{M-RSA},\kappa}(\mathcal{A}) \right\},$$

where the maximum is taken over all adversaries of time complexity at most t.

The initialization $\mathcal{I}(1^\kappa)$ in each NIZKPI system consists of generating $\tilde{N} = \tilde{P}\tilde{Q}$, where $\tilde{P} = 2\tilde{P}' + 1$ and $\tilde{Q} = \tilde{Q}' + 1$ are safe primes of length $\kappa'/2$, a random $h_2 \stackrel{R}{\leftarrow} \mathbb{Z}^*_{\tilde{N}}$ of order $\tilde{P}'\tilde{Q}'$, a random $\chi \stackrel{R}{\leftarrow} \mathbb{Z}_{\tilde{P}'\tilde{Q}'}$, and $h_1 \leftarrow (h_2)^\chi \bmod \tilde{N}$. Note that we must include the initialization of these values in the initialization of S-DSA.

6.3 Proof Π

In this section we show how to efficiently implement the proof Π in our protocol when the Paillier cryptosystem is used. Π′ is detailed in Sect. 6.4.

Note that Π uses random oracle hash and Π′ uses a different random oracle hash′. By doing this, it is easy to see that simulations of Π proofs, even on strings not satisfying the predicate, could not be used to construct fraudulent Π′ proofs. Both hash and hash′ output elements in $\mathbb{Z}_q$.

Now consider the proof Π. Let p and q be as in a DSA public key, $pk = \langle N, g \rangle$ a Paillier public key, and $sk = \langle N, g, \lambda(N) \rangle$ the corresponding private key, where $N > q^6$. For public values c, d, w_1, w_2, m_1, m_2, we construct a zero-knowledge proof Π of:

$$\begin{bmatrix} \exists x_1, x_2 : x_1, x_2 \in [-q^3, q^3] \\ \wedge \quad c^{x_1} \equiv_p w_1 \\ \wedge \quad d^{x_2/x_1} \equiv_p w_2 \\ \wedge \quad D_{sk}(m_1) = x_1 \\ \wedge \quad D_{sk}(m_2) = x_2 \end{bmatrix} . \tag{3}$$

The proof is constructed in Fig. 2, and its verification procedure is given in Fig. 3. We assume that $c, d, w_1, w_2 \in \mathbb{Z}^*_p$ and are of order q, and that $m_1, m_2 \in \mathbb{Z}^*_{N^2}$. (The prover should verify this if necessary and abort if not true.) We assume the prover knows $x_1, x_2 \in \mathbb{Z}_q$ and $r_1, r_2 \in \mathbb{Z}^*_N$ such that $c^{x_1} \equiv_p w_1$, $d^{x_2/x_1} \equiv_p w_2$, $m_1 \equiv_{N^2} g^{x_1}(r_1)^N$ and $m_2 \equiv_{N^2} g^{x_2}(r_2)^N$. The prover need not know sk, though a malicious prover might.

$\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3}$	$\delta \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3}$
$\beta \stackrel{R}{\leftarrow} \mathbb{Z}^*_N$	$\mu \stackrel{R}{\leftarrow} \mathbb{Z}^*_N$
$\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3\tilde{N}}$	$\nu \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3\tilde{N}}$
$\rho_1 \stackrel{R}{\leftarrow} \mathbb{Z}_{q\tilde{N}}$	$\rho_2 \stackrel{R}{\leftarrow} \mathbb{Z}_{q\tilde{N}}$
	$\rho_3 \stackrel{R}{\leftarrow} \mathbb{Z}_q$
	$\epsilon \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$z_1 \leftarrow (h_1)^{x_1}(h_2)^{\rho_1} \bmod \tilde{N}$	$z_2 \leftarrow (h_1)^{x_2}(h_2)^{\rho_2} \bmod \tilde{N}$
$u_1 \leftarrow c^\alpha \bmod p$	$y \leftarrow d^{x_2+\rho_3} \bmod p$
$u_2 \leftarrow g^\alpha \beta^N \bmod N^2$	$v_1 \leftarrow d^{\delta+\epsilon} \bmod p$
$u_3 \leftarrow (h_1)^\alpha (h_2)^\gamma \bmod \tilde{N}$	$v_2 \leftarrow (w_2)^\alpha d^\epsilon \bmod p$
	$v_3 \leftarrow g^\delta \mu^N \bmod N^2$
	$v_4 \leftarrow (h_1)^\delta (h_2)^\nu \bmod \tilde{N}$
$e \leftarrow \mathsf{hash}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4)$	
$s_1 \leftarrow ex_1 + \alpha$	$t_1 \leftarrow ex_2 + \delta$
$s_2 \leftarrow (r_1)^e \beta \bmod N$	$t_2 \leftarrow e\rho_3 + \epsilon \bmod q$
$s_3 \leftarrow e\rho_1 + \gamma$	$t_3 \leftarrow (r_2)^e \mu \bmod N^2$
	$t_4 \leftarrow e\rho_2 + \nu$
$\Pi \leftarrow \langle z_1, z_2, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4 \rangle$	

Fig. 2. Construction of Π

$\langle z_1, z_2, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4 \rangle \leftarrow \Pi$	
Verify $s_1, t_1 \in \mathbb{Z}_{q^3}$	$v_1 \leftarrow d^{t_1+t_2} y^{-e} \bmod p$
$u_1 \leftarrow c^{s_1}(w_1)^{-e} \bmod p$	$v_2 \leftarrow (w_2)^{s_1} d^{t_2} y^{-e} \bmod p$
$u_2 \leftarrow g^{s_1}(s_2)^N (m_1)^{-e} \bmod N^2$	$v_3 \leftarrow g^{t_1}(t_3)^N (m_2)^{-e} \bmod N^2$
$u_3 \leftarrow (h_1)^{s_1}(h_2)^{s_3}(z_1)^{-e} \bmod \tilde{N}$	$v_4 \leftarrow (h_1)^{t_1}(h_2)^{t_4}(z_2)^{-e} \bmod \tilde{N}$
Verify $e = \mathsf{hash}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4)$	

Fig. 3. Verification of Π

Intuitively, the proof works as follows. Commitments z_1 and z_2 are made to x_1 and x_2, respectively, over the RSA modulus $\tilde{N}$, and (using values u_3 and v_4, respectively) these are proven to fall in the desired range using range proofs as in [20]. Simultaneously, it is shown (using values u_1 and u_2) that the commitment z_1 corresponds to the decryption of m_1 and the discrete log of w_1. Also, simultaneously it is shown (using values y, v_1, v_2, and v_3) that the commitment z_2 corresponds to the decryption of m_2 and that the discrete log of w_2 is the quotient of the two commitments. The full proof is shown in two columns, the left column used to prove the desired properties of x_1, w_1, and m_1 and the right column used to prove the desired properties of x_2, w_2, and m_2.

Lemma 1. *Π is an NIZKPI for predicate (3), with $\text{SIMERR}_\Pi(\kappa, n_{\text{ro}}, n_{\text{pr}}) \le n_{\text{pr}}(n_{\text{ro}}+8)2^{-\kappa+1}$ and $\text{SERR}_\Pi(\kappa, t, n_{\text{ro}}) \le \max\{4\sqrt{82 n_{\text{ro}} \mathit{Succ}_{\text{M-RSA},\kappa}(t')}, \quad 36 n_{\text{ro}} 2^{-\kappa+1}\}$, where $t' = O(t)$.*

Proof. To be completely specific, we will show that the proof Π is a proof of membership for the language

$$L = \{\langle c, w_1, d, w_2, m_1, m_2\rangle : \exists x_1, x_2 \in [-q^3, q^3] : c^{x_1} \equiv_p w_1 \wedge d^{x_2/x_1} \equiv_p w_2 \wedge D_{sk}(m_1) = x_1 \wedge D_{sk}(m_2) = x_2\}.$$

Recall that $p, q, N, g, \tilde{N}, h_1, h_2$ are determined in the initialization, which we assume for now uses a trusted party.
Completeness: Follows from inspection.
Soundness: Let $\epsilon = \text{SERR}_\Pi(\kappa, t, n_{\text{ro}})$, $\epsilon' = \epsilon/4$, and assume $\epsilon' \ge 9 n_{\text{ro}} 2^{-\kappa+1}$. Say we are given a Strong RSA instance $(\tilde{N}, C)$ generated by $\tilde{N} \leftarrow G_{\text{RSA}}(1^\kappa)$ and $C \stackrel{R}{\leftarrow} \mathbb{Z}^*_{\tilde{N}}$. Let $\mathsf{Siminit}(1^\kappa)$ compute $h_2 \leftarrow C$ and $h_1 \leftarrow C^\chi$ for $\chi \stackrel{R}{\leftarrow} \mathbb{Z}_{2^\kappa \tilde{N}}$, and output $(\tilde{N}, h_1, h_2)$. Note that $\mathsf{Siminit}(1^\kappa)$ produces a distribution statistically indistinguishable from $\mathcal{I}(1^\kappa)$ as long as C is a quadratic residue, which happens with probability $\frac{1}{4}$, and thus $\mathsf{Expt}_{\mathcal{A},\Pi}(\kappa)$ with this simulated initialization will return TRUE with probability at least ϵ'.

Now consider the following experiment, except with $\mathsf{Expt}_{\mathcal{A},\Pi}(\kappa)$ using the simulated initialization. Run $\mathsf{Expt}_{\mathcal{A},\Pi}(\kappa)$ once. Say ω consists of (1) the values determined in initialization and (2) the random tape of $\mathcal{A}$, and hash is the random oracle in this experiment. If the experiment returns TRUE, then let $\mathsf{Ind}(\omega, \mathsf{hash})$ be the index of the hash query corresponding to the string/proof pair (w, Π) returned by $\mathcal{A}$ (or $\mathsf{Ind}(\omega, \mathsf{hash}) = \infty$ if the pair (w, Π) does not correspond to any hash query made by the $\mathcal{A}$).[5] Let $\ell \leftarrow \mathsf{Ind}(\omega, \mathsf{hash})$. Then we run the experiment again with the same ω and a new random oracle hash^* that returns the same answers to all hash queries prior to hash query ℓ and random answers to hash query ℓ and all subsequent hash queries. If the experiment returns TRUE and $\mathcal{A}$ returns a pair $(w, \hat{\Pi})$, with $\mathsf{Ind}(\omega, \mathsf{hash}^*) = \ell$, where the hash ℓ returns a different value, then we output a root of C with probability at least $\frac{1}{2} - 2^{-\kappa}$ according to the algorithm below, and otherwise we abort.

Let BREAK be the probability that the algorithm does not abort. Here we show that $\Pr(\text{BREAK}) \ge 2(\epsilon')^2/81 n_{\text{ro}}$, which implies we can break Strong RSA in time $O(t)$ and probability at least $(\epsilon')^2/82 n_{\text{ro}}$, assuming $\kappa \ge 8$. Similarly to [45], let $\mathcal{A}^{\mathsf{hash}}_\omega$ denote $\mathcal{A}$ running with initialization values and random tape determined by ω and using random oracle hash. Then let

$$\mathcal{S} = \{(\omega, \mathsf{hash}) : \mathcal{A}^{\mathsf{hash}}_\omega \text{ succeeds and } \mathsf{Ind}(\omega, \mathsf{hash}) \ne \infty\}$$

and

$$\mathcal{S}_i = \{(\omega, \mathsf{hash}) : \mathcal{A}^{\mathsf{hash}}_\omega \text{ succeeds and } \mathsf{Ind}(\omega, \mathsf{hash}) = i\},$$

for $i \in \{1, \ldots, n_{\text{ro}}\}$.

Let $\delta = \Pr[\mathcal{S}] \ge \epsilon' - 2^{-\kappa+1} \ge \frac{8\epsilon'}{9}$. Let $I = \{i : \Pr(\mathcal{S}_i|\mathcal{S}) \ge 1/2n_{\text{ro}}\}$. Then as in [45, Lemma 9], $\Pr(\mathsf{Ind} \in I|\mathcal{S}) \ge \frac{1}{2}$. Obviously, for any $\mathsf{Ind} \in I$, $\Pr(\mathcal{S}_i) \ge \delta/2n_{\text{ro}}$, and then by the splitting lemma [45, Lemma 7], there exists a subset of executions Ω_i such that $\Pr(\Omega_i|\mathcal{S}_i) \ge \frac{1}{2}$ and for any $(\omega, \mathsf{hash}) \in \Omega_i$,

$$\Pr((\omega, \mathsf{hash}^*) \in \mathcal{S}_i | \mathsf{hash}^*_{<i} = \mathsf{hash}_{<i}) \ge \delta/4n_{\text{ro}},$$

where $\mathsf{hash}_{<i}$ denotes the restriction of hash to its first $i-1$ queries.

Since all $\mathcal{S}_i$ are disjoint,

$$\begin{aligned}
\Pr_{\omega,\mathsf{hash}}((\exists i \in I)(\omega, \mathsf{hash}) \in \Omega_i \cap \mathcal{S}_i|\mathcal{S}) &= \Pr\left(\bigcup_{i\in I}(\Omega_i \cap \mathcal{S}_i)|\mathcal{S}\right) \\
&= \sum_{i\in I} \Pr(\Omega_i \cap \mathcal{S}_i|\mathcal{S}) \\
&= \sum_{i\in I} \Pr(\Omega_i|\mathcal{S}_i)\Pr(\mathcal{S}_i|\mathcal{S}) \\
&\ge \frac{1}{2}\sum_{i\in I}\Pr(\mathcal{S}_i|\mathcal{S}) \\
&\ge \frac{1}{4}.
\end{aligned}$$

Recall that $\ell = \mathsf{Ind}(\omega, \mathsf{hash})$. Then with probability at least $\frac{1}{4}$, $\ell \in I$ and $(\omega, \mathsf{hash}) \in \Omega_\ell \cap \mathcal{S}_\ell$. Thus with probability at least $\delta/4$, the first execution is a success, and the adversary will succeed with probability $\delta/4n_{\text{ro}}$ on the second execution. Let ρ_ℓ denote the response of hash on query ℓ and ρ^*_ℓ the response of hash^* on query ℓ. The probability that the adversary succeeds on the second execution with $\rho_\ell \ne \rho^*_\ell$ is

$$\begin{aligned}
&\Pr((\omega, \mathsf{hash}^*) \in \mathcal{S}_\ell \wedge \rho_\ell \ne \rho^*_\ell | \mathsf{hash}^*_{<\ell} = \mathsf{hash}_{<\ell}) \\
&\quad \ge \Pr((\omega, \mathsf{hash}) \in \mathcal{S}_\ell | \mathsf{hash}^*_{<\ell} = \mathsf{hash}_{<\ell}) \\
&\qquad - \Pr(\rho_\ell = \rho^*_\ell | \mathsf{hash}^*_{<\ell} = \mathsf{hash}_{<\ell}) \\
&\quad \ge \delta/4n_{\text{ro}} - 2^{-\kappa+1}.
\end{aligned}$$

[5] Without loss of generality we may assume that all hash queries are distinct.

Thus the probability desired above is at least $\frac{\delta}{4}(\frac{\delta}{4n_{\rm ro}} - 2^{-\kappa+1}) \geq \frac{2\epsilon'}{9}(\frac{\epsilon'}{9n_{\rm ro}}) \geq 2(\epsilon')^2/81n_{\rm ro}$.

Now we show how to break Strong RSA with probability at least $\frac{1}{2} - 2^{-\kappa}$ using two pairs (w, Π) and $(w, \hat{\Pi})$ output by $\mathcal{A}_\omega^{\mathsf{hash}}$ and $\mathcal{A}_\omega^{\mathsf{hash}^*}$, respectively, where $\mathsf{Ind}(\omega, \mathsf{hash}) = \mathsf{Ind}(\omega, \mathsf{hash}^*)$.

Say $w = \texttt{<}c, w_1, d, w_2, m_1, m_2\texttt{>}$, where $w \notin L$. Note that the inputs to the hash function for Π and $\hat{\Pi}$ are the same, including $u_1, u_2, u_3, v_1, v_2, v_3, v_4$ as computed in the verification procedure, but the output of the hash function is different, say, e for Π and $\hat{e}$ for $\hat{\Pi}$. That is, we get

$$\Pi = \texttt{<}z_1, z_2, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4\texttt{>}$$
$$\hat{\Pi} = \texttt{<}z_1, z_2, y, \hat{e}, \hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{t}_1, \hat{t}_2, \hat{t}_3, \hat{t}_4\texttt{>}\,,$$

where $e \neq \hat{e}$ and

$$\begin{array}{ll}
c^{s_1} \equiv_p (w_1)^e u_1 & c^{\hat{s}_1} \equiv_p (w_1)^{\hat{e}} u_1 \\
g^{s_1}(s_2)^N \equiv_{N^2} (m_1)^e u_2 & g^{\hat{s}_1}(\hat{s}_2)^N \equiv_{N^2} (m_1)^{\hat{e}} u_2 \\
(h_1)^{s_1}(h_2)^{s_3} \equiv_{\tilde{N}} (z_1)^e u_3 & (h_1)^{\hat{s}_1}(h_2)^{\hat{s}_3} \equiv_{\tilde{N}} (z_1)^{\hat{e}} u_3 \\
d^{t_1+t_2} \equiv_p y^e v_1 & d^{\hat{t}_1+\hat{t}_2} \equiv_p y^{\hat{e}} v_1 \\
(w_2)^{s_1} d^{t_2} \equiv_p y^e v_2 & (w_2)^{\hat{s}_1} d^{\hat{t}_2} \equiv_p y^{\hat{e}} v_2 \\
g^{t_1}(t_3)^N \equiv_{N^2} (m_2)^e v_3 & g^{\hat{t}_1}(\hat{t}_3)^N \equiv_{N^2} (m_2)^{\hat{e}} v_3 \\
(h_1)^{t_1}(h_2)^{t_4} \equiv_{\tilde{N}} (z_2)^e v_4 & (h_1)^{\hat{t}_1}(h_2)^{\hat{t}_4} \equiv_{\tilde{N}} (z_2)^{\hat{e}} v_4\,.
\end{array}$$

Let $\Delta E = e - \hat{e}$, $\Delta S_1 = s_1 - \hat{s}_1$, $\Delta S_3 = s_3 - \hat{s}_3$, $\Delta T_1 = t_1 - \hat{t}_1$, and $\Delta T_4 = t_4 - \hat{t}_4$. Let $\zeta = \gcd(\Delta S_3 + \chi(\Delta S_1), \Delta E)$, and $\zeta' = \gcd(\Delta T_4 + \chi(\Delta T_1), \Delta E)$. If $\zeta \neq \Delta E$, then we can use the extended Euclidean algorithm to compute Y and Z such that $((\Delta S_3 + \chi(\Delta S_1))/\zeta)Y + ((\Delta E)/\zeta)Z = 1$, and output $((z_1)^Y C^Z \bmod \tilde{N}, (\Delta E)/\zeta)$, since $C = C^{((\Delta S_3+\chi(\Delta S_1))/\zeta)Y+((\Delta E)/\zeta)Z} = ((z_1)^{(\Delta E)/\zeta})^Y C^{((\Delta E)/\zeta)Z} = ((z_1)^Y C^Z)^{(\Delta E)/\zeta}$. Similarly, if $\zeta' \neq \Delta E$, then we can use the extended Euclidean algorithm to compute Y and Z such that $((\Delta T_4 + \chi(\Delta T_1))/\zeta')Y + ((\Delta E)/\zeta')Z = 1$, and output $((z_2)^Y C^Z \bmod \tilde{N}, (\Delta E)/\zeta')$. In either case, we would solve the Strong RSA problem. On the other hand, we will show that $\zeta = \zeta' = \Delta E$ could occur with probability at most $\frac{1}{2} + 2^{-\kappa}$.

Consider the case $\zeta = \zeta' = \Delta E$, but $\Delta E \not| \Delta S_1$. Note that we can write $\chi = \chi_0 + \chi_1 \tilde{P}'\tilde{Q}'$, and thus $\Delta S_3 + \Delta S_1 \chi = \Delta S_3 + \Delta S_1 \chi_0 + \Delta S_1 \chi_1 \tilde{P}'\tilde{Q}'$, with χ_1 randomly chosen uniformly from a set[6] of size $K \geq 2^\kappa$, and unknown to $\mathcal{A}$ (even if $\mathcal{A}$ had infinite power). Then there is a prime power a^b ($a \geq 2$) such that $a^b | \Delta E$ and $a^{b-1} | \Delta S_1$, but $a^b \not| \Delta S_1$. Note that this implies $a^{b-1} | \Delta S_3$. Now with $c_0 = (\Delta S_3 + \Delta S_1 \chi_0)/a^{b-1}$ and $c_1 = \Delta S_1 \tilde{P}'\tilde{Q}'/a^{b-1}$, we have that $0 \equiv_a c_0 + c_1\chi_1$, where $c_1 \not\equiv_a 0$. The number of elements $\chi_1 \in \mathbb{Z}_K$ in which this equivalence holds is at most $\lfloor K/a \rfloor + 1$, and thus the probability that this holds for a random choice of χ_1 is at most $\frac{1}{a} + \frac{1}{K}$. Since $a \geq 2$ and $K \geq 2^\kappa$, there is at most a probability of $\frac{1}{2} + 2^{-\kappa}$ of this equivalence holding, and otherwise we are in the case above with $\zeta \neq \Delta E$.

[6] The size of this set depends on χ_0.

The case where $\zeta = \zeta' = \Delta E$, but $\Delta E \not| \Delta T_1$, is similar.

We now show that the case where $\zeta = \zeta' = \Delta E$, $\Delta E | \Delta S_1$, and $\Delta E | \Delta T_1$ could not occur. Note that the assumptions in this case further imply $\Delta E | \Delta S_3$, and $\Delta E | \Delta T_4$. Then we can extract

$$\begin{array}{ll}
x_1'' \leftarrow \Delta S_1/\Delta E & x_2'' \leftarrow \Delta T_1/\Delta E \\
\rho_1 \leftarrow \Delta S_3/\Delta E & \rho_2 \leftarrow \Delta T_4/\Delta E \\
\alpha'' \leftarrow (e\hat{s}_1 - \hat{e}s_1)/\Delta E & \delta'' \leftarrow (e\hat{t}_1 - \hat{e}t_1)/\Delta E \\
\gamma \leftarrow (e\hat{s}_3 - \hat{e}s_3)/\Delta E & \nu \leftarrow (e\hat{t}_4 - \hat{e}t_4)/\Delta E.
\end{array}$$

These ensure $z_1 \equiv_{\tilde{N}} (h_1)^{x_1''}(h_2)^{\rho_1}$, $z_2 \equiv_{\tilde{N}} (h_1)^{x_2''}(h_2)^{\rho_2}$, $u_3 \equiv_{\tilde{N}} (h_1)^{\alpha''}(h_2)^{\gamma}$, $v_4 \equiv_{\tilde{N}} (h_1)^{\delta''}(h_2)^{\nu}$, $s_1 = ex_1'' + \alpha''$, $t_1 = ex_2'' + \delta''$, $\hat{s}_1 = \hat{e}x_1'' + \alpha''$, and $\hat{t}_1 = \hat{e}x_2'' + \delta''$. Now extract

$$x_1' \leftarrow \begin{cases} X_1' & \text{if } X_1 \leq (N-1)/2 \\ X_1' - N & \text{otherwise} \end{cases}$$
$$\text{where } X_1' \leftarrow \Delta S_1 (\Delta E)^{-1} \bmod N$$

$$x_2' \leftarrow \begin{cases} X_2' & \text{if } X_2 \leq (N-1)/2 \\ X_2' - N & \text{otherwise} \end{cases}$$
$$\text{where } X_2' \leftarrow \Delta S_2 (\Delta E)^{-1} \bmod N$$

$$\alpha' \leftarrow \begin{cases} A' & \text{if } A' \leq (N-1)/2 \\ A' - N & \text{otherwise} \end{cases}$$
$$\text{where } A' \leftarrow (e\hat{s}_1 - \hat{e}s_1)(\Delta E)^{-1} \bmod N$$

$$\delta' \leftarrow \begin{cases} D' & \text{if } D' \leq (N-1)/2 \\ D' - N & \text{otherwise} \end{cases}$$
$$\text{where } D' \leftarrow (e\hat{t}_1 - \hat{e}t_1)(\Delta E)^{-1} \bmod N.$$

Note that $x_1', x_2', \alpha', \delta' \in M_{\texttt{<}N,g\texttt{>}}$, and there must exist $r_1, r_2, \beta, \nu \in \mathbb{Z}_N^*$ such that $m_1 \equiv_{N^2} g^{x_1'}(r_1)^N$, $m_2 \equiv_{N^2} g^{x_2'}(r_2)^N$, $u_2 \equiv_{N^2} g^{\alpha'}\beta^N$, and $v_3 \equiv_{N^2} g^{\delta'}\nu^N$. Next extract

$$x_1 \leftarrow \Delta S_1 (\Delta E)^{-1} \bmod q \quad \alpha \leftarrow (e\hat{s}_1 - \hat{e}s_1)(\Delta E)^{-1} \bmod q$$

so that $s_1 \equiv_q ex_1 + \alpha$ and $\hat{s}_1 \equiv_q \hat{e}x_1 + \alpha$. Finally, to extract an x_2 and δ so that $t_1 \equiv_q ex_2 + \delta$ and $\hat{t}_1 \equiv_q \hat{e}x_2 + \delta$, note that there exist values $x_3, \rho_3, \epsilon, \eta_1, \eta_2, \eta_3 \in \mathbb{Z}_q$ such that

$$\begin{array}{ll}
t_1 + t_2 \equiv_q e\eta_3 + \eta_1 & \hat{t}_1 + \hat{t}_2 \equiv_q \hat{e}\eta_3 + \eta_1 \\
x_3 s_1 + t_2 \equiv_q e\eta_3 + \eta_2 & x_3\hat{s}_1 + \hat{t}_2 \equiv_q \hat{e}\eta_3 + \eta_2\,.
\end{array}$$

From the second row we can extract η_3 and η_1. From the third row and the value η_3, noting that $s_1 \neq \hat{s}_1$, we can extract x_3 and η_2. Then we can compute $x_2 \leftarrow x_3 x_1 \bmod q$, $\rho_3 \leftarrow \eta_3 - x_2 \bmod q$, $\epsilon \leftarrow \eta_2 - x_3\alpha \bmod q$, and $\delta \leftarrow \eta_1 - \epsilon \bmod q$.

Now since $w \notin L$, either $(x_1' \notin [-q^3, q^3]) \vee (x_1' \not\equiv_q x_1)$ or $(x_2' \notin [-q^3, q^3]) \vee (x_2' \not\equiv_q x_2)$. First assume $(x_1' \notin [-q^3, q^3]) \vee (x_1' \not\equiv_q x_1)$. Note that we have the following equations, with all values known:

$$\begin{array}{ll}
s_1 \equiv_q ex_1 + \alpha & \hat{s}_1 \equiv_q \hat{e}x_1 + \alpha \\
s_1 \equiv_N ex_1' + \alpha' & \hat{s}_1 \equiv_N \hat{e}x_1' + \alpha' \\
s_1 = ex_1'' + \alpha'' & \hat{s}_1 = \hat{e}x_1'' + \alpha'',
\end{array}$$

where $s_1, \hat{s}_1 \in \mathbb{Z}_{q^3}$.

Using these equations, we can determine the following facts:

1. $x_1'' \equiv_q x_1$ and $\alpha'' \equiv_q \alpha$
   Proof: $ex_1'' + \alpha'' \equiv_q ex_1 + \alpha$ and $\hat{e}x_1'' + \alpha'' \equiv_q \hat{e}x_1 + \alpha$ implies $x_1''(e - \hat{e}) \equiv_q x_1(e - \hat{e})$, which implies the result.
2. $x_1'' \equiv_N x_1'$ and $\alpha'' \equiv_N \alpha'$
   Proof: similar to the previous fact.
3. $x_1'' \in [-q^3, q^3]$ and $\alpha'' \in [-q^4, q^4]$
   Proof: $|x_1''| \leq |s_1 - \hat{s}_1| \leq q^3$, and thus $|\alpha''| = |s_1 - ex_1''| \leq q^4$.
4. $x_1' \in [-q^3, q^3]$ and $\alpha' \in [-q^4, q^4]$
   Proof: Since $N > q^6$ and $x_1'' \equiv_N x_1'$.
5. $s_1 = ex_1' + \alpha'$ and $\hat{s}_1 = \hat{e}x_1' + \alpha'$
   Proof: Since $N > q^6$, this follows from the previous fact, and the equations $s_1 \equiv_N ex_1' + \alpha'$ and $\hat{s}_1 \equiv_N \hat{e}x_1' + \alpha'$.
6. $x_1' \equiv_q x_1$
   Proof: $ex_1' + \alpha' \equiv_q ex_1 + \alpha$ and $\hat{e}x_1' + \alpha' \equiv_q \hat{e}x_1 + \alpha$ implies $x_1'(e - \hat{e}) \equiv_q x_1(e - \hat{e})$, which implies the result.

But this would contradict our assumption that $(x_1' \notin [-q^3, q^3]) \vee (x_1' \not\equiv_q x_1)$.

So now assume $(x_2' \notin [-q^3, q^3]) \vee (x_2' \not\equiv_q x_2)$. Note that we have the following equations, with all values known:

$$\begin{array}{ll} t_1 \equiv_q ex_2 + \delta & \hat{t}_1 \equiv_q \hat{e}x_2 + \delta \\ t_1 \equiv_N ex_2' + \delta' & \hat{t}_1 \equiv_N \hat{e}x_2' + \delta' \\ t_1 = ex_2'' + \delta'' & \hat{t}_1 = \hat{e}x_2'' + \delta'', \end{array}$$

where $t_1, \hat{t}_1 \in \mathbb{Z}_{q^3}$.

By arguments analogous to those above, we can show that this would contradict our assumpion that $(x_2' \notin [-q^3, q^3]) \vee (x_2' \not\equiv_q x_2)$.

Zero-knowledge: We construct a simulator Sim that takes a string <c, w_1, d, w_2, m_1, m_2>, where $c, w_1, d, w_2 \in \mathbb{Z}_p^*$ are of order q and $m_1, m_2 \in \mathbb{Z}_{N^2}^*$, and outputs a valid proof, using the standard technique of "backpatching" random oracle queries. The simulator operates as in Fig. 4. From inspection, the proof Π is valid (i.e., it verifies).

It is trivial to see that with overwhelming probability this backpatching is consistent, i.e., the random oracle has not been previously queried on the input for which we are backpatching. If the backpatching is inconsistent, we abort. If we only consider the random value y, we see that backpatching can be inconsistent with probability at most n_{ro}/q. Also, the probability of any values $s_1, t_1 \in [q^2, q^3 - q^2]$ is the same for the real protocol and simulator, the probability of any values $s_3, t_4 \in [q^2\tilde{N}, q^3\tilde{N} - q^2\tilde{N}]$ is the same for the real protocol and simulator, and the probability of any values s_2, t_2, t_3 is the same for the real protocol and the simulator. Thus in total the distinguishing probability (for each simulated proof) is at most

$$\frac{n_{\mathrm{ro}}}{q} + 2 \cdot \frac{2q^2}{q^3} + 2 \cdot \frac{2q^2\tilde{N}}{q^3\tilde{N}} = \frac{n_{\mathrm{ro}}}{q} + \frac{8}{q} \leq (n_{\mathrm{ro}} + 8)2^{-\kappa+1}.$$

The bound in the lemma comes from the fact that this simulator is called n_{pr} times.

6.4 Proof Π'

Now we look at the proof Π'. Let p and q be as in a DSA public key, $pk =$ <N, g> and $sk =$ <$N, g, \lambda(N)$> be a Paillier key pair with $N > q^9$, and $pk' =$ <N', g'> and $sk' =$

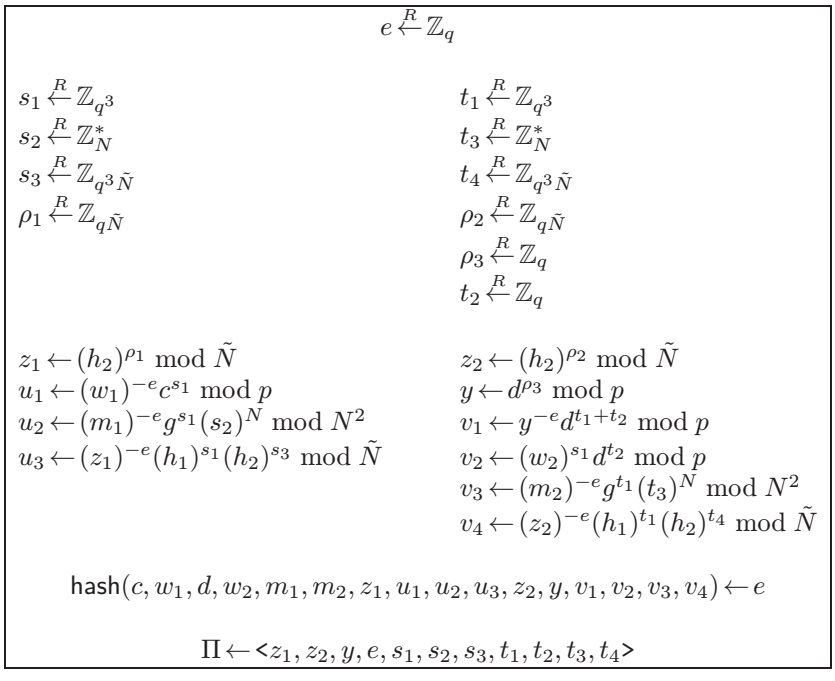

Fig. 4. Simulator for Π

$<N', g', \lambda(N')>$ be a Paillier key pair with $N' > q^6$. For values c, d, w_1, w_2, m_1, m_2, m_3, m_4 such that $D_{sk}(m_3) \in [-q^4, q^4]$ and $D_{sk}(m_4) \in [-q^4, q^4]$, we construct a zero-knowledge proof Π' of:

$$\left[\begin{array}{ll} \exists x_1, x_2, x_3 : & x_1, x_2 \in [-q^3, q^3] \\ \wedge & x_3 \in [-q^7, q^7] \\ \wedge & c^{x_1} \equiv_p w_1 \\ \wedge & d^{x_2/x_1} \equiv_p w_2 \\ \wedge & D_{sk'}(m_1) = x_1 \\ \wedge & D_{sk}(m_2) = (D_{sk}(m_3))x_1 + (D_{sk}(m_4))x_2 + qx_3 \end{array}\right] \quad (4)$$

We note that Eq. (4) is stronger than what is needed, as shown in Fig. 1. The proof is constructed in Fig. 5, and the verification procedure for it is given in Fig. 6. We assume that $c, d, w_1, w_2 \in \mathbb{Z}_p^*$ and are of order q, and that $m_1 \in \mathbb{Z}^*_{(N')^2}$ and $m_2 \in \mathbb{Z}^*_{N^2}$. (The prover should verify this if necessary.) We assume the prover knows $x_1, x_2 \in \mathbb{Z}_q$, $x_3 \in \mathbb{Z}_{q^5}$, and $r_1, r_2 \in \mathbb{Z}_N^*$, such that $c^{x_1} \equiv_p w_1$, $d^{x_2/x_1} \equiv_p w_2$, $m_1 \equiv_{(N')^2} (g')^{x_1}(r_1)^{N'}$, and $m_2 \equiv_{N^2} (m_3)^{x_1}(m_4)^{x_2} g^{qx_3}(r_2)^N$. The prover need not know sk or sk', though a malicious prover might know sk'. We assume the verifier knows $D_{sk}(m_3)$ and $D_{sk}(m_4)$. If necessary, the verifier should verify that $c, d, w_1, w_2 \in \mathbb{Z}_p^*$ and are of order q, and that $m_1 \in \mathbb{Z}^*_{(N')^2}$ and $m_2 \in \mathbb{Z}^*_{N^2}$.

Lemma 2. *Π' is an NIZKPI for predicate Eq. (4), with* $\textsc{Simerr}_{\Pi'}(\kappa, n_{\rm ro}, n_{\rm pr}) \le n_{\rm pr}(n_{\rm ro} + 12)2^{-\kappa+1}$ *and* $\textsc{Serr}_{\Pi'}(\kappa, t, n_{\rm ro}) \le \max\{4\sqrt{82 n_{\rm ro} \mathsf{Succ}_{\text{M-RSA},\kappa}(t')}, 36 n_{\rm ro} 2^{-\kappa+1}\}$, *where $t' = O(t)$.*

Proof. To be completely specific, we will show that the proof Π' is a proof of membership for the language

$$L' = \{<c, w_1, d, w_2, m_1, m_2, m_3, m_4> : \exists x_1, x_2 \in [-q^3, q^3], x_3 \in [-q^7, q^7] : c^{x_1} \equiv_p w_1 \wedge d^{x_2/x_1} \equiv_p w_2 \wedge D_{sk'}(m_1) = x_1 \wedge D_{sk}(m_2) = (D_{sk}(m_3))x_1 + (D_{sk}(m_4))x_2 + qx_3\}.$$

Recall that $p, q, N, g, N', g', \tilde{N}, h_1, h_2$ are determined in the initialization, which we assume for now uses a trusted party, and that $D_{sk}(m_3) \in [-q^4, q^4]$ and $D_{sk}(m_4) \in [-q^4, q^4]$.

Completeness: Follows from inspection.

$\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3}$	$\delta \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3}$
$\beta \stackrel{R}{\leftarrow} \mathbb{Z}^*_{N'}$	$\mu \stackrel{R}{\leftarrow} \mathbb{Z}^*_N$
$\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3\tilde{N}}$	$\nu \stackrel{R}{\leftarrow} \mathbb{Z}_{q^3\tilde{N}}$
$\rho_1 \stackrel{R}{\leftarrow} \mathbb{Z}_{q\tilde{N}}$	$\rho_2 \stackrel{R}{\leftarrow} \mathbb{Z}_{q\tilde{N}}$
	$\rho_3 \stackrel{R}{\leftarrow} \mathbb{Z}_q$
	$\rho_4 \stackrel{R}{\leftarrow} \mathbb{Z}_{q^5\tilde{N}}$
	$\epsilon \stackrel{R}{\leftarrow} \mathbb{Z}_q$
	$\sigma \stackrel{R}{\leftarrow} \mathbb{Z}_{q^7}$
	$\tau \stackrel{R}{\leftarrow} \mathbb{Z}_{q^7\tilde{N}}$
$z_1 \leftarrow (h_1)^{x_1}(h_2)^{\rho_1} \bmod \tilde{N}$	$z_2 \leftarrow (h_1)^{x_2}(h_2)^{\rho_2} \bmod \tilde{N}$
$u_1 \leftarrow c^{\alpha} \bmod p$	$y \leftarrow d^{x_2+\rho_3} \bmod p$
$u_2 \leftarrow (g')^{\alpha}\beta^{N'} \bmod (N')^2$	$v_1 \leftarrow d^{\delta+\epsilon} \bmod p$
$u_3 \leftarrow (h_1)^{\alpha}(h_2)^{\gamma} \bmod \tilde{N}$	$v_2 \leftarrow (w_2)^{\alpha} d^{\epsilon} \bmod p$
	$v_3 \leftarrow (m_3)^{\alpha}(m_4)^{\delta} g^{q\sigma}\mu^N \bmod N^2$
	$v_4 \leftarrow (h_1)^{\delta}(h_2)^{\nu} \bmod \tilde{N}$
	$z_3 \leftarrow (h_1)^{x_3}(h_2)^{\rho_4} \bmod \tilde{N}$
	$v_5 \leftarrow (h_1)^{\sigma}(h_2)^{\tau} \bmod \tilde{N}$
$e \leftarrow \mathsf{hash}'(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, z_3, y, v_1, v_2, v_3, v_4, v_5)$	
$s_1 \leftarrow ex_1 + \alpha$	$t_1 \leftarrow ex_2 + \delta$
$s_2 \leftarrow (r_1)^e \beta \bmod N'$	$t_2 \leftarrow e\rho_3 + \epsilon \bmod q$
$s_3 \leftarrow e\rho_1 + \gamma$	$t_3 \leftarrow (r_2)^e \mu \bmod N$
	$t_4 \leftarrow e\rho_2 + \nu$
	$t_5 \leftarrow ex_3 + \sigma$
	$t_6 \leftarrow e\rho_4 + \tau$
$\Pi' \leftarrow <z_1, z_2, z_3, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4, t_5, t_6>$	

Fig. 5. Construction of Π'

$<z_1, z_2, z_3, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4, t_5, t_6> \leftarrow \Pi'$

Verify $s_1, t_1 \in \mathbb{Z}_{q^3}$	$v_1 \leftarrow d^{t_1+t_2} y^{-e} \bmod p$
Verify $t_5 \in \mathbb{Z}_{q^7}$	$v_2 \leftarrow (w_2)^{s_1} d^{t_2} y^{-e} \bmod p$
$u_1 \leftarrow c^{s_1}(w_1)^{-e} \bmod p$	$v_3 \leftarrow (m_3)^{s_1}(m_4)^{t_1} g^{qt_5}(t_3)^N (m_2)^{-e}$
$u_2 \leftarrow (g')^{s_1}(s_2)^{N'}(m_1)^{-e}$	$\bmod N^2$
$\bmod (N')^2$	$v_4 \leftarrow (h_1)^{t_1}(h_2)^{t_4}(z_2)^{-e} \bmod \tilde{N}$
$u_3 \leftarrow (h_1)^{s_1}(h_2)^{s_3}(z_1)^{-e} \bmod \tilde{N}$	$v_5 \leftarrow (h_1)^{t_5}(h_2)^{t_6}(z_3)^{-e} \bmod \tilde{N}$

Verify $e = \mathsf{hash}'(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, z_3, y, v_1, v_2, v_3, v_4, v_5)$

Fig. 6. Verification of Π'

Soundness: Let $\epsilon = \textsc{Serr}_{\Pi'}(\kappa, t, n_{\rm ro})$, $\epsilon' = \epsilon/4$ and assume $\epsilon' \geq 9n_{\rm ro}2^{-\kappa+1}$. Say we are given a Strong RSA instance $(\tilde{N}, C)$ generated by $\tilde{N} \leftarrow G_{\rm RSA}(1^\kappa)$ and $C \stackrel{R}{\leftarrow} \mathbb{Z}^*_{\tilde{N}}$. Let $\mathsf{Siminit}(1^\kappa)$ compute $h_2 \leftarrow C$ and $h_1 \leftarrow C^\chi$ for $\chi \stackrel{R}{\leftarrow} \mathbb{Z}_{2^\kappa \tilde{N}}$, and output $(\tilde{N}, h_1, h_2)$. Note that $\mathsf{Siminit}(1^\kappa)$ produces a distribution statistically indistinguishable from $\mathcal{I}(1^\kappa)$ as long as C is a quadratic residue, which happens with probability $\frac{1}{4}$, and thus $\mathsf{Expt}_{\mathcal{A},\Pi'}(\kappa)$ with this simulated initialization will also return TRUE with probability at least ϵ'.

Now as in Lemma 1, consider the following experiment, except with $\mathsf{Expt}_{\mathcal{A},\Pi'}(\kappa)$ using the simulated initialization. Run $\mathsf{Expt}_{\mathcal{A},\Pi'}(\kappa)$ once. Say ω consists of (1) the values determined in initialization and (2) the random tape of $\mathcal{A}$, and hash' is the random oracle in this experiment. If the experiment returns TRUE, then let $\mathsf{Ind}(\omega, \mathsf{hash}')$ be the index of the hash query corresponding to the string/proof pair (w, Π') returned by $\mathcal{A}$ (or $\mathsf{Ind}(\omega, \mathsf{hash}') = \infty$ if the pair (w, Π') does not correspond to any hash query made by the $\mathcal{A}$).[7] Let $\ell \leftarrow \mathsf{Ind}(\omega, \mathsf{hash}')$. Then we run the experiment again with the same ω and a new random oracle hash^* that returns the same answers to all hash queries prior to hash query ℓ and random answers to hash query ℓ and all subsequent hash queries. If the experiment returns TRUE and $\mathcal{A}$ returns a pair $(w, \hat{\Pi}')$, with $\mathsf{Ind}(\omega, \mathsf{hash}^*) = \ell$, where the hash ℓ returns a different value, then we output a root of C with probability at least $\frac{1}{2} - 2^{-\kappa}$ according to the algorithm below, and otherwise we abort.

Let BREAK be the probability that the algorithm does not abort. As in the proof of Lemma 1, we can show that $\Pr(\textsc{break}) \geq 2(\epsilon')^2/81n_{\rm ro}$, which implies we can break Strong RSA in time $O(t)$ and with probability at least $(\epsilon')^2/82n_{\rm ro}$, assuming $\kappa \geq 8$.

Now we show how to break Strong RSA with probability at least $\frac{1}{2} - 2^{-\kappa}$ using two pairs (w, Π') and $(w, \hat{\Pi}')$ output by $\mathcal{A}^{\mathsf{hash}'}_\omega$ and $\mathcal{A}^{\mathsf{hash}^*}_\omega$, respectively, where $\mathsf{Ind}(\omega, \mathsf{hash}') = \mathsf{Ind}(\omega, \mathsf{hash}^*)$.

Say $w = <c, w_1, d, w_2, m_1, m_2, m_3, m_4>$, where $w \notin L$. Note that the inputs to the hash function for Π' and $\hat{\Pi}'$ are the same, including $u_1, u_2, u_3, v_1, v_2, v_3, v_4, v_5$ as computed in the verification procedure, but the output of the hash function is different, say, e for Π' and $\hat{e}$ for $\hat{\Pi}'$. That is, we get

$$\Pi' = <z_1, z_2, z_3, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4, t_5, t_6>$$
$$\hat{\Pi}' = <z_1, z_2, z_3, y, \hat{e}, \hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{t}_1, \hat{t}_2, \hat{t}_3, \hat{t}_4, \hat{t}_5, \hat{t}_6>,$$

where $e \neq \hat{e}$ and

$c^{s_1} \equiv_p (w_1)^e u_1$	$c^{\hat{s}_1} \equiv_p (w_1)^{\hat{e}} u_1$
$(g')^{s_1}(s_2)^{N'} \equiv_{(N')^2} (m_1)^e u_2$	$(g')^{\hat{s}_1}(\hat{s}_2)^{N'} \equiv_{(N')^2} (m_1)^{\hat{e}} u_2$
$(h_1)^{s_1}(h_2)^{s_3} \equiv_{\tilde{N}} (z_1)^e u_3$	$(h_1)^{\hat{s}_1}(h_2)^{\hat{s}_3} \equiv_{\tilde{N}} (z_1)^{\hat{e}} u_3$
$d^{t_1+t_2} \equiv_p y^e v_1$	$d^{\hat{t}_1+\hat{t}_2} \equiv_p y^{\hat{e}} v_1$
$(w_2)^{s_1} d^{t_2} \equiv_p y^e v_2$	$(w_2)^{\hat{s}_1} d^{\hat{t}_2} \equiv_p y^{\hat{e}} v_2$
$(m_3)^{s_1}(m_4)^{t_1} g^{qt_5}(t_3)^N$	$(m_3)^{\hat{s}_1}(m_4)^{\hat{t}_1} g^{q\hat{t}_5}(\hat{t}_3)^N$
$\equiv_{N^2} (m_2)^e v_3$	$\equiv_{N^2} (m_2)^{\hat{e}} v_3$
$(h_1)^{t_1}(h_2)^{t_4} \equiv_{\tilde{N}} (z_2)^e v_4$	$(h_1)^{\hat{t}_1}(h_2)^{\hat{t}_4} \equiv_{\tilde{N}} (z_2)^{\hat{e}} v_4$
$(h_1)^{t_5}(h_2)^{t_6} \equiv_{\tilde{N}} (z_3)^e v_5$	$(h_1)^{\hat{t}_5}(h_2)^{\hat{t}_6} \equiv_{\tilde{N}} (z_3)^{\hat{e}} v_5$.

Let $\Delta E = e - \hat{e}$, $\Delta S_1 = s_1 - \hat{s}_1$, $\Delta S_3 = s_3 - \hat{s}_3$, $\Delta T_1 = t_1 - \hat{t}_1$, $\Delta T_4 = t_4 - \hat{t}_4$, $\Delta T_5 = t_5 - \hat{t}_5$, $\Delta T_6 = t_6 - \hat{t}_6$, and let $\zeta = \gcd(\Delta S_3 + \chi(\Delta S_1), \Delta E)$, $\zeta' = \gcd(\Delta T_4 + \chi(\Delta T_1), \Delta E)$, and $\zeta'' = \gcd(\Delta T_6 + \chi(\Delta T_5), \Delta E)$. If $\zeta \neq \Delta E$, then we can use the extended Euclidean algorithm to compute Y and Z such that $((\Delta S_3 + \chi(\Delta S_1))/\zeta)Y + ((\Delta E)/\zeta)Z = 1$, and output $((z_1)^Y C^Z \bmod \tilde{N}, (\Delta E)/\zeta)$, since

$$\begin{aligned} C &= C^{((\Delta S_3 + \chi(\Delta S_1))/\zeta)Y + ((\Delta E)/\zeta)Z} \\ &= ((z_1)^{(\Delta E)/\zeta})^Y C^{((\Delta E)/\zeta)Z} \\ &= ((z_1)^Y C^Z)^{(\Delta E)/\zeta}. \end{aligned}$$

Similarly, if $\zeta' \neq \Delta E$, then we can use the extended Euclidean algorithm to compute Y and Z such that $((\Delta T_4 + \chi(\Delta T_1))/\zeta')Y + ((\Delta E)/\zeta')Z = 1$, and output $((z_2)^Y C^Z \bmod \tilde{N}, (\delta E)/\zeta')$. Similarly, if $\zeta'' \neq \Delta E$, then we can use the extended Euclidean algorithm to compute Y and Z such that $((\Delta T_6 + \chi(\Delta T_5))/\zeta'')Y + ((\Delta E)/\zeta'')Z = 1$, and output $((z_3)^Y C^Z \bmod \tilde{N}, (\delta E)/\zeta'')$. In any case, we would solve the Strong RSA problem. On the other hand, we would show that $\zeta = \zeta' = \zeta'' = \Delta E$ could occur with probability at most $\frac{1}{2} + 2^{-\kappa}$.

Consider the case $\zeta = \zeta' = \zeta'' = \Delta E$, but $\Delta E \not| \Delta S_1$. Note that we can write $\chi = \chi_0 + \chi_1 \tilde{P}' \tilde{Q}'$, and thus $\Delta S_3 +$

[7] Without loss of generality we may assume that all hash queries are distinct.

$\Delta S_1\chi = \Delta S_3 + \Delta S_1\chi_0 + \Delta S_1\chi_1\tilde{P}'\tilde{Q}'$, with χ_1 randomly chosen uniformly from a set[8] of size $K \geq 2^\kappa$, and unknown to $\mathcal{A}$ (even if $\mathcal{A}$ had infinite power). Then there is a prime power a^b ($a \geq 2$) such that $a^b|\Delta E$ and $a^{b-1}|\Delta S_1$, but $a^b \not| \Delta S_1$. Note that this implies $a^{b-1}|\Delta S_3$. Now with $c_0 = (\Delta S_3 + \Delta S_1\chi_0)/a^{b-1}$ and $c_1 = \Delta S_1\tilde{P}'\tilde{Q}'/a^{b-1}$, we have that $0 \equiv_a c_0 + c_1\chi_1$, where $c_1 \not\equiv_a 0$. The number of elements $\chi_1 \in \mathbb{Z}_K$ in which this equivalence holds is at most $\lfloor K/a \rfloor + 1$, and thus the probability that this holds for a random choice of χ_1 is at most $\frac{1}{a} + \frac{1}{K}$. Since $a \geq 2$ and $K \geq 2^\kappa$, there is at most a probability of $\frac{1}{2} + 2^{-\kappa}$ of this equivalence holding, and otherwise we are in the case above with $\zeta \neq \Delta E$.

The cases where $\zeta = \zeta' = \zeta'' = \Delta E$, but $\Delta E \not| \Delta T_1$ or $\Delta E \not| \Delta T_5$ are similar.

We now show that the case where $\zeta = \zeta' = \zeta'' = \Delta E$, $\Delta E|\Delta S_1$, $\Delta E|\Delta T_1$, and $\Delta E|\Delta T_5$ could not occur. Note that the assumptions in this case further imply $\Delta E|\Delta S_3$, $\Delta E|\Delta T_4$, and $\Delta E|\Delta T_6$. Then we can extract $x_1'', \rho_1, \alpha'', \gamma$, where $z_1 \equiv_{\tilde{N}} (h_1)^{x_1''}(h_2)^{\rho_1}$ and $u_3 \equiv_{\tilde{N}} (h_1)^{\alpha''}(h_2)^{\gamma}$, and moreover, $s_1 = ex_1'' + \alpha''$ and $\hat{s}_1 = \hat{e}x_1'' + \alpha''$. Also, we can extract $x_2'', \rho_2, \delta'', \nu$, where $z_2 \equiv_{\tilde{N}} (h_1)^{x_2''}(h_2)^{\rho_2}$ and $v_4 \equiv_{\tilde{N}} (h_1)^{\delta''}(h_2)^{\nu}$, and moreover, $t_1 = ex_2'' + \delta''$ and $\hat{t}_1 = \hat{e}x_2'' + \delta''$. Finally, we can extract $x_3'', \rho_4, \sigma'', \tau$, where $z_3 \equiv_{\tilde{N}} (h_1)^{x_3''}(h_2)^{\rho_4}$ and $v_5 \equiv_{\tilde{N}} (h_1)^{\sigma''}(h_2)^{\tau}$, and moreover, $t_5 = ex_3'' + \sigma''$ and $\hat{t}_5 = \hat{e}x_3'' + \sigma''$.

Similar to the soundness proof of Lemma 1, we can extract $x_1, x_2, \alpha, \delta \in \mathbb{Z}_q$ satisfying the following equations:

$$\begin{array}{ll} s_1 \equiv_q ex_1 + \alpha & \hat{s}_1 \equiv_q \hat{e}x_1 + \alpha \\ t_1 \equiv_q ex_2 + \delta & \hat{t}_1 \equiv_q \hat{e}x_2 + \delta. \end{array}$$

Also similar to the soundness proof of Lemma 1, we can extract $x_1', x_2', x_3', \alpha', \delta', \sigma' \in \mathbb{Z}_N$ satisfying

$$\begin{array}{ll} s_1 \equiv_N ex_1' + \alpha' & \hat{s}_1 \equiv_N \hat{e}x_1' + \alpha' \\ t_1 \equiv_N ex_2' + \delta' & \hat{t}_1 \equiv_N \hat{e}x_2' + \delta' \\ t_5 \equiv_N ex_3' + \sigma' & \hat{t}_5 \equiv_N \hat{e}x_3' + \sigma', \end{array}$$

where $s_1, \hat{s}_1, t_1, \hat{t}_1 \in \mathbb{Z}_{q^3}$ and $t_5, \hat{t}_5 \in \mathbb{Z}_{q^7}$.

Now, since $w \notin L$, either $(x_1' \notin [-q^3, q^3]) \vee (x_1' \not\equiv_q x_1)$ or $(\nexists x_2' \in [-q^3, q^3], x_3' \in [-q^7, q^7] : (D_{sk}(m_2) = (D_{sk}(m_3))x_1' + (D_{sk}(m_4))x_2' + qx_3') \wedge (x_2' \equiv_q x_2))$.

First assume $(x_1' \notin [-q^3, q^3]) \vee (x_1' \not\equiv_q x_1)$. By the same reasoning as in the proof of Lemma 1, we get a contradiction.

So instead assume that $(\nexists x_2' \in [-q^3, q^3], x_3' \in [-q^7, q^7] : (D_{sk}(m_2) = (D_{sk}(m_3))x_1' + (D_{sk}(m_4))x_2' + qx_3') \wedge (x_2' \equiv_q x_2))$. By the same reasoning as in the proof of Lemma 1, we get that $x_1'' \equiv_q x_1$ and $x_2'' \equiv_q x_2$, as well as $x_1'' \equiv_N x_1'$, $x_2'' \equiv_N x_2'$, and $x_3'' \equiv_N x_3'$. Also we get that $x_1', x_2' \in [-q^3, q^3]$ and $x_3' \in [-q^7, q^7]$ (using the fact that $N > q^9$).

[8] The size of this set depends on χ_0.

$e \xleftarrow{R} \mathbb{Z}_q$

$s_1 \xleftarrow{R} \mathbb{Z}_{q^3}$	$t_1 \xleftarrow{R} \mathbb{Z}_{q^3}$
$s_2 \xleftarrow{R} \mathbb{Z}^*_{N'}$	$t_3 \xleftarrow{R} \mathbb{Z}^*_N$
$s_3 \xleftarrow{R} \mathbb{Z}_{q^3\tilde{N}}$	$t_4 \xleftarrow{R} \mathbb{Z}_{q^3\tilde{N}}$
$\rho_1 \xleftarrow{R} \mathbb{Z}_{q\tilde{N}}$	$\rho_2 \xleftarrow{R} \mathbb{Z}_{q\tilde{N}}$
	$\rho_3 \xleftarrow{R} \mathbb{Z}_q$
	$\rho_4 \xleftarrow{R} \mathbb{Z}_{q^7\tilde{N}}$
	$t_2 \xleftarrow{R} \mathbb{Z}_q$
	$t_5 \xleftarrow{R} \mathbb{Z}_{q^7}$
	$t_6 \xleftarrow{R} \mathbb{Z}_{q^7\tilde{N}}$
$z_1 \leftarrow (h_2)^{\rho_1} \bmod \tilde{N}$	$z_2 \leftarrow (h_2)^{\rho_2} \bmod \tilde{N}$
$u_1 \leftarrow (w_1)^{-e} c^{s_1} \bmod p$	$y \leftarrow d^{\rho_3} \bmod p$
$u_2 \leftarrow (m_1)^{-e}(g')^{s_1}(s_2)^{N'} \bmod (N')^2$	$v_1 \leftarrow y^{-e} d^{t_1+t_2} \bmod p$
$u_3 \leftarrow (z_1)^{-e}(h_1)^{s_1}(h_2)^{s_3} \bmod \tilde{N}$	$v_2 \leftarrow (w_2)^{s_1} d^{t_2} \bmod p$
	$v_3 \leftarrow (m_2)^{-e}(m_3)^{s_1}(m_4)^{t_1} g^{qt_5}(t_3)^N \bmod N^2$
	$v_4 \leftarrow (z_2)^{-e}(h_1)^{t_1}(h_2)^{t_4} \bmod \tilde{N}$
	$z_3 \leftarrow (h_2)^{\rho_4} \bmod \tilde{N}$
	$v_5 \leftarrow (z_3)^{-e}(h_1)^{t_5}(h_2)^{t_6} \bmod \tilde{N}$

$\mathsf{hash}'(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, z_3, y, v_1, v_2, v_3, v_4, v_5) \leftarrow e$

$\Pi' \leftarrow <z_1, z_2, z_3, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4, t_5, t_6>$

Fig. 7. Simulator for Π'

Consequently, again as in the proof of Lemma 1, $x'_1 \equiv_q x_1$, $x'_2 \equiv_q x_2$, and $x'_3 \equiv_q x_3$. Thus $x'_2 \in [-q^3, q^3]$, $x'_3 \in [-q^7, q^7]$, and $x'_2 \equiv_q x_2$, which contradicts our assumption.

Zero-knowledge: We construct a simulator Sim that takes a string <c, w_1, d, w_2, m_1, m_2, m_3, m_4>, where $c, w_1, d, w_2 \in \mathbb{Z}_p^*$ are of order q, $m_1 \in \mathbb{Z}^*_{(N')^2}$, and m_2, m_3, $m_4 \in \mathbb{Z}^*_{N^2}$, and outputs a valid proof, using the standard technique of "backpatching" random oracle queries. The simulator operates as in Fig. 7. From inspection, the proof Π' is valid (i.e., it verifies).

It is trivial to see that with overwhelming probability this backpatching is consistent, i.e., the random oracle has not been previously queried on the input for which we are backpatching. If the backpatching is inconsistent, we abort. If we only consider the random value y, we see that backpatching can be inconsistent with probability at most $n_{\rm ro}/q$. Also, the probability of any values $s_1, t_1 \in [q^2, q^3 - q^2]$ is the same for the real protocol and simulator, the probability of any values $s_3, t_4 \in [q^2\tilde{N}, q^3\tilde{N} - q^2\tilde{N}]$ is the same for the real protocol and simulator, the probability of any value $t_5 \in [q^6, q^7 - q^6]$ is the same for the real protocol and simulator, the probability of any value $t_6 \in [q^6\tilde{N}, q^7\tilde{N} - q^6\tilde{N}]$ is the same for the real protocol and simulator, and the probability of any values for s_2, t_2, t_3 is the same for the real protocol and the simulator. Thus in total the distinguishing probability is at most

$$\frac{n_{\rm ro}}{q} + 2 \cdot \frac{2q^2}{q^3} + 2 \cdot \frac{2q^2\tilde{N}}{q^3\tilde{N}} + \frac{2q^6}{q^7} + \frac{2q^6\tilde{N}}{q^7\tilde{N}}$$
$$= \frac{n_{\rm ro}}{q} + \frac{12}{q} \leq (n_{\rm ro} + 12)2^{-\kappa+1}.$$

The bound in the lemma comes from the fact that this simulator is called $n_{\rm pr}$ times.

References

1. Benaloh J (1994) Dense probabilistic encryption. In: Proc. workshop on selected areas of cryptography, pp 120–128
2. Barić N, Pfitzmann B (1997) Collision-free accumulators and fail-stop signature schemes without trees. In: Proc. EUROCRYPT '96. LNCS, vol 1233. Springer, Berlin Heidelberg New York, pp 480–494
3. Blum M, DeSantis A, Micali S, Persiano G (1991) Noninteractive zero-knowledge. SIAM J Comput 20(6):1084–1118
4. Blum M, Feldman P, Micali S (1988) Non-interactive zero knowledge and its applications. In: Proc. 20th ACM symposium on theory of computing, pp 103–112
5. Boyar J, Friedl K, Lund C (1991) Practical zero-knowledge proofs: giving hints and using deficiencies. J Cryptol 4(3):185–206
6. Boyd C (1986) Digital multisignatures. In: Beker HJ, Piper FC (eds) Cryptography and coding. Clarendon Press, Oxford, UK, pp 241–246
7. Bellare M, Rogaway P (1993) Random oracles are practical: A paradigm for designing efficient protocols. In: Proc. 1st ACM conference on computer and communications security, November 1993, pp 62–73
8. Croft RA, Harris SP (1989) Public-key cryptography and reusable shared secrets. In: Baker H, Piper F (eds) Cryptography and coding, Clarendon Press, Oxford, UK, pp 189–201
9. Camenisch J, Michels M (1999) Separability and efficiency for generic group signature schemes. In: Proc. CRYPTO'99. LNCS, vol 1666. Springer, Berlin Heidelberg New York, pp 413–430
10. Cerecedo M, Matsumoto T, Imai H (1993) Efficient and secure multiparty generation of digital signatures based on discrete logarithms. IEICE Trans Fundament Electron Commun Comput Sci E76A(4):532–545
11. Cramer R, Damgård I, Schoenmakers B (1994) Proofs of partial knowledge and simplified design of witness hiding protocols. In: Proc. CRYPTO '94. LNCS, vol 839. Springer, Berlin Heidelberg New York, pp 174–187
12. Desmedt Y (1987) Society and group oriented cryptography: a new concept. In: Proc. CRYPTO '87, LNCS, vol 293. Springer, Berlin Heidelberg New York, pp 120–127
13. De Santis A, Di Crescenzo G, Ostrovsky R, Persiano G, Sahai A (2001) Robust non-interactive zero knowledge. In: Proc. CRYPTO 2001. LNCS, vol 2139. Springer, Berlin Heidelberg New York, pp 566–598
14. Desmedt Y, Frankel Y (1989) Threshold cryptosystems. In: Proc. CRYPTO '89. LNCS, vol 435. Springer, Berlin Heidelberg New York, pp 307–315
15. ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31:469–472
16. FIPS 180-1 (1995) Secure hash standard. Federal Information Processing Standards Publication 180-1, U.S. Dept. of Commerce/NIST, National Technical Information Service, Springfield, VA
17. FIPS 186 (1994) Digital signature standard. Federal Information Processing Standards Publication 186, U.S. Dept. of Commerce/NIST, National Technical Information Service, Springfield, VA
18. Frankel Y (1989) A practical protocol for large group oriented networks. In: Proc. EUROCRYPT '89. LNCS, vol 434. Springer, Berlin Heidelberg New York, pp 56–61
19. Frankel Y, MacKenzie P, Yung M (1999) Adaptively-secure distributed threshold public key systems. In: Proc. European symposium on algorithms. LNCS, vol 1643. Springer, Berlin Heidelberg New York, pp 4–27
20. Fujisaki E, Okamoto T (1997) Statistical zero-knowledge protocols to prove modular polynomial relations. In: Proc. CRYPTO '97. LNCS, vol 1294. Springer, Berlin Heidelberg New York, pp 16–30
21. Galil Z, Haber S, Yung M (1985) A private interactive test of a boolean predicate and minimum-knowledge public-key cryptosystems. In: Proc. 26th IEEE symposium on foundations of computer science, pp 360–371
22. Gennaro R, Jarecki S, Krawczyk H, Rabin T (1996) Robust threshold DSS signatures. In: Proc. EUROCRYPT '96. LNCS, vol 1070. Springer, Berlin Heidelberg New York, pp 354–371
23. Gennaro R, Jarecki S, Krawczyk H, Rabin T (1999) Secure distributed key generation for discrete-log based cryptosystems. In: Proc. EUROCRYPT '99. LNCS, vol 1592. Springer, Berlin Heidelberg New York, pp 295–310
24. Gennaro R, Krawczyk H, Rabin T (1997) RSA-based undeniable signatures. In: Proc. CRYPTO '97. LNCS, vol 1294. Springer, Berlin Heidelberg New York, pp 132–149
25. Gennaro R, Micciancio D, Rabin T (1998) An efficient noninteractive statistical zero-knowledge proof system for quasi-safe prime products. In: Proc. 5th ACM conference on computer and communications security, pp 67–72
26. Goldreich O, Oren Y (1994) Definitions and properties of zero-knowledge proof systems. J Cryptol 7:1–32
27. Goldwasser S, Micali S (1984) Probabilistic encryption. J Comput Sys Sci 28:270–299
28. Goldwasser S, Micali S, Rivest RL (1988) A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput 17(2):281–308
29. Harn L (1994) Group oriented (t, n) threshold digital signature scheme and digital multisignature. IEE Proc Comput Digit Tech 141(5):307–313

30. Herzberg A, Jakobsson M, Jarecki S, Krawczyk H, Yung M (1997) Proactive public-key and signature schemes. In: Proc. 4th ACM conference on computer and communications security, pp 100–110
31. Hwang T (1990) Cryptosystem for group oriented cryptography. In: Proc. EUROCRYPT '90. LNCS, vol 473. Springer, Berlin Heidelberg New York, pp 352–360
32. Jarecki S, Lysyanskaya A (2000) Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In: Proc. EUROCRYPT 2000. LNCS, vol 1807. Springer, Berlin Heidelberg New York, pp 221–242
33. Kilian J, Petrank E, Rackoff C (1998) Lower bounds for zero knowledge on the internet. In Proc. 39th IEEE symposium on foundations of computer science, pp 484–492
34. Kravitz DW (1993) Digital signature algorithm. U.S. Patent 5,231,668, 27 July 1993
35. Langford S (1995) Threshold DSS signatures without a trusted party. In Proc. CRYPTO '95. LNCS, vol 963. Springer, Berlin Heidelberg New York, pp 397–409
36. MacKenzie P, Reiter MK (2003) Networked cryptographic devices resilient to capture. Int J Inf Secur 2(1):1–20
37. MacKenzie P, Reiter MK (2003) Delegation of cryptographic servers for capture-resilient devices. Distrib Comput 16(4):307–327
38. Naccache D, Stern J (1997) A new public-key cryptosystem. In: Proc. EUROCRYPT '97. LNCS, vol 1233. Springer, Berlin Heidelberg New York, pp 27–36
39. Naor M, Yung M (1990) Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proc. 22nd ACM symposium on theory of computing, pp 427–437
40. Okamoto T, Uchiyama S (1998) A new public-key cryptosystem, as secure as factoring. In: Proc. EUROCRYPT '98. LNCS, vol 1403. Springer, Berlin Heidelberg New York, pp 308–318
41. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Proc. EUROCRYPT '99. LNCS, vol 1592. Springer, Berlin Heidelberg New York, pp 223–238
42. Park C, Kurosawa K (1996) New ElGamal type threshold digital signature scheme. IEICE Trans Fundament Electron Commun Comput Sci E79A(1):86–93
43. Pedersen T (1991) A threshold cryptosystem without a trusted party. In: EUROCRYPT '91, LNCS, vol 547, pp 522–526
44. Pointcheval D, Stern J (1996) Security proofs for signature schemes. In: Proc. EUROCRYPT '96. LNCS, vol 1070. Springer, Berlin Heidelberg New York, pp 387–398
45. Pointcheval D, Stern J (2000) Security arguments for digital signatures and blind signatures. J Cryptol 13(3):361–396
46. Poupard G, Stern J (2000) Short proofs of knowledge for factoring. In: Proc. Public Key Cryptosystems, PKC 2000. LNCS, vol 1751. Springer, Berlin Heidelberg New York, pp 147–166
47. Tompa M, Woll H (1987) Random self-reducibility and zero-knowledge interactive proofs of possession of information. In: Proc. 28th IEEE symposium on foundations of computer science, pp 472–482
48. van de Graaf J, Peralta R (1987) A simple and secure way to show the validity of your public key. In: Proc. CRYPTO '87. LNCS, vol 293. Springer, Berlin Heidelberg New York, pp 128–134
49. Yao A (1982) Protocols for secure computation. In: Proc. 23rd IEEE symposium on foundations of computer science, pp 160–164

Appendix : Initialization protocol

The requirements of the initialization protocol for our S-DSA system are described in properties I1–I3 of Sect. 4.1. As mentioned there, a trusted third party can be used to achieve these initialization properties in a straightforward way, though we would prefer to achieve I1–I3 without relying on a trusted party. In this section we describe such an initialization protocol for the S-DSA system that runs between alice and bob. We require that this initialization protocol be run before any S-DSA signature sessions and that it be run sequentially. (Naturally, after the initialization, signature sessions may still be run concurrently.)

A.1 Definitions

The only additional definition that we require to present our protocol is that of a zero-knowledge proof of knowledge.

– **Zero-knowledge proofs of knowledge**: A zero-knowledge proof of knowledge system (ZKPK system) Ψ for an NP language L_R, with witness relation R, is a tuple $(\mathcal{P}, \mathcal{V})$, where $\mathcal{P}$ and $\mathcal{V}$ are probabilistic polynomial-time interactive Turing machines, satisfying
  1. Completeness: For all $w \in L_R$, $\langle \mathcal{P}, \mathcal{V} \rangle(w)$ is an accepting transcript for $\mathcal{V}$.
  2. Soundness: There is a function $\textsc{Kerr}(\kappa, t)$ (*knowledge error*), a polynomial $T_{\mathcal{X}}(\kappa)$, and a probabilistic polynomial-time oracle machine $\mathcal{X}$ (*knowledge extractor*) such that the following holds: For all $w \in L_R$ with $|w| = \kappa$, and any interactive machine $\mathcal{P}'$ running in time t with success probability $\epsilon(w)$ on input w, the machine $\mathcal{X}$, having rewindable black-box access to $\mathcal{P}'$, outputs some witness v for w within an expected number of steps bounded by
$$\frac{T_{\mathcal{X}}(\kappa)}{\epsilon(w) - \textsc{Kerr}(|w|)}.$$
  3. Black-box zero-knowledge: There is a function $\textsc{Simerr}(\kappa, t)$ (*simulation error*) and a probabilistic expected[9] polynomial-time oracle machine Sim such that for every input $w \in L_R$ with $|w| = \kappa$, every probabilistic interactive machine $\mathcal{V}'$ that runs in time t, every auxiliary input y and random input r to $\mathcal{V}'$, and every probabilistic polynomial-time oracle machine Dist (the "distinguisher") such that $\mathsf{Dist}^{\mathcal{V}'_{y,r}(w)}$ runs in time t:
$$\begin{aligned}&\Pr\left(\mathsf{Dist}^{\mathcal{V}'_{y,r}(w)}(\langle \mathcal{P}, \mathcal{V}'_{y,r} \rangle(w))\right)\\ &\quad - \Pr\left(\mathsf{Dist}^{\mathcal{V}'_{y,r}(w)}(\mathsf{Sim}^{\mathcal{V}'_{y,r}(w)}(w))\right)\\ &\quad \le \textsc{Simerr}(\kappa, t).\end{aligned}$$

[9] We could require strict polynomial time, but it is sufficient for our purposes, and, moreover, the protocol of Poupard and Stern [46] is only proven secure under the weaker definition of expected polynomial time.

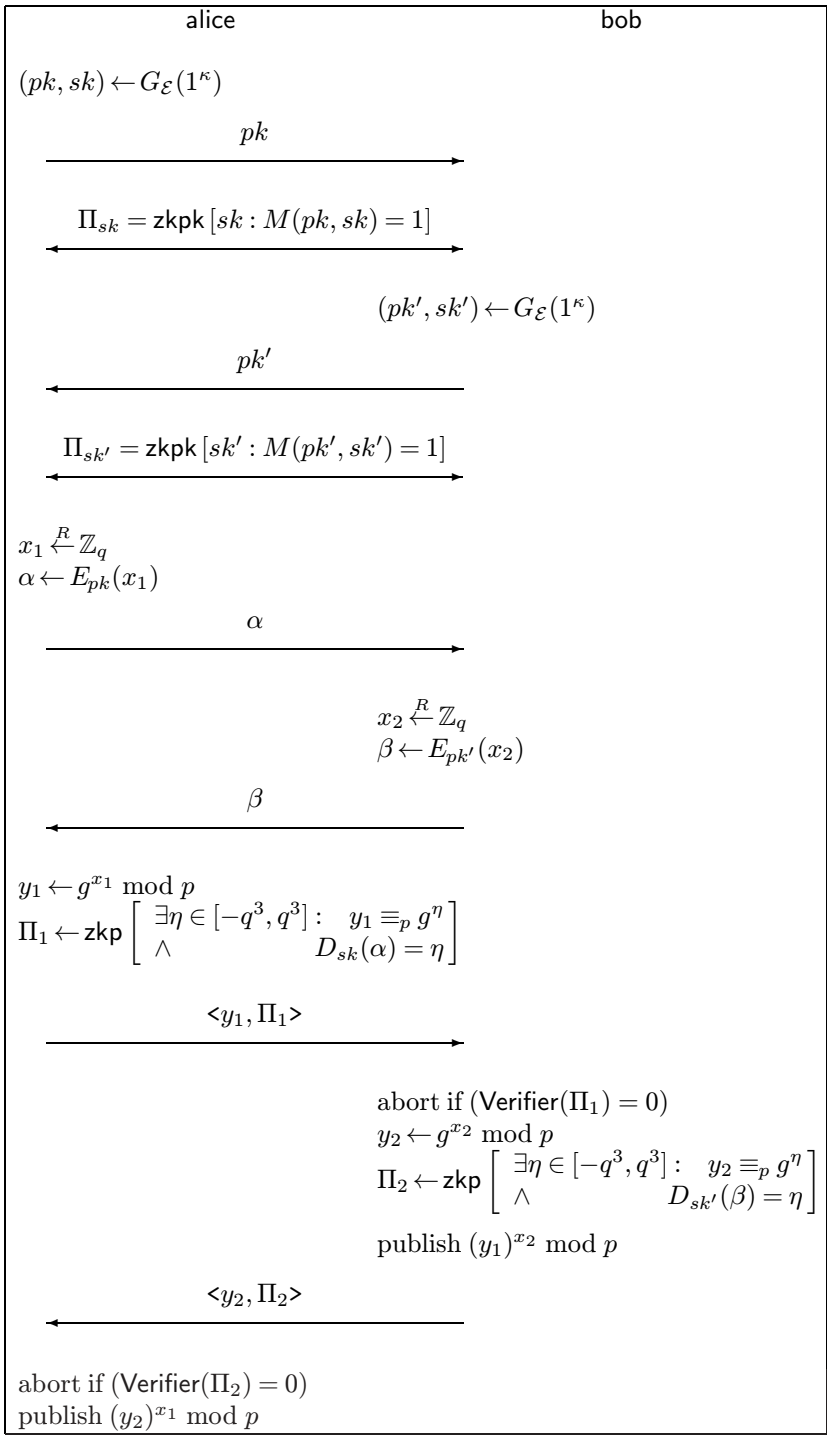

Fig. 8. S-DSA initialization protocol

> In other words, the transcript of $\langle \mathcal{P}, \mathcal{V}'_{y,r} \rangle(w)$ can be distinguishied from that of $\mathsf{Sim}^{\mathcal{V}'_{y,r}(w)}(w)$ with probability at most $\textsc{Simerr}(\kappa, t)$, even when the distinguisher is allowed black-box access to the machine $\mathcal{V}'_{y,r}(w)$.

We denote a zero-knowledge proof of knowledge of a value w satisfying a predicate P (i.e., that w is in the language of elements satisfying P) by $\mathsf{zkpk}\,[w : P]$.

A.2 Protocol

The protocol for performing initialization, achieving properties I1, I2, and I3 from Sect. 4.1, is shown in Fig. 8. This protocol takes public DSA parameters <g, p, q> as input and produces public values y, y_1, and y_2, with x shared between alice and bob as stated in I2. Though the value y is not mentioned explicitly in Fig. 8, it is taken as the value published by alice and bob (assum-

ing they publish the same value) when each completes the protocol. The protocol is initiated by alice, who generates an encryption key pair (pk, sk) and proves knowledge of sk using proof Π_{sk}. Then bob generates an encryption key pair (pk', sk') and proves knowledge of sk' using proof $\Pi_{sk'}$. Next, alice generates her secret x_1 and an encryption α of x_1, and sends α to bob. Analogously, bob generates his secret x_2 and an encryption β of x_2, and sends β to alice. Finally, alice generates $y_1 \equiv_p g^{x_1}$ and a proof Π_1 that α encrypts the discrete log of y_1, and sends y_1 and Π_1 to bob. Analogously, bob generates $y_2 \equiv_p g^{x_2}$ and a proof Π_2 that β encrypts the discrete log of y_2 and sends y_2 and Π_2 to alice.

The protocol of Fig. 8 employs interactive zero-knowledge proofs of knowledge (Π_{sk} and $\Pi_{sk'}$) and noninteractive zero-knowledge proofs of consistency (Π_1 and Π_2). These proofs are dependent on the form of sk and sk'. If we adopt the Paillier cryptosystem as in Sect. 6, and so $pk = <N, g>$, alice can prove that $N = PQ$ for primes P and Q by first proving in zero-knowledge that there are primes P, Q and values r, s such that $N = P^r Q^s$ (e.g., using [21], [48], or [25]), and then proving that $\gcd(N, \lambda(N)) = 1$ (e.g., using [5] or [25]). Note that $\gcd(N, \lambda(N)) = 1$ implies that $r = s = 1$. In addition, any proof obligations regarding g can be discharged if g is simply set to $g = N + 1$. Thus, it is not difficult for alice to give a zero-knowledge proof that $pk = <N, g>$ is well formed. Finally, alice can prove knowledge of P and Q using either the protocol of [47] or the more efficient protocol of [46]. The proofs Π_1 and Π_2 can be implemented using straightforward adaptations of the techniques in proofs Π and Π' (Sect. 6) and are omitted.

We remind the reader that Π_{sk} and $\Pi_{sk'}$ are interactive proofs of knowledge. By making these interactive (instead of using random oracles to make them noninteractive), we force each party to fix a public key before completing the proof. This facilitates our security simulation (Sect. A.3) by allowing extraction of secret keys for public keys generated according to the correct distribution. The black-box zero-knowledge property of these proofs also facilitates our simulation, as it implies that the simulator for alice (resp. bob) could effectively replace alice (resp. bob) in the actual protocol, when running against a given bob-compromising (resp. alice-compromising) adversary.

A.3 Simulatability

For our initialization protocol to be secure, we show that there is an expected polynomial-time simulator[10] for the uncompromised party that takes a DSA key y and public key pk^* for the encryption scheme $\mathcal{E}$ as input such that the following properties hold:

S1. The adversary cannot distinguish the real protocol from the simulated protocol when a random DSA key and a random public key from $\mathcal{E}$ are input to the simulator.

S2. If the simulated protocol completes successfully, then
   (a) The public key produced by the protocol is y;
   (b) The public key of the uncorrupted party is set to pk^*; and
   (c) The simulator outputs the secret DSA share of the compromised party and the secret key of the encryption scheme of the compromised party.

If the initialization protocol is secure in this sense, i.e., there is a simulator with properties S1 and S2, then we can replace the trusted initialization with this initialization protocol. The probability with which the simulator fails to achieve S1 and S2 augments the failure probability of the composite simulation for the resulting signature protocol as an additive term. To state this property carefully, we would also need to change the definition of security for signature schemes (and encryption schemes for bob-compromising adversaries) to consider forgers (attackers) that run in expected time t, rather than strict time t.

The simulator for alice plays the part of alice in the real initialization, except that it performs the operations in Fig. 9. The simulator for bob plays the part of bob in the real initialization, except that it performs the operations in Fig. 10.

[10] This simulator has the ability to "rewind" the adversary, since we assume that initialization is performed sequentially.

Set $pk \leftarrow pk^*$.
Simulate Π_{sk} and halt if the simulation fails.
Extract sk' using the extractor for $\Pi_{sk'}$ and halt if extraction fails.
Set $\alpha \leftarrow E_{pk}(0)$.
Decrypt β to obtain x_2.
Halt if Π_2 is a fraudulent proof.
Set $y_1 \leftarrow y^{(x_2)^{-1}} \bmod p$.
Simulate Π_1 and halt if the simulation fails.
Publish y.
Output x_2 and sk'.

Fig. 9. Simulator for alice initialization

Extract sk using the extractor for Π_{sk} and halt if extraction fails.
Set $pk' \leftarrow pk^*$.
Simulate $\Pi_{sk'}$ and halt if the simulation fails.
Decrypt α to obtain x_1.
Set $\beta \leftarrow E_{pk'}(0)$.
Set $y_2 \leftarrow y^{(x_1)^{-1}}$.
Simulate Π_2 and halt if the simulation fails.
Publish y.
Output x_1 and sk.

Fig. 10. Simulator for bob initialization

Property S2 stated above is obviously satisfied by these simulators. Finally, we must prove that property S1 holds.

Lemma 3. *Consider a* bob*-compromising adversary that runs in time t and completes the initialization protocol with probability $\epsilon \geq 2\textsc{Kerr}_{\Pi_{sk'}}(\kappa', t) + \textsc{Simerr}_{\Pi_{sk}}(\kappa', t)$. Then the simulator for* alice *initialization runs in expected time $O(T_{\mathcal{X}}(\kappa') + t_{exp})$ and can be distinguished from the real initialization protocol with probability at most*

$$\textsc{Simerr}_{\Pi_{sk}}(\kappa', t) + \textsc{Serr}_{\Pi_2}(\kappa, t, q_{\mathsf{hash}_{\Pi_2}}) + \textsc{Simerr}_{\Pi_1}(\kappa, q_{\mathsf{hash}_{\Pi_1}}, 1) + \mathit{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t).$$

Proof sketch: Let I_0 be the alice initialization protocol. As in our proofs for S-DSA, the idea behind the proof is to construct a series of alice simulators $I_1, I_2, \ldots$ related to I_0 and such that we eventually come to an I_j that is perfectly indistinguishable from the alice simulator of Fig. 9. At each step, we relate the probability of the bob-compromising adversary distinguishing I_j from I_{j+1} to the simulation or soundness errors of some zero-knowledge proof or to the probability of breaking the encryption scheme.

1. Let I_1 be like I_0, except that I_1 sets $pk \leftarrow pk^*$ and simulates Π_{sk}. Then I_1 can be distinguished from I_0 with probability at most $\textsc{Simerr}_{\Pi_{sk}}(\kappa', t)$.
2. Let I_2 be like I_1, except that if $\Pi_{sk'}$ succeeds, I_2 extracts sk' using the extractor for $\Pi_{sk'}$. Since the adversary completes initialization in I_1 with probability at least $\epsilon - \textsc{Simerr}_{\Pi_{sk}} \geq 2\textsc{Kerr}_{\Pi_{sk'}}(\kappa', t)$, and so succeeds in generating $\Pi_{sk'}$ with some probability $\epsilon' \geq \epsilon - \textsc{Simerr}_{\Pi_{sk}}$, this extraction adds expected time at most
$$\epsilon' \left(\frac{T_{\mathcal{X}}(\kappa')}{\epsilon' - \textsc{Kerr}_{\Pi_{sk'}}(\kappa', t)} \right) \leq \epsilon' \left(\frac{T_{\mathcal{X}}(\kappa')}{\epsilon'/2} \right) = 2T_{\mathcal{X}}(\kappa')$$
to the simulation.
3. Let I_3 be like I_2, except that I_3 decrypts β to obtain x_2 and halts if Π_2 is a fraudulent proof. Then I_3 can be distinguished from I_2 with probability at most $\textsc{Serr}_{\Pi_2}(\kappa, t, q_{\mathsf{hash}_{\Pi_2}})$.
4. Let I_4 be like I_3, except that I_4 simulates Π_1 and halts if the simulation fails. Then I_4 can be distinguished from I_3 with probability at most $\textsc{Simerr}_{\Pi_1}(\kappa, q_{\mathsf{hash}_{\Pi_1}}, 1)$.
5. Let I_5 be like I_4, except that the simulator sets $\alpha \leftarrow E_{pk}(0)$. Then I_5 can be distinguished from I_4 with probability at most $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t)$. To see this, take a public key pk and a test oracle, use pk as the public key for alice, and run I_4 except setting α to be the output of the test oracle for pk with inputs x_1 and 0. Note that this is equivalent to I_4 when x_1 is used, and I_5 when 0 is used.
6. Let I_6 be like I_5, except that y is chosen randomly from $\{g^x \bmod p\}_{x \in \mathbb{Z}_{q^*}}$ and $y_1 \leftarrow y^{(x_2)^{-1}} \bmod p$. Then I_6 is perfectly indistinguishable from I_5.

Now I_6 is perfectly indistinguishable from the simulator for alice. □

Lemma 4. *Consider an* alice*-compromising adversary that runs in time t and completes initialization with probability $\epsilon \geq 2\textsc{Kerr}_{\Pi_{sk}}(\kappa', t)$. Then the simulator for* bob *initialization runs in expected time $O(T_{\mathcal{X}}(\kappa') + t_{\mathsf{exp}})$ and can be distinguished from the real initialization protocol with probability at most*

$$\textsc{Simerr}_{\Pi_{sk'}}(\kappa', t) + \textsc{Serr}_{\Pi_1}(\kappa, t, q_{\mathsf{hash}_{\Pi_1}}) + \textsc{Simerr}_{\Pi_2}(\kappa, q_{\mathsf{hash}_{\Pi_2}}, 1) + \mathit{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t).$$

Proof sketch: Let I_0 be the bob initialization protocol. As in our previous proof, our approach is to construct a series of bob simulator $I_1, I_2, \ldots$ related to I_0 and such that we eventually come to an I_j that is perfectly indistinguishable from the bob simulator of Fig. 10. At each step, we relate the probability of the alice-compromising adversary distinguishing I_j from I_{j+1} to the simulation or soundness errors of some zero-knowledge proof or to the probability of breaking the encryption scheme.

1. Let I_1 be like I_0, except that if Π_{sk} succeeds, the simulator extracts sk using the extractor for Π_{sk}. Since the adversary completes initialization in I_0 with probability $\epsilon \geq 2\textsc{Kerr}_{\Pi_{sk}}(\kappa', t)$, and so succeeds in generating Π_{sk} with at least that probability, this extraction adds expected time $2T_{\mathcal{X}}(\kappa')$ to the simulation.
2. Let I_2 be the I_1 protocol, except that the simulator sets $pk' \leftarrow pk^*$ and simulates $\Pi_{sk'}$. Then I_2 can

be distinguished from I_1 with probability at most $\textsc{Simerr}_{\Pi_{sk'}}(\kappa', t)$.

3. Let I_3 be the I_2 protocol, except that the simulator decrypts α using sk to obtain x_1 and halts if Π_1 is a fraudulent proof. Then I_3 can be distinguished from I_2 with probability at most $\textsc{Serr}_{\Pi_1}(\kappa, t, q_{\mathsf{hash}_{\Pi_1}})$.
4. Let I_4 be the I_3 protocol, except that the simulator simulates Π_2 and halts if the simulation fails. Then I_4 can be distinguished from I_3 with probability at most $\textsc{Simerr}_{\Pi_2}(\kappa, q_{\mathsf{hash}_{\Pi_2}}, 1)$.
5. Let I_5 be the I_4 protocol, except that the simulator sets $\beta \leftarrow E_{pk'}(0)$. Then I_5 can be distinguished from I_4 with probability at most $\mathsf{Adv}^{\mathrm{ss}}_{\mathcal{E},\kappa}(t)$. To see this, take a public key pk' and a test oracle, use pk' as the public key for bob, and run I_4 except setting β to be the output of the test oracle for pk' with inputs x_2 and 0. Note that this is equivalent to I_4 when x_2 is used, and I_5 when 0 is used.
6. Let I_6 be the I_5 protocol, except that y is chosen randomly from $\{g^x \bmod p\}_{x \in \mathbb{Z}_q{}^*}$ at the start of the protocol and $y_2 \leftarrow y^{(x_1)^{-1}} \bmod p$. Then I_6 is perfectly indistinguishable from I_5.

Now I_6 is perfectly indistinguishable from the simulator for bob. □