Two-Party Generation of DSA Signatures
(Extended Abstract)

Philip MacKenzie and Michael K. Reiter

Bell Labs, Lucent Technologies, Murray Hill, NJ, USA

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.

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 [25] 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., [26,7,16]), shared generation of DSA signatures tends to be more complicated than shared generation of many other types of ElGamal-based signatures [10] 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 [20] 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

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

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

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

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.

J. Kilian (Ed.): CRYPTO 2001, LNCS 2139, pp. 137–154, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Of course, protocols for generic secure two-party computation (e.g., [34]) 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) [31] and the strong RSA assumption [2] when executed sequentially, or one that is provably secure in the random oracle model [5] 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.

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 [27]. 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 [27] 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 [4], Desmedt [8], Croft and Harris [6], Frankel [13], and Desmedt and Frankel [9]. Work in threshold cryptography for discrete-log based cryptosystems other than DSA is due to Desmedt and Frankel [9], Hwang [22], Pedersen [33], Harn [20], Park and Kurosawa [32], Herzberg *et al.* [21], Frankel *et al.* [14], and Jarecki and Lysyanskaya [23].

Several works have developed techniques directly for shared generation of DSA signatures. Langford [26] 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.* [7] and Gennaro *et al.* [16] 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 [26, Section 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 Section 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) [26, Section 5.2], thereby precluding this solution from being used in the two-party case. In contrast, our solution suffices for the two-party case without requiring the players to store precomputed, per-signature values. Since our motivating application naturally admits a trusted party for initializing the system (see [27]), for the purposes of this extended abstract we assume a trusted party to initialize alice and bob with shares of the private signing key. In the full version of this paper, we will describe the additional machinery needed to remove this assumption.

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 use $\kappa' > \kappa$ as a secondary security parameter for public key modulus size; reasonable values today may be $\kappa' = 1024$ or $\kappa' = 2048$.

Signature schemes. A *digital signature scheme* is a triple (G_{sig}, S, V) of algorithms, the first two being probabilistic, and all running in expected polynomial time. G_{sig} takes as input $1^{\kappa'}$ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_{sig}(1^{\kappa'})$. 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 [25] was proposed by NIST in April 1991, and in May 1994 was adopted as a standard digital signature scheme in the U.S. [12]. It is a variant of the ElGamal signature scheme [10], and is defined as follows, with $\kappa = 160$, κ' set to a multiple of 64 between 512 and 1024, inclusive, and hash function hash defined as SHA-1 [11]. Let "$z \leftarrow_R S$" denote the assignment to z of an element of S selected uniformly at random. Let $\equiv_q$ denote equivalence modulo q.

$G_{DSA}(1^{\kappa'})$: 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 $x \leftarrow_R \mathbb{Z}_q$ 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 $k \leftarrow_R \mathbb{Z}_q$ and ephemeral public key $r \leftarrow g^k \bmod p$. Compute $s \leftarrow k^{-1}(\mathsf{hash}(m) + xr) \bmod q$. Return $<r \mod q, s>$ as the signature of m.

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

Encryption schemes. An *encryption scheme* is a triple (G_{enc}, E, D) of algorithms, the first two being probabilistic, and all running in expected polynomial time. G_{enc} takes as input $1^{\kappa'}$ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow G_{enc}(1^{\kappa'})$. 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 and returns either a message m such that c is a valid encryption of m, if such an m exists, and otherwise returns $\perp$.

Our protocol employs a semantically secure encryption scheme with a certain additive homomorphic property. For any public key pk output from the G_{enc} function, let M_{pk} be the space of possible inputs to E_{pk}, and C_{pk} to be the space of possible outputs of E_{pk}. Then we require that there exist an efficient implementation of an additional function $+_{pk} : C_{pk} \times C_{pk} \rightarrow C_{pk}$ 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 $+_{pk}$ exist (with $M_{pk} = [-v, v]$ for a certain value v) are due to Naccache and Stern [28], Okamoto and Uchiyama [30], and Paillier [31].[1] Note that (1) further implies the existence of an efficient function $\times_{pk} : C_{pk} \times M_{pk} \rightarrow C_{pk}$ 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. We denote such a proof as $\mathsf{zkp}[P]$. In Section 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.

[1] 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.

System model. Our system includes two parties, alice and bob. Communication between alice and bob occurs in *sessions* (or protocol runs), one per message that they sign together. alice plays the role of session initiator in our protocol. We presume that each message is implicitly labeled with an identifier for the session to which it belongs. Multiple 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 learns all private initialization information for alice. A bob-compromising adversary is defined similarly.

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 Signature Protocol

In this section we present a new protocol called S-DSA by which alice and bob sign a message m.

4.1 Initialization

For our signature protocol, we assume that the private key x is multiplicatively shared between alice and bob, i.e., 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 $x \equiv_q x_1 x_2$. We also assume that along with y, $y_1 = g^{x_1} \bmod p$ and $y_2 = g^{x_2} \bmod p$ are public. In this extended abstract, we do not concern ourselves with this initialization step, but simply assume it is performed correctly, e.g., by a trusted third party. We note, however, that achieving this without a trusted third party is not straightforward (e.g., see [17]), and so we will describe such an initialization protocol in the full version of this paper.

We use a multiplicative sharing of x to achieve greater efficiency than 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.

In addition to sharing x, our protocol assumes that alice holds the private key sk corresponding to a public encryption key pk, and that there is another public encryption key pk' for which alice does not know the corresponding sk'. (As above, we assume that these keys are generated correctly, e.g., by a trusted third party.) Also, it is necessary for our particular zero-knowledge proof constructions 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 Figure 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).

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 modulo $\mathbb{Z}_p^*$), 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 $\mathsf{hash}(m)$, $z_2 = (k_2)^{-1} \bmod q$, $r \bmod q$, 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 usual completeness, soundness, and zero-knowledge properties as defined in [3,29], except using a public random hash function (i.e., a random oracle) instead of a public random string. In particular, we assume in Section 5 that (1) these proofs have negligible simulation error probability, and in fact a simulator exists that generates a proof that is statistically indistinguishable from a proof generated by the real prover, and (2) these proofs have negligible soundness error probability, i.e., the probability that a prover could generate a proof for a false statement is negligible. The implementations of Π and Π' in Section 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 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 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., [24]).

5 Security for S-DSA

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

5.1 Security for DSA and Encryption

First we state requirements for security of DSA and encryption. For DSA, we specify existential unforgeability versus chosen message attacks [19]. That is, a forger is given $<g, p, q, y>$, where $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{DSA}(1^{\kappa'})$, and tries to forge signatures with respect to $<g, p, q, y>$. It is allowed to query a signature oracle (with respect to $<g, p, q, x>$) on messages of its choice. It succeeds if after this it can output some (m, σ) where $V_{<g,p,q,y>}(m, \sigma) = 1$ but m was not one of the messages signed by the signature oracle. We say a forger (q, ϵ)-breaks DSA if the forger makes q queries to the signature oracle and succeeds with probability at least ϵ.

For encryption, we specify semantic security [18]. That is, an attacker A is given pk, where $(pk, sk) \leftarrow G_{enc}(1^{\kappa'})$. A generates $X_0, X_1 \in M_{pk}$ and sends these to a test oracle, which chooses $b \leftarrow_R \{0, 1\}$, and returns $Y = E_{pk}(X_b)$. Finally A outputs b', and succeeds if $b' = b$. We say an attacker A ϵ-breaks encryption if $2 \cdot \Pr(A \text{ succeeds}) - 1 \geq \epsilon$. Note that this implies $\Pr(A \text{ guesses } 0 \mid b = 0) - \Pr(A \text{ guesses } 0 \mid b = 1) \geq \epsilon$.

5.2 Security for S-DSA

A forger F is given $<g, p, q, y>$, where $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{DSA}(1^{\kappa'})$, and the public data generated by the initialization procedure for S-DSA, along with the secret data of either alice or bob (depending on the type of forger). As in the security definition for signature schemes, the goal of the forger is to forge signatures with respect to $<g, p, q, y>$. Instead of a signature oracle, there is an alice oracle and a bob oracle.

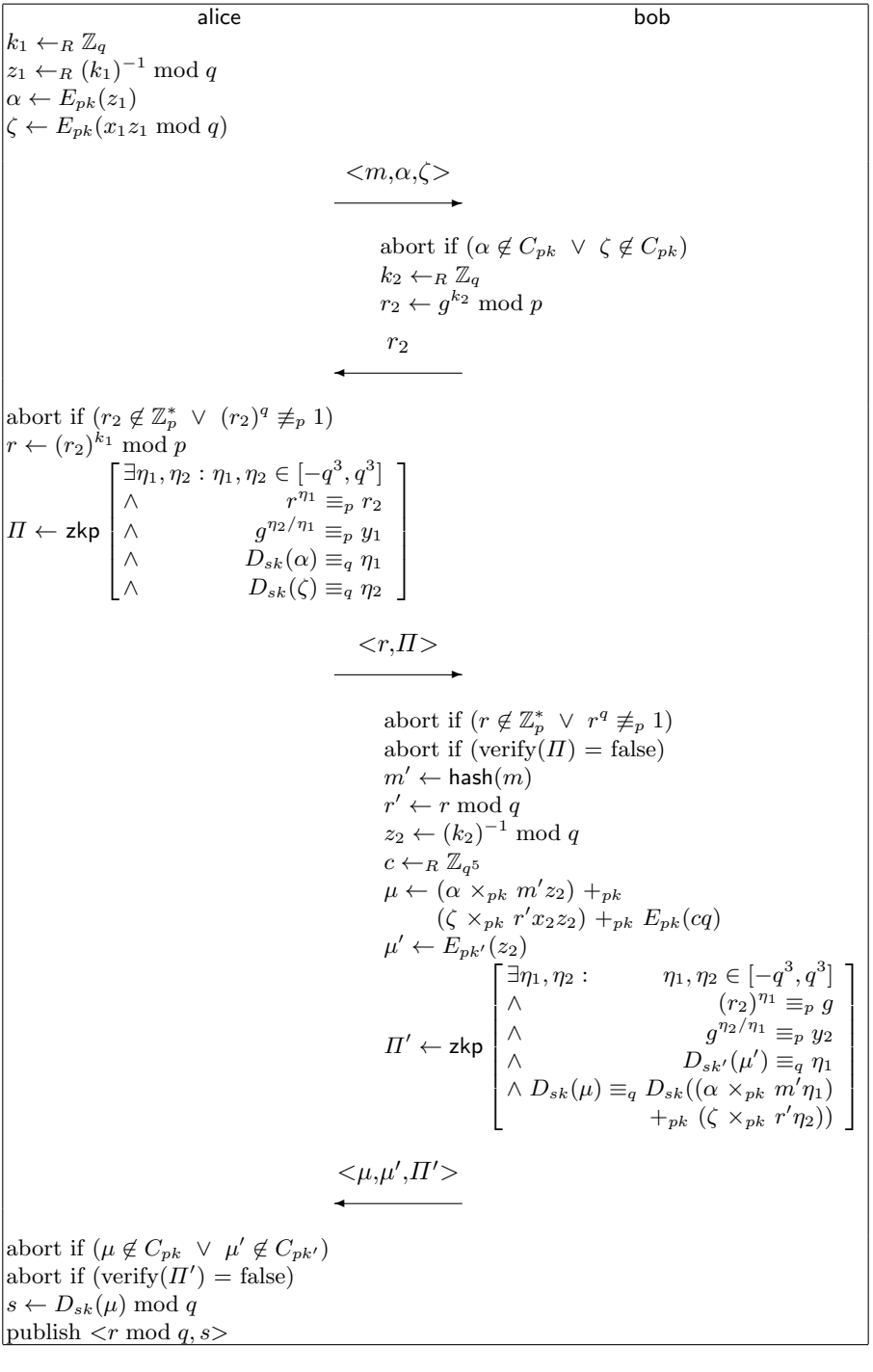

Fig. 1. S-DSA shared signature protocol

F may query the alice oracle by invoking *aliceInv1*(m), *aliceInv2*(r_2), or *aliceInv3*($<\mu, \mu', \Pi'>$) for input parameters of F's choosing. (These invocations are also accompanied by a session identifier, which is left implicit.) These invocations correspond to a request to initiate the protocol for message m and the first and second messages received ostensibly from bob, respectively. These return outputs of the form $<m, \alpha, \zeta>$, $<r, \Pi>$, or a signature for the message m from the previous *aliceInv1* query in the same session, respectively, or abort. Analagously, F may query the bob oracle by invoking *bobInv1*($<m, \alpha, \zeta>$) or *bobInv2*($<r, \Pi>$) for arguments of the F's choosing. These return messages of the form r_2 or $<\mu, \mu', \Pi'>$, respectively, or abort. F may invoke these queries in any order, arbitrarily many times.

An alice-compromising forger F *succeeds* if after gaining access to the private initialization state of alice, and invoking the alice and bob oracles 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 *bobInv1* query. Similarly, a bob-compromising forger F *succeeds* if after gaining access to the private initialization state of bob, and invoking the alice and bob oracles 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 a *aliceInv1* query.

Let q_{alice} be the number of *aliceInv1* queries to alice. Let q_{bob} be the number of *bobInv1* queries. Let q_o be the number of other oracle queries. Let $\overline{q} = <q_{\mathsf{alice}}, q_{\mathsf{bob}}, q_o>$. In a slight abuse of notation, let $|\overline{q}| = q_{\mathsf{alice}} + q_{\mathsf{bob}} + q_o$, i.e., the total number of oracle queries. We say a forger $(\overline{q}, \epsilon)$-breaks S-DSA if it makes $|\overline{q}|$ oracle queries (of the respective type and to the respective oracles) and succeeds with probability at least ϵ.

5.3 Theorems

Here we state theorems and provide proof sketches showing that if a forger breaks the S-DSA system with non-negligible probability, then either DSA or the underlying encryption scheme used in S-DSA can be broken with non-negligible probability. This implies that if DSA and the underlying encryption scheme are secure, our system will be secure.

We prove security separately for alice-compromising and bob-compromising forgers. The idea behind each proof is a simulation argument. Assuming that a forger F can break the S-DSA system, we then construct a forger F^* that breaks DSA. Basically F^* will run F over a simulation of the S-DSA system, and when F succeeds in forging a signature in the simulation of S-DSA, then F^* will succeed in forging a DSA signature.

In the security proof against an alice-compromising forger F, there is a slight complication. If F were able to break the encryption scheme (G_{enc}, E, D), an attacker F^* as described above may not be able to simulate properly. Thus we show that either F forges signatures in a simulation where the encryptions are of strings of zeros, and thus we can construct a forger F^* for DSA, or F does not forge signatures in that simulation, and thus it must be able to distinguish the true encryptions from the zeroed encryptions. Then we can construct an attacker A that breaks the underlying encryption scheme. A similar complication arises

in the security proof against a bob-compromising forger F, and the simulation argument is modified in a similar way.

Theorem 1 below states that an alice-compromising forger that breaks S-DSA with a non-negligible probability can break either DSA or (G_{enc}, E, D) with non-negligible probability. Theorem 2 makes a similar claim for a bob-compromising forger. In these theorems, we use "$\approx$" to indicate equality to within negligible factors. Moreover, in our simulations, the forger F is run at most once, and so the times of our simulations are straightforward and omitted from our theorem statements.

Theorem 1. *Suppose an* alice*-compromising forger* $(\overline{q}, \epsilon)$*-breaks* S-DSA*. Then either there exists an attacker that* ϵ'*-breaks* (G_{enc}, E, D) *with* $\epsilon' \approx \frac{\epsilon}{2q_{\mathsf{bob}}}$*, or there exists a forger that* $(q_{\mathsf{bob}}, \epsilon'')$*-breaks DSA with* $\epsilon'' \approx \frac{\epsilon}{2}$*.*

Proof. Assume an alice-compromising forger F $(\overline{q}, \epsilon)$-breaks the S-DSA scheme. Then consider a simulation Sim of the S-DSA scheme that takes as input a DSA public key $<g, p, q, y>$, a corresponding signature oracle, and a public key pk' for the underlying encryption scheme. Sim generates the initialization data for alice: $x_1 \leftarrow_R \mathbb{Z}_q$ and $(pk, sk) \leftarrow G_{enc}(1^{\kappa'})$, and gives these to F. The public data y, $y_2 = g^{(x_1^{-1} \bmod q)} \bmod p$, and pk' are also revealed to F. Then Sim responds to alice queries as a real alice oracle would, and to bob queries using the help of the DSA signature oracle, since Sim does not know the x_2 value used by a real bob oracle. Specifically Sim answers as follows:

1. *bobInv1*$(<m, \alpha, \zeta>)$: Set $z_1 \leftarrow D_{sk}(\alpha)$. Query the DSA signature oracle with m to get a signature $<\hat{r}, \hat{s}>$, and compute $r \leftarrow g^{\mathsf{hash}(m)\hat{s}^{-1}} y^{\hat{r}\hat{s}^{-1}} \bmod p$ where $\hat{s}^{-1}$ is computed modulo q. Compute $r_2 \leftarrow r^{z_1} \bmod p$, and return r_2.
2. *bobInv2*$(<r, \Pi>)$: Reject if Π is invalid, $r \notin \mathbb{Z}_p^*$ or $r^q \not\equiv_p 1$. Else, choose $c \leftarrow_R \mathbb{Z}_{q^5}$ and set $\mu \leftarrow E_{pk}(\hat{s} + cq)$. Set $\mu' \leftarrow E_{pk'}(0)$, and generate Π' using the simulator for the zkp []. Return $<\mu, \mu', \Pi'>$.

Notice that Sim sets μ' to an encryption of zero, and simulates the proof of consistency Π'. In fact, disregarding the negligible statistical difference between the simulated Π' proofs and the real Π' proofs, the only way Sim and the real S-DSA scheme differ (from F's viewpoint) is with respect to the μ' values, i.e., the (at most) q_{bob} ciphertexts generated using pk'.

Now consider a forger F^* that takes as input a DSA public key $<g, p, q, y>$ and corresponding signature oracle, generates a public key pk' using $<pk', sk'> \leftarrow G_{enc}(1^{\kappa'})$, runs Sim using these parameters as inputs, and outputs whatever F outputs. If F produces a forgery with probability at least $\frac{\epsilon}{2}$ in Sim, F^* produces a forgery in the underlying DSA signature scheme with probability at least $\frac{\epsilon}{2}$.

Otherwise F produces a forgery with probability less than $\frac{\epsilon}{2}$ in Sim. Then using a standard hybrid argument, we can construct an attacker A that ϵ'-breaks the semantic security of the underlying encryption scheme for pk', where $\epsilon' \approx \frac{\epsilon}{2q_{\mathsf{bob}}}$. Specifically, A takes a public key pk' and corresponding test oracle

as input, generates a DSA public/private key pair $(<g,p,q,y>, <g,p,q,x>) \leftarrow G_{DSA}(1^{\kappa'})$, and runs a slightly modified SIM using $<g,p,q,y>$ as the DSA public key parameter, simulating the DSA signature oracle with $<g,p,q,x>$, and using pk' as the public encryption key parameter. SIM is modified only in the *bobInv2* query, as follows:

1. A computes the value $z_2 \leftarrow (kz_1)^{-1} \bmod q$, where k was computed in the simulation of the DSA signature oracle in the corresponding *bobInv1* query,
2. A chooses to produce the first j ciphertexts under pk' as in the real protocol (i.e., $\mu' \leftarrow E_{pk'}(z_2)$), for a random $j \in \{0, \ldots, q_{\mathsf{bob}}\}$, and
3. A produces the next ciphertext under pk' by using the response from the test oracle with input $X_0 = z_2$ and $X_1 = 0$.

Finally A outputs 0 if F produces a forgery, and 1 otherwise. Since the case of $j = 0$ corresponds to SIM, and the case of $j = q_{\mathsf{bob}}$ corresponds to the real protocol, an averaging argument can be used to show that A ϵ'-breaks the semantic security of the underlying encryption scheme for pk' with probability $\epsilon' \approx \frac{\epsilon}{2q_{\mathsf{bob}}}$.

Theorem 2. *Suppose a* bob*-compromising forger* $(\overline{q}, \epsilon)$*-breaks* S-DSA*. Then either there exists an attacker that* ϵ'*-breaks* (G_{enc}, E, D) *with* $\epsilon' \approx \frac{\epsilon}{4q_{\mathsf{alice}}}$*, or there exists a forger that* $(q_{\mathsf{alice}}, \epsilon'')$*-breaks DSA, with* $\epsilon'' \approx \frac{\epsilon}{2}$*.*

Proof. Assume a bob-compromising forger F $(\overline{q}, \epsilon)$-breaks the S-DSA scheme. Then consider a simulation SIM of the S-DSA scheme that takes as input a DSA public key $<g,p,q,y>$, a corresponding signature oracle, and a public key pk for the underlying encryption scheme. SIM generates the initialization data for bob: $x_2 \leftarrow_R \mathbb{Z}_q$ and $(pk', sk') \leftarrow G_{enc}(1^{\kappa'})$, and gives these to F. The public data y, $y_1 = g^{(x_2^{-1} \bmod q)} \bmod p$, and pk are also revealed to F. Then SIM responds to bob queries as a real bob oracle would, and to alice queries using the help of the DSA signature oracle, since SIM does not know the x_1 value used by a real alice oracle. Specifically SIM answers as follows:

1. *aliceInv1*(m): Set $\alpha \leftarrow E_{pk}(0)$ and $\zeta \leftarrow E_{pk}(0)$, and return $<m, \alpha, \zeta>$.
2. *aliceInv2*(r_2): Reject if $r_2 \notin \mathbb{Z}_p^*$ or $(r_2)^q \not\equiv_p 1$. Call the DSA signature oracle with m, let $(\hat{r}, \hat{s})$ be the resulting signature, and compute $r \leftarrow g^{\mathsf{hash}(m)\hat{s}^{-1}} y^{\hat{r}\hat{s}^{-1}} \bmod p$ where $\hat{s}^{-1}$ is computed modulo q. Construct Π using the simulator for the zkp []. Store $<\hat{r}, \hat{s}>$ and return $<r, \Pi>$.
3. *aliceInv3*$(<\mu, \mu', \Pi'>)$: Reject if $\mu \notin C_{pk}$, $\mu' \notin C_{pk'}$, or the verification of Π' fails. Otherwise, return $<\hat{r}, \hat{s}>$.

Notice that SIM sets α and ζ to encryptions of zero, and simulates the proof of consistency Π. In fact, disregarding the negligible statistical difference between the simulated Π proofs and the real Π proofs, the only way SIM and the real S-DSA scheme differ (from F's viewpoint) is with respect to the α and ζ values, i.e., the (at most) $2q_{\mathsf{alice}}$ ciphertexts generated using pk.

Now consider a forger F^* that takes as input a DSA public key $<g, p, q, y>$ and a corresponding signature oracle, generates a public key pk using $<pk, sk> \leftarrow G_{enc}(1^{\kappa'})$, runs SIM using these parameters as inputs, and outputs whatever F outputs. If F produces a forgery with probability at least $\frac{\epsilon}{2}$ in SIM, F^* produces a forgery in the underlying DSA signature scheme with probability at least $\frac{\epsilon}{2}$.

Otherwise F produces a forgery with probability less than $\frac{\epsilon}{2}$ in SIM. Then using a standard hybrid argument, we can construct an attacker A that ϵ'-breaks the semantic security of the underlying encryption scheme for pk, where $\epsilon' \approx \frac{\epsilon}{4q_{\mathsf{alice}}}$. Specifically, A takes a public key pk and corresponding test oracle as input, generates a DSA public/private key pair $(<g, p, q, y>, <g, p, q, x>) \leftarrow G_{DSA}(1^{\kappa'})$, and runs a slightly modified SIM using $<g, p, q, y>$ as the DSA public key parameter, and using pk as the public encryption key parameter. SIM is modified only in the alice oracle queries, as follows:

1. In *aliceInv1*,
   a) A chooses to produce the first j ciphertexts under pk as in the real protocol (i.e., either $\alpha \leftarrow E_{pk}(z_1)$ or $\zeta \leftarrow E_{pk}(x_1 z_1 \bmod q)$), for a random $j \in \{0, \ldots, 2q_{\mathsf{alice}}\}$,
   b) A produces the next ciphertext under pk by using the response from the test oracle with input X_0 being the plaintext from the real protocol (i.e., either $X_0 = z_1$ or $X_0 = x_1 z_1 \bmod q$, depending on whether j is even or odd) and $X_1 = 0$.
2. In *aliceInv2*, A computes r as in the real protocol, without calling the DSA signature oracle.
3. In *aliceInv3*, instead of returning the result of calling the DSA signature oracle, A computes $z_2 \leftarrow D_{sk'}(\mu')$ and $k_2 \leftarrow (z_2)^{-1} \bmod q$, sets $k \leftarrow k_1 k_2 \bmod q$, and returns the DSA signature for m using DSA secret key $<g, p, q, x>$ with k as the ephemeral secret key.

Finally A outputs 0 if F produces a forgery, and 1 otherwise. Since the case of $j = 0$ corresponds to SIM (in particular, notice that the distribution of r is identical), and the case of $j = 2q_{\mathsf{alice}}$ corresponds to the real protocol, an averaging argument can be used to show that A ϵ'-breaks the semantic security of the underlying encryption scheme for pk with probability $\epsilon' \approx \frac{\epsilon}{4q_{\mathsf{alice}}}$.

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'. The form of these proofs naturally depends on the encryption scheme (G_{enc}, E, D), and the particular encryption scheme for which we detail Π and Π' here is that due to Paillier [31]. 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 (see Section 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 [31]. 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 [31] only in that we define the message space M_{pk} for public key $pk = <N, g>$ as $M_{<N,g>} = [-(N-1)/2, (N-1)/2]$ (versus $\mathbb{Z}_N$ in [31]).

$G_{Pai}(1^{\kappa'})$: Choose $\kappa'/2$-bit primes p, q, set $N = pq$, and choose a random element $g \in \mathbb{Z}^*_{N^2}$ such that $\gcd(L(g^{\lambda(N)} \bmod N^2), N) = 1$. Return the public key $<N, g>$ and the private key $<N, g, \lambda(N)>$.

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

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

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

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

Paillier [31] 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.

6.2 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 Section 6.3. Both proofs rely on the following assumption:

> **Strong RSA Assumption.** Given an RSA modulus generator G_{RSA} that takes as input $1^{\kappa'}$ and produces a value N that is the product of two random primes of length $\kappa'/2$, the Strong RSA assumption states that for any probabilistic polynomial-time attacker A:
>
> $$\Pr[N \leftarrow G_{RSA}(1^{\kappa'}); y \leftarrow_R \mathbb{Z}^*_N; (x, e) \leftarrow A(N, y) : (e \geq 3) \wedge (y \equiv_N x^e)]$$
>
> is negligible.

In our proofs, it is assumed that there are public values $\tilde{N}$, h_1 and h_2. Soundness requires that $\tilde{N}$ be an RSA modulus that is the product of two strong

primes and for which the factorization is unknown to the prover, and that the discrete logs of h_1 and h_2 relative to each other modulo $\tilde{N}$ are unknown to the prover. Zero knowledge requires that discrete logs of h_1 and h_2 relative to each other modulo $\tilde{N}$ exist (i.e., that h_1 and h_2 generate the same group). As in Section 4.1, here we assume that these parameters are distributed to alice and bob by a trusted third party. In the full paper, we will describe how this assumption can be eliminated.

Now consider the proof Π. Let p and q be as in a DSA public key, $pk = <N, g>$ be a Paillier public key, and $sk = <N, g, \lambda(N)>$ be 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:

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

The proof is constructed in Figure 2, and its verification procedure is given in Figure 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. If necessary, the verifier should verify that $c, d, w_1, w_2 \in \mathbb{Z}_p^*$ and are of order q, and that $m_1, m_2 \in \mathbb{Z}_{N^2}^*$.

Intuitively, the proof works as follows. Commitments z_1 and z_2 are made to x_1 and x_2 over the RSA modulus $\tilde{N}$, and these are proven to fall in the desired range using proofs as in [15]. Simultaneously, it is shown that the commitment z_1 corresponds to the decryption of m_1 and the discrete log of w_1. Also simultaneously, it is shown 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 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. The proof of the following lemma will appear in the full version of this paper.

Lemma 1. *Π is a noninteractive zero-knowledge proof of P.*

6.3 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 $N > q^8$, and $pk' = <N', g'>$ and $sk' = <N', g', \lambda(N')>$ 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 for some $n_1, n_2 \in [-q^4, q^4]$, $D_{sk}(m_3) = n_1$ and $D_{sk}(m_4) = n_2$, we construct a zero-knowledge proof Π' of:

$$P' = \begin{bmatrix} \exists x_1, x_2, x_3 : & x_1, x_2 \in [-q^3, q^3] \\ \wedge & x_3 \in [-q^7, q^7] \\ \wedge & c^{x_1} \equiv_p w_1 \\ \wedge & d^{x_2/x_1} \equiv_p w_2 \\ \wedge & D_{sk'}(m_1) = x_1 \\ \wedge & D_{sk}(m_2) = n_1 x_1 + n_2 x_2 + q x_3 \end{bmatrix}$$

We note that P' is stronger than what is needed as shown in Figure 1. The proof is constructed in Figure 4, and the verification procedure for it is given in Figure 5. 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 $m_2 \equiv_{N^2} (m_3)^{x_1}(m_4)^{x_2} g^{q x_3}(r_2)^N$. The

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

Fig. 2. Construction of Π

$<z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4, s_1, s_2, s_3, t_1, t_2, t_3, t_4> \leftarrow \Pi$	
Verify $s_1, t_1 \in \mathbb{Z}_{q^3}$.	Verify $d^{t_1+t_2} \equiv_p y^e v_1$.
Verify $c^{s_1} \equiv_p (w_1)^e u_1$.	Verify $(w_2)^{s_1} d^{t_2} \equiv_p y^e v_2$.
Verify $g^{s_1}(s_2)^N \equiv_{N^2} (m_1)^e u_2$.	Verify $g^{t_1}(t_3)^N \equiv_{N^2} (m_2)^e v_3$.
Verify $(h_1)^{s_1}(h_2)^{s_3} \equiv_{\tilde{N}} (z_1)^e u_3$.	Verify $(h_1)^{t_1}(h_2)^{t_4} \equiv_{\tilde{N}} (z_2)^e v_4$.

Fig. 3. Verification of Π

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

Fig. 4. Construction of Π'

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

Fig. 5. Verification of Π'

prover need not know sk or sk', though a malicious prover might know sk'. We assume the verifier knows n_1 and n_2. 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 $m_2 \in \mathbb{Z}^*_{N^2}$. The proof of the following lemma will appear in the full version of this paper.

Lemma 2. *Π' is a noninteractive zero-knowledge proof of P'.*

References

1. J. Benaloh. Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, pages 120–128, 1994.
2. N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT '96* (LNCS 1233), pages 480–494, 1997.
3. M. Blum, A. DeSantis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal of Computing* 20(6):1084–1118, 1991.
4. C. Boyd. Digital multisignatures. In H. J. Beker and F. C. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, 1986.
5. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1st *ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.
6. R. A. Croft and S. P. Harris. Public-key cryptography and reusable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 189–201, 1989.
7. M. Cerecedo, T. Matsumoto, H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. *IEICE Trans. Fundamentals of Electronics Communications and Computer Sciences* E76A(4):532–545, April 1993.
8. Y. Desmedt. Society and group oriented cryptography: a new concept. In *CRYPTO '87* (LNCS 293), pages 120–127, 1987.
9. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO '89* (LNCS 435), pages 307–315, 1989.
10. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.
11. FIPS 180-1. Secure hash standard. Federal Information Processing Standards Publication 180-1, U.S. Dept. of Commerce/NIST, National Technical Information Service, Springfield, Virginia, 1995.
12. FIPS 186. Digital signature standard. Federal Information Processing Standards Publication 186, U.S. Dept. of Commerce/NIST, National Technical Information Service, Springfield, Virginia, 1994.
13. Y. Frankel. A practical protocol for large group oriented networks. In *EUROCRYPT '89* (LNCS 434), pages 56–61, 1989.
14. Y. Frankel, P. MacKenzie, and M. Yung. Adaptively-secure distributed threshold public key systems. In *European Symposium on Algorithms* (LNCS 1643), pages 4–27, 1999.
15. E. Fujisaki and T. Okamoto. Statistical zero-knowledge protocols to prove modular polynomial relations. In *CRYPTO '97* (LNCS 1294), pages 16–30, 1997.
16. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *EUROCRYPT '96* (LNCS 1070), pages 354–371, 1996.
17. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT '99* (LNCS 1592), pages 295–310, 1999.
18. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences* 28:270–299, 1984.
19. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* 17(2):281–308, April 1988.
20. L. Harn. Group oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proc.-Comput. Digit. Tech.* 141(5):307–313, 1994.

21. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public-key and signature schemes. In 4th *ACM Conference on Computer and Communications Security*, pages 100–110, 1997.
22. T. Hwang. Cryptosystem for group oriented cryptography. In *EUROCRYPT '90* (LNCS 473), pages 352–360, 1990.
23. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In *EUROCRYPT 2000* (LNCS 1807), pages 221–242, 2000.
24. J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the internet. In 39th *IEEE Symposium on Foundations of Computer Science*, pages 484–492, 1998.
25. D. W. Kravitz. Digital signature algorithm. U.S. Patent 5,231,668, 27 July 1993.
26. S. Langford. Threshold DSS signatures without a trusted party. In *CRYPTO '95* (LNCS 963), pages 397–409, 1995.
27. P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. DIMACS Technical Report 2001-19, May 2001. Extended abstract in *2001 IEEE Symposium on Security and Privacy*, May 2001.
28. D. Naccache and J. Stern. A new public-key cryptosystem. In *EUROCRYPT '97* (LNCS 1233), pages 27–36, 1997.
29. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In 22nd *ACM Symposium on Theory of Computing*, pages 427–437, 1990.
30. T. Okamoto and S. Uchiyama. A new public-key cryptosystem, as secure as factoring. In *EUROCRYPT '98* (LNCS 1403), pages 308–318, 1998.
31. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99* (LNCS 1592), pages 223–238, 1999.
32. C. Park and K. Kurosawa. New ElGamal type threshold digital signature scheme. *IEICE Trans. Fundamentals of Electronics Communications and Computer Sciences* E79A(1):86–93, January, 1996.
33. T. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT '91* (LNCS 547), pages 522–526, 1991.
34. A. Yao. Protocols for secure computation. In 23rd *IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.