Efficient Construction of Nominative Signature Secure under Symmetric Key Primitives and Standard Assumptions on Lattice

Meenakshi Kansal, Ratna Dutta and Sourav Mukhopadhyay
Department of Mathematics,
Indian Institute of Technology Kharagpur,
Kharagpur, India
Email:{kansal,ratna,sourav}@maths.iitkgp.ernet.in

Abstract

Nominative signature is a cryptographic primitive where two parties collude to produce a signature. It is a user certification system and has applications in variety of sectors where nominee cannot trust heavily on the nominator to validate nominee's certificate and only targeted entities are allowed to verify signature on sensitive data. We provide a new construction for nominative signature from standard assumptions on lattice. Our construction relies on collision resistant preimage sampleable function and symmetric key primitives like collision resistant pseudorandom function and zero knowledge proof system ZKB++ for Boolean circuits. We provide a detailed security analysis and show that our construction achieves security under *unforgeability*, *invisibility*, *impersonation* and *non-repudiation* in existing model. Furthermore, our construction exhibits *non-transferability.* The security under non-repudiation is achieved in the quantum random oracle model using Unruh transform to ZKB++.

Keywords: Nominative signature, User certification system, Zero knowledge proof, Unruh transform.

1 Introduction

User certification system. User certification system is an important mechanism that addresses privacy issues associated with the distribution of signed digital certificates. Generally speaking, a user certification system enables a user 'U' to convince a verifier 'V' that the certificate 'c' issued by an authority 'A' is authentic. Typically, the signed certificate contains the truthfulness of certain statements and attributes linked to the identity of the user to which the certificate is issued. The real life examples include the issuing of identity cards, legal permanent residential proofs, birth certificates and driving licenses. A user can present proofs to a verifier, who in turn can verify signature and be convinced of the truth of the statements contained in the certificate.

A large fraction of applications can benefit from user certification system where the data to be verified are sensitive and requires only the targeted party to verify the signature. Standard digital signatures are publicly verifiable and thereby, insufficient to apply in user certification system.

Nominative Signature. A nominative signature (NS) scheme is a user certification system without any trusted certifying authority where a nominator (nr) and a nominee (ne) jointly produce a nominative signature featuring the following properties simultaneously.

P1: The nominator or nominee 'alone' cannot produce a valid nominative signature.

P2: The nominative signature must be verifiable only by the nominee.

P3: Only the nominee can prove the validity or invalidity of the nominative signature to a public verifier.

The central advantage of using nominative signature is that the nominee does not need to trust the nominator as the nominator cannot validate nominee's certificate to a third party. We review two examples below.

- **NS in Medical Scenario.** Consider medical records of patients in a hospital. These records are sensitive and the patients do not want a third party to verify their records. At the same time, a patient might need to recover the medical expenses from the insurance company. The patient must prove the validity of his claim to the insurance company. Nominative signature can be issued on the claim jointly by a patient (nominee) and the hospital authority (nominator). This signature is verifiable only by the patient. Beside, the validity of the claim can be proved only by the patient to the insurance company (verifier).
- **NS in Tax Bill.** Suppose a government agency 'A' is tasked with monitoring the income tax. The income tax of an individual carries sensitive data, hence should not be verified by everyone. A nominative signature can be used to serve the purpose. Suppose an employer (nominator) and an employee (nominee) together generate a nominative signature tax bill of the employee. Only the employee can verify the signature. The tax amount in the tax bill is deducted by the employer from the employee's salary and is sent to the government agency 'A'. If the employee is not satisfied with the tax bill, he communicates with 'A' providing his documents and requests for the tax return. If 'A' is convinced with the documents then the agency 'A' returns the money. Otherwise rejects the employee's request. Note that NS enables only the employee to prove the validity or invalidity of the tax bill to 'A'.

Our Contribution. Our main result is to put forth a new construction of nominative signature (NS) based on the worst case hardness of lattice. The scheme is secure assuming the existence of preimage sampleable function (PSF) and symmetric key primitives such as collision resistant pseudorandom function (CRPRF) and collision resistant hash function. We integrate zero knowledge proof system $\mathsf{ZKB{+}{+}}$ using Unruh transform [2]. For security under unforgeabiliy, we adopt the security model of Huang et al. [12]. For security under invisibility, we follow the security model of Schuldt et al. [21]. Our scheme attains security in the random oracle model. Besides, our construction achieves security under non-repudiation in the quantum random oracle model $\mathsf{Q(ROM)}$ unlike [14].

Our construction derives its security from the worst-case hardness assumptions of lattice and the security of the underlying symmetric key primitives. More precisely, we have the following.

- We achieve unforgeability against malicious nominee and unforgeability against malicious nominator under the hardness of collision resistance preimage sampleable function (CRPSF). The invisibility relies on the hardness of PRF.
- Non-repudiation follows from the completeness and soundness properties of the non-interactive zero knowledge proof system $\mathsf{ZKB{+}{+}}$. We achieve the security under non-repudiation in quantum random oracle model (QROM) by integrating $\mathsf{ZKB{+}{+}}$ using Unruh transform [2].
- Similar to [21], the security against impersonation in our model is addressed in the unforgeability against malicious nominator.
- Non-transferability guarantees that the verifier cannot convince a third party that he has received a valid/invalid signature from the nominee on a message. Non-transferability follows from the combination of invisibility and the zero knowledge property of the protocol.

Our Technique. At a high level, our nominative signature NS=(setup, keygen, neSign, nrSign, verify, cfORds=(neTM, vrTM)) proceeds as follows.
At the time of key generation, a user's public and secret key $(\mathsf{pk}_a, \mathsf{sk}_a)$ are first generated using the probabilistic polynomial time (PPT) algorithm $\mathsf{TrapGen}(n, m, q) \rightarrow (\mathbf{A}_a, \mathsf{T}_{\mathbf{A}_a})$. The pair $(\mathsf{pk}_a, \mathsf{sk}_a) = (\mathbf{A}_a, \mathsf{T}_{\mathbf{A}_a})$ is generated by the user a. The user a can be a nominator nr or a nominee ne. We informally describe below how we produce a nominative signature nsig on a message $M \in \{0,1\}^*$ such that nsig satisfies the three properties P1, P2, P3 of a nominative signature mentioned earlier. To issue a signature $\mathsf{Sig} = (M, \mathbf{v}, \mathbf{y}_1, \Pi)$ on a message M to the nominator (nr), the nominee (ne) first finds a short vector $\mathbf{v} \in \mathbb{Z}_q^m$ satisfying $g_{\mathbf{B}_{\mathsf{ne}}}(\mathbf{v}) = \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}) \bmod q$ where $\mathbf{B}_{\mathsf{ne}} = \mathbf{A}_{\mathsf{ne}}$ and $H : \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2 \rightarrow \mathbb{Z}_q^n$ is a hash function. The nominee ne then samples a uniform vector $\mathbf{r}_1 \in \mathbb{Z}_q^m$ and computes $\mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v})$ where

Table 1: Comparative summary of secured nominative signature schemes using bilinear map.

Scheme	size of $\mathsf{pk}_{\mathsf{nr}}$	size of $\mathsf{sk}_{\mathsf{nr}}$	size of $\mathsf{pk}_{\mathsf{ne}}$	size of $\mathsf{sk}_{\mathsf{ne}}$	NS size	pairings	exponentiations
[12]	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{G}$	$4\mathbb{G}$	2	5
[21]	$(k+3)\mathbb{G}$	$\mathbb{G}$	$(k+6)\mathbb{G}$	$(k+3)\mathbb{G}$	$4\mathbb{G}$	3	11

The notations $\mathsf{pk}_{\mathsf{nr}}$, $\mathsf{sk}_{\mathsf{nr}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{sk}_{\mathsf{ne}}$ are respectively the public and secret key size of the nominator and nominee.

Table 2: Comparative summary of lattice based nominative signature schemes.

Scheme	Key size	NS size	Communication cost	Security
[14]	$\widetilde{\mathcal{O}}(n^2)$	$\widetilde{\mathcal{O}}(n)$	$t \cdot (c + \mathcal{O}(L) + 1)$	ROM
Ours	$\widetilde{\mathcal{O}}(n^2)$	$\widetilde{\mathcal{O}}(n)$	$t \cdot [c + 3\lambda + \log_2 3 + \lceil \log q \rceil (m + 2b)]$	$\mathsf{Q(ROM)}$

Here $L = 6(m+1)p$, $p = \lfloor \log_2 \beta \rfloor + 1$, $\beta = 2\sigma\sqrt{m}$ where σ is the standard deviation of the discrete Gaussian distribution.

$f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \to \mathbb{Z}_q^n$ is a collision resistant pseudorandom function (CRPRF). The nominee (ne) finally generates a zero knowledge proof Π using $\mathsf{ZKB++}$ to prove the knowledge of the witness $\mathbf{r}_1 \in \mathbb{Z}_q^m$ and sends the signature $\mathsf{Sig} = (M, \mathbf{v}, \mathbf{y}_1, \Pi)$ to the nominator nr. Note that we sample $\mathbf{r}_1 \in \mathbb{Z}_q^m$ uniformly random, hide $\mathbf{r}_1$ inside a CRPRF and send the knowledge of $\mathbf{r}_1$ through a zero knowledge proof Π. By the soundness property of $\mathsf{ZKB++}$ and the collision resistant property of the PRF f, the probability of finding $\mathbf{r}_1$ by an adversary becomes negligible. After receiving and verifying the signature $\mathsf{Sig} = (M, \mathbf{v}, \mathbf{y}_1, \Pi)$, the nominator ($\mathsf{nr}$) generates the nominative signature $\mathsf{nsig} = (\mathbf{z}, \mathbf{v}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$. The vector $\mathbf{z} \in \mathbb{Z}_q^m$ is a short vector that satisfies the equation $g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{z}) = \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = \mathbf{y}_1 \bmod q$.

Security. The verification of nsig can be performed only by the entity who possess $\mathbf{r}_1 \in \mathbb{Z}_q^m$. We have shown that an adversary which makes successful verification without the knowledge of $\mathbf{r}_1$ with non-negligible probability, can be reduced to an adversary attacking the output indistinguishable property of the pseudorandom function i.e., given $\mathbf{x} \in \mathbb{Z}_q^m$, $\mathbf{y} \in \mathbb{Z}_q^n$ and a pseudorandom function $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^n$, it is hard to distinguish whether $\mathbf{y} \in \mathbb{Z}_q^n$ is uniformly chosen from the set $\mathbb{Z}_q^n$ or $\mathbf{y} = f(\mathbf{k}, \mathbf{x})$ for some $\mathbf{k} \in \mathbb{Z}_q^m$. It has also been spotted that a nominee solves the preimage sampleable function without trapdoor if it can 'alone' produce a nominative signature without interacting with the nominator. In a similar manner, if a nominator produces a nominee's signature then either nominator finds a solution to preimage sampleable function without the trapdoor or solves the collision resistant property of PRF. We achieve the following results.

Theorem 1. *(informal) Our nominative signature scheme achieves unforgeability against malicious nominee under the hardness of collision resistant preimage sampleable function.*

Theorem 2. *(informal) If collision resistant preimage sampleable function is hard then our proposed construction of nominative signature achieves unforgeability against malicious nominator.*

Theorem 3. *(informal) Our scheme follows security under invisibility if for given $\boldsymbol{x} \in \mathbb{Z}_q^m$, $\boldsymbol{y} \in \mathbb{Z}_q^n$ and a pseudorandom function $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^n$, it is hard to decide whether $\boldsymbol{y} \in \mathbb{Z}_q^n$ is uniformly chosen from the set $\mathbb{Z}_q^n$ or $\boldsymbol{y} = f(\boldsymbol{k}, \boldsymbol{x})$ for some $\boldsymbol{k} \in \mathbb{Z}_q^m$.*

Theorem 4. *(informal) If the zero knowledge proof of knowledge* $\mathsf{ZKB++}$ *follows completeness and soundness properties then our nominative signature scheme is secure under non-repudiation.*

Efficiency. Our construction for nominative signature enjoys improved efficiency compared to [12], [21]. It achieves public and secret key sizes $\widetilde{\mathcal{O}}(n^2)$ for a user. The size of the nominative signature is $\widetilde{\mathcal{O}}(n)$ in our scheme. While [12] has public-secret key sizes $|\mathbb{G}|$ for nominator and nominee, the approach in [21] leads the nominator's public key and secret key sizes $(k+3)|\mathbb{G}|$ and $|\mathbb{G}|$ respectively, the nominee's public key and secret key sizes $(k+6)|\mathbb{G}|$ and $(k+3)|\mathbb{G}|$ respectively. In [12], [21] the size of the nominative signature is $4|\mathbb{G}|$. Here $|\mathbb{G}|$ is the bit size of an element of the group $\mathbb{G}$ and k is the length of the message. Thus the size of the public key and secret key of both nominator and nominee in [21] are linear in length k of the message. The NS in [12] uses 2 pairings and 5 exponentiations and the NS in [21] uses 3 pairings and 11 exponentiation.

In our approach, the nominee needs to send a non-interactive zero knowledge proof Π to the nominator and to a public verifier to prove the knowledge of a secret witness $\mathbf{x} \in \mathbb{Z}_q^m$ satisfying the equation $\mathbf{y} = f(\mathbf{x})$ where $\mathbf{y} \in \mathbb{Z}_q^n$ is publicly known. The size of the proof is a central issue apart from the strong security guarantees and fast operations. We use $\mathsf{ZKB{+}{+}}$ with Unruh transform to generate the proof Π. Concretely, the size of the proof Π is $t \cdot [c + 3\lambda + \log_2 3 + \lceil \log q \rceil (m + 2b)]$ where λ is the security parameter, b is the number of binary multiplication gates, t is the number of parallel repetitions and c is the size of the commitments (in bits).

Related Work. There are several variants of user certification system– undeniable signature, designated confirmer signature, designated verifier proofs, universal designated verifier signature, nominative signature etc. [4, 3, 13, 22, 15]. *Undeniable signature* developed by Chaum et al. [4] is one of the user certification systems that allows a signer to have complete control over its signature. The participation of the signer is required in confirmation of the signature by a recipient. If the signer becomes unavailable (such as might refuse to cooperate), a recipient cannot make use of the signature. *Designated confirmer signature* [3] solves this issue by allowing designated parties to confirm the signature to a recipient without the signer.

Undeniable signature allows only the prover to decide 'when' a signature (proof) is verified and not by 'whom' or 'how many parties'. Thus in order to avoid man-in-the-middle attack in such scenario, the prover must be able to designate 'who' will be convinced by the proof. To address this fact, Jakobsson et al. [13] introduced *designated verifier proofs* whereby only the designated verifier can be convinced about the validity or invalidity of the proof. This is due to the fact that the designated verifier can always create a signature for himself indistinguishable from the original signature.

In a user certification system, Alice can send a copy of her certificate issued by a certifying authority to any verifier Bob. After receiving the certificate, Bob can learn all the statements about Alice and will also get convinced about the truth of the statement by verifying the signature of the certifying authority on the certificate. Further, Bob can pass Alice's certificate to any third party and thus it may cause a threat to Alice's privacy. This user privacy in user certification system is addressed by *universal designated verifier signature* (UDVS) [22]. A UDVS scheme can work as a standard digital signature with an additional feature which allows the holder of a signature to designate the signature to any desired designated verifier. Given the designated signature, the designated verifier can verify that the message was signed by the signer, but is unable to convince anyone else of this fact.

All the above mentioned user certification systems require either the direct involvement of the signer or involvement of a designated party appointed by the signer to verify the signature. Nominative signature solves this issue and provides the verification privacy to the recipient.

Nominative signature is first introduced by Kim et al. [15] in 1995. Since then it has been studied extensively [15],[6],[11],[12],[18],[21],[23],[25], [27], [14]. However, none of these except [12], [21] and [14] are secure. Both the constructions [12], [21] use bilinear maps on prime order groups. The underlying hard problems in [12] are bilinear Diffie-Hellman exponent (BDHE) problem, weak computational Diffie-Hellman-I (WCDH-I) problem, WCDH-II problem, weak discrete logarithm (WDLOG) problem and weak decisional Diffie-Hellman (WDDH) problem. The scheme in [21] is secure under the hardness of discrete logarithm problem and the decisional linear problem. This is the only scheme secure in the standard model and uses the transformation proposed by Cramer et al. [5] which converts a sigma protocol into a perfect zero knowledge proof of knowledge. The resulting zero knowledge proofs are efficient 4–move protocols and no additional hardness assumptions are required in the transformation. The construction in [12] is secure in the random oracle model (ROM) and employs the witness indistinguishable Fiege and Fiat-Shamir protocol [8] to prove/disprove the DH-tuple. Nominator starts the communication to issue the nominative signature in both the schemes [12, 21] and the communication between nominee and nominator does not involve the zero knowledge proof in [12, 21]. In contrast, in the work of [14] as well as ours nominee begins the communication between nominee and nominator. The zero knowledge argument system of [16] is used in the process of issuing nominative signature in [14]. To the best of our knowledge, the scheme in [14] is the only scheme secure under standard assumptions on lattice. The scheme in [14] is secure in the random oracle model (ROM).

Organization of the paper. We define notations and recall some concepts from lattice, symmetric key primitives in Section 2. We provide the syntax and security model of nominative signature in Section 3. In Section 4, we present our construction and provide the security proofs for the security models given in Section 3. Section 5 concludes with a conclusion.

2 Preliminaries

Notations. Throughout this paper we assume that a vector $\mathbf{a} \in \mathbb{S}^n$ denotes a column vector of dimension $n \times 1$ with entries from the set $\mathbb{S}$. For $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{R}^n$, let $||\mathbf{u}||_\infty = \max_i |u_i|$ denotes the maximum norm and $||\mathbf{u}|| = \sqrt{u_1^2 + \ldots + u_n^2}$ stands for the Euclidean norm. Let $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m)$ be a matrix with m columns in $\mathbb{R}^n$. Then $||\mathbf{A}|| = \max_{1 \le i \le m} ||\mathbf{a}_i||$. We say that a function f is negligible in λ if $f = \lambda^{-\omega(1)}$. The abbreviation PPT refers to the probabilistic polynomial time.

Definition 1. (Hash function). *[20] A hash function $H : \mathbf{D} \to \mathbf{R}$ is a function that takes an input $x \in \mathbf{D}$ of arbitrary length and returns $H(x) \in \mathbf{R}$ of fixed finite length. Hash functions must be easy to compute. The "ideal" hash function verifies the following properties.*

(i) Collision resistance*: Finding two inputs $x, x' \in \mathbf{D}$ such that $x \neq x'$ and $H(x) = H(x') \in \mathbf{R}$ should cost $\Omega(|\mathbf{R}|^{\frac{1}{2}})$ by the birthday paradox [26].*

(ii) Second preimage resistance*: Given $x \in \mathbf{D}$, $H(x) \in \mathbf{R}$, finding another $x' \in \mathbf{D}$ such that $H(x) = H(x') \in \mathbf{R}$ should cost $\Theta(|\mathbf{R}|)$ by classical exhaustive search.*

(iii) Preimage resistance*: From a hash value $h \in \mathbf{R}$, finding an input $x \in \mathbf{D}$ so that $H(x) = h$ should cost $\Theta(|\mathbf{R}|)$ by classical exhaustive search.*

Definition 2. (Pseudorandom function (PRF)) *[19]. Let $\mathcal{F} = \{F : \mathbf{D} \to \mathbf{R}\}$ denotes the set of all functions mapping from the domain $\mathbf{D}$ to the range $\mathbf{R}$. Let $f : \mathcal{K} \times \mathbf{D} \to \mathbf{R}$ be an efficiently computable, length preserving keyed function where $\mathcal{K}$ is the set of keys, $\mathbf{D}$ is the domain and $\mathbf{R}$ is the range of f. We say that f is a pseudorandom function (PRF) if for all PPT distinguisher D,*

$$|\mathsf{Pr}[D^{f(k,.)}(1^n) = 1] - \mathsf{Pr}[D^{F(.)}(1^n) = 1]| \le \mathsf{negl}(n)$$

where F is chosen uniformly at random from $\mathcal{F}$ and $\mathsf{negl}(n)$ is a negligible function in n.

Definition 3. (Collision resistant pseudorandom function (CRPRF)). *Let $f : \mathcal{K} \times \mathbf{D} \to \mathbf{R}$ be a* PRF. *We say f is a* CRPRF *if given $x_0 \in \mathcal{K}, y_0 \in \mathbf{D}$ and $f(x_0, y_0) \in \mathbf{R}$, it is computationally hard to find $x_1 \in \mathcal{K}, y_1 \in \mathbf{D}$ such that $f(x_0, y_0) = f(x_1, y_1)$.*

Definition 4. (Problem-I) *[19] Given $\boldsymbol{x} \in \mathbb{Z}_q^m$, $\boldsymbol{y} \in \mathbb{Z}_q^n$ and a pseudorandom function $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^n$, it is hard to decide whether $\boldsymbol{y} \in \mathbb{Z}_q^n$ is uniformly chosen from the set $\mathbb{Z}_q^n$ or $\boldsymbol{y} = f(\boldsymbol{k}, \boldsymbol{x})$ for some $\boldsymbol{k} \in \mathbb{Z}_q^m$.*

2.1 Lattice Problems

Definition 5. (Lattice).[10] *For any $m \ge n$, let $\boldsymbol{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m\}$ be any linearly independent set of vectors in $\mathbb{R}^n$. A lattice generated by the set $\boldsymbol{B}$ is defined as $\Lambda(\boldsymbol{B}) = \{\sum_{\boldsymbol{b}_i \in \boldsymbol{B}} c_i \boldsymbol{b}_i : c_i \in \mathbb{Z}\}$ with basis $\boldsymbol{B}$ of dimension n.*

For $q \in \mathbb{N}$, matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and vector $\mathbf{u} \in \mathbb{Z}_q^n$, we define the following three q-ary lattices generated by $\mathbf{A}$: $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = \mathbf{0} \bmod q\}$, $\Lambda_q^{\mathbf{u}}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = \mathbf{u} \bmod q\}$, $\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}^t\mathbf{s} = \mathbf{x} \bmod q,$ for some $\mathbf{s} \in \mathbb{Z}_q^n\}$, where m, n are integers with $m \ge n \ge 1$ and $\mathbf{0}$ is a zero vector of size $n \times 1$.

Definition 6. (Gaussian distribution over a lattice). [10] *For a lattice Λ and a real number $\sigma > 0$, discrete Gaussian distribution over Λ centered at $\boldsymbol{0}$, denoted by $D_{\Lambda,\sigma}$, is defined as: $\forall \boldsymbol{y} \in \Lambda$, $D_{\Lambda,\sigma}[\boldsymbol{y}] \sim exp(-\pi||\boldsymbol{y}||^2/\sigma^2)$, i.e. $D_{\Lambda,\sigma}[\boldsymbol{y}]$ is proportional to $exp(-\pi||\boldsymbol{y}||^2/\sigma^2)$ where $D_{\Lambda,\sigma}[\boldsymbol{y}]$ means the vector $\boldsymbol{y} \hookleftarrow D_{\Lambda,\sigma}$. We say that $D_{\Lambda,\sigma}$ is a distribution with standard deviation σ.*

Lemma 1. [10] *For any n-dimensional lattice Λ and for any real number $\sigma > 0$, we have the following results and probabilistic polynomial time (PPT) algorithms:*

(i) $\mathsf{Pr}_{\boldsymbol{b}\hookleftarrow D_{\Lambda,\sigma}}[||\boldsymbol{b}|| \leq \sigma\sqrt{n}] \geq 1 - 2^{-\Omega(n)}$, i.e. if $\boldsymbol{b} \hookleftarrow D_{\Lambda,\sigma}$ then $||\boldsymbol{b}|| \leq \sigma\sqrt{n}$ with overwhelming probability.

(ii) $\mathsf{TrapGen}(n, m, q) \rightarrow (\boldsymbol{A}, \mathsf{T}_{\boldsymbol{A}})$. This randomized algorithm outputs a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ and a short basis $\mathsf{T}_{\boldsymbol{A}} \in \mathbb{Z}^{m\times m}$ of $\Lambda_q^{\perp}(\boldsymbol{A})$ such that $\boldsymbol{A}$ is within the statistical distance $2^{-\Omega(n)}$ to $U(\mathbb{Z}_q^{n\times m})$ and $||\widetilde{\mathsf{T}}_{\boldsymbol{A}}|| \leq \mathcal{O}(\sqrt{n\log q})$. Here $U(\mathbb{Z}_q^{n\times m})$ is the uniform distribution of integer matrices over $\mathbb{Z}_q$ of order $n \times m$ and $\widetilde{\mathsf{T}}_{\boldsymbol{A}}$ is the Gram-Schmidt orthogonalization of $\mathsf{T}_{\boldsymbol{A}}$.

(iii) $\mathsf{SampleD}(1^n) \rightarrow \boldsymbol{x}$. This algorithm samples a vector $\boldsymbol{x} \in \mathbb{Z}^m$ from the distribution $D_m = \{\boldsymbol{e} \in \mathbb{Z}^m \mid ||\boldsymbol{e}|| \leq \sigma\sqrt{m}\}$. Here $m \geq 5n \log q$ and $\sigma \geq m^{1+\epsilon} \cdot \omega(\sqrt{\log m})$ for any $\epsilon > 0$.

(iv) $\mathsf{SamplePre}(\boldsymbol{A}, \mathsf{T}_{\boldsymbol{A}}, \boldsymbol{u}, \sigma) \rightarrow (\boldsymbol{x} \in \Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}))$. This algorithm takes as input a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$, a short basis $\mathsf{T}_{\boldsymbol{A}} \in \mathbb{Z}^{m\times m}$ of $\Lambda_q^{\perp}(\boldsymbol{A})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^m$ and a real number σ. The algorithm returns a short vector $\boldsymbol{x} \in \Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ sampled from a distribution statistically close to $D_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}),\sigma}$ whenever $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ is non empty i.e., $\boldsymbol{x}$ satisfies the relation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{u} \bmod q$.

Definition 7. ((Inhomogeneous) short integer solution (I(SIS)) search problem) [1]. *Given an integer q, a real number β, a uniformly random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ and a uniformly random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, the* ISIS *problem is to find a non-zero integer vector $\boldsymbol{e} \in \mathbb{Z}^m$ such that $\boldsymbol{A}\boldsymbol{e} = \boldsymbol{u} \bmod q$ and $||\boldsymbol{e}|| \leq \beta$ with non-negligible probability. If $\boldsymbol{u} = \boldsymbol{0} \in \mathbb{Z}_q^n$, then it is known as* short integer solution (SIS) *problem.*

If $m, \beta = poly(n)$ and $q > \beta \cdot \widetilde{\mathcal{O}}(\sqrt{n})$, then the SIS problem with parameters n, m, q, β is at least as hard as the worst-case lattice problem shortest independent vector problem (SIVP_γ) for some $\gamma = \beta\cdot\widetilde{\mathcal{O}}(\sqrt{mn})$ [17].

Definition 8. *(Preimage Sampleable Function* (PSF)) [10]. *A collection of* PSFs *based on the average-case hardness of* SIS *and/or* ISIS *is defined as follows for $q = poly(n)$, $m \geq 5n \log q$, $L = m^{1+\epsilon}$ for any $\epsilon > 0$, and Gaussian parameter $\sigma \geq L \cdot \omega(\sqrt{\log m})$.*

(i) The function generator uses the algorithm $\mathsf{TrapGen}(n, m, q) \rightarrow (\boldsymbol{A}, \mathsf{T}_{\boldsymbol{A}})$ described in Lemma 1 where $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$, $\mathsf{T}_{\boldsymbol{A}} \in \mathbb{Z}^{m\times m}$.

(ii) The function $g_{\boldsymbol{A}} : D_{\mathbb{Z}^m,\sigma} \rightarrow \mathbb{Z}_q^n$ is an efficiently computable function with public basis $\boldsymbol{A}$ and is defined as $g_{\boldsymbol{A}}(\boldsymbol{d}) = \boldsymbol{A} \cdot \boldsymbol{d} \bmod q$ with $\boldsymbol{d} \in D_{\mathbb{Z}^m,\sigma} = \{\boldsymbol{e} \in \mathbb{Z}^m \mid ||\boldsymbol{e}|| \leq \sigma\sqrt{m}\}$ and $g_{\boldsymbol{A}}(\boldsymbol{d}) \in \mathbb{Z}_q^n$. The input $\boldsymbol{d}$ is sampled using the algorithm SampleD *given in Lemma 1 with the standard basis for $\mathbb{Z}^m$. The distribution of $g_{\boldsymbol{A}}(\boldsymbol{u})$ is uniform over the range $\mathbb{Z}_q^n$.*

(iii) The trapdoor inversion algorithm $\mathsf{SamplePre}(\boldsymbol{A}, \mathsf{T}_{\boldsymbol{A}}, \boldsymbol{u}, \sigma)$ samples from the conditional distribution of $\boldsymbol{d} \leftarrow \mathsf{SampleD}(1^n)$ given $g_{\boldsymbol{A}}(\boldsymbol{d}) = \boldsymbol{u}$ for every $\boldsymbol{u} \in \mathbb{Z}_q^n$.

(iv) For any PPT algorithm $\mathcal{F}$, $\mathsf{Pr}[\mathcal{F}(1^n, \boldsymbol{A}, \boldsymbol{u}) = \boldsymbol{d} \in g_{\boldsymbol{A}}^{-1}(\boldsymbol{u}) \subseteq D_{\mathbb{Z}^m,\sigma}] \leq \mathsf{negl}(n)$ where the probability is taken over the choice of $\boldsymbol{A}$ and the target value $\boldsymbol{u} \leftarrow \mathbb{Z}_q^n$ chosen uniformly at random.

A collision-resistant PSF *satisfies the above four properties in addition to the following two properties.*

(v) Preimage min-entropy*: For every $\boldsymbol{u} \in \mathbb{Z}_q^n$, the conditional min-entropy of $\boldsymbol{d} \leftarrow \mathsf{SampleD}(1^n)$ given $g_{\boldsymbol{A}}(\boldsymbol{d}) = \boldsymbol{u}$ is atleast $\omega(\log n)$. It means that the most likely element $\boldsymbol{d} \leftarrow \mathsf{SampleD}(1^n)$ given $g_{\boldsymbol{A}}(\boldsymbol{d}) = \boldsymbol{u}$ for any $\boldsymbol{u} \in \mathbb{Z}_q^n$, occurs with probability $2^{-\omega(\log n)}$.*

(vi) Collision-resistance*: For any PPT algorithm $\mathcal{F}$, $\mathsf{Pr}[\mathcal{F}(1^n, \boldsymbol{A}) \rightarrow (\boldsymbol{d}, \boldsymbol{d}') \in D_{\mathbb{Z}^m,\sigma} \times D_{\mathbb{Z}^m,\sigma} \mid g_{\boldsymbol{A}}(\boldsymbol{d}) = g_{\boldsymbol{A}}(\boldsymbol{d}')] \leq \mathsf{negl}(n)$.*

2.2 ZKB++ [2]

This section recalls the non-interactive zero knowledge proof system ZKB++ on arbitrary circuits that uses Fiat-Shamir transform [9] and $(2,3)$-decomposition of a function $f : \mathbf{F}^s \to \mathbf{F}^l$ which can be expressed as an n-gate arithmetic circuit on an arbitrary finite field $\mathbf{F}$ using addition by constant, multiplication by constant, binary addition and binary multiplication gates. The prover wants to prove the knowledge of a witness x satisfying $y = f(x)$ where y, f are public. A $(2,3)$-decomposition of f is given by the following functions where all the arithmetic operations are over $\mathbf{F}$.
(i) $\mathsf{Share}(x, \rho_1, \rho_2, \rho_3) \to (\mathsf{sh}_1^{(0)}, \mathsf{sh}_2^{(0)}, \mathsf{sh}_3^{(0)}) \in (\mathbf{F}^s)^3$ is a randomized invertible function that generates $\mathsf{sh}_1^{(0)} = H_1(\rho_1)$, $\mathsf{sh}_2^{(0)} = H_1(\rho_2)$, $\mathsf{sh}_3^{(0)} = x - \mathsf{sh}_1^{(0)} - \mathsf{sh}_2^{(0)}$. Here $\mathsf{sh}_j^{(0)}$ is the share for party P_j, $\rho_j \in \mathbf{F}^*$ is the randomness used by the party P_j for $j = 1,2,3$ and $H_1 : \mathbf{F}^* \to \{0,1\}^s$ is a hash function.
(ii) $\mathsf{Update}_j^{(k)}(\mathsf{sh}_j^{(k)}, \mathsf{sh}_{j+1}^{(k)}, \rho_j, \rho_{j+1}) \to \mathsf{sh}_j^{(k+1)} \in \mathbf{F}^s$ computes party P_j's share of the output wire of gate g_k for $k = 0, 1, \ldots, n-1$ using shares, randomness of parties P_j, P_{j+1} and using the following gate specific operations. The notation w_d refers to the d-th wire and $w_d^{(j)}$ refers to the value of w_d in party P_j's share.
addition by constant $(w_b = w_a + c)$: $w_b^{(j)} = \begin{cases} w_a^{(j)} + c & \text{if } j = 1 \\ w_a^{(j)} & \text{otherwise} \end{cases}$
multiplication by constant $(w_b = w_a \cdot c)$: $w_b^{(j)} = c \cdot w_a^{(j)}$
binary addition $(w_c = w_a + w_b)$: $w_c^{(j)} = w_a^{(j)} + w_b^{(j)}$
binary multiplication $(w_c = w_a \cdot w_b)$: $w_c^{(j)} = w_a^{(j)} \cdot w_b^{(j)} + w_a^{(j+1)} \cdot w_b^{(j)} + w_a^{(j)} \cdot w_b^{(j+1)} + Q_j(C) - Q_{j+1}(C)$, where $Q_j(C)$ is the C-th output of a pseudorandom generator seeded with ρ_j.
(iii) $\mathsf{Output}_j(\mathsf{sh}_j^{(n)}) \to \mathsf{out}_j \in \mathbf{F}^l$ selects l output wires of the circuit stored in the share $\mathsf{sh}_j^{(n)}$.
(iv) $\mathsf{Reconstruct}(\mathsf{out}_1, \mathsf{out}_2, \mathsf{out}_3) \to y = \mathsf{out}_1 + \mathsf{out}_2 + \mathsf{out}_3 \in \mathbf{F}^l$.

Theorem 5. *[2] For all* $(\mathsf{sh}_1^{(0)}, \mathsf{sh}_2^{(0)}, \mathsf{sh}_3^{(0)}) \leftarrow \mathsf{Share}(x, \rho_1, \rho_2, \rho_3)$, $\mathsf{sh}_j^{(k+1)} \leftarrow \mathsf{Update}_j(\mathsf{sh}_j^{(k)}, \mathsf{sh}_{j+1}^{(k)}, \rho_j, \rho_{j+1})$ *for* $k = 0, 1, \ldots, n-1$, $\mathsf{out}_j \leftarrow \mathsf{Output}_j(\mathsf{sh}_j^{(n)})$, *and for all circuits* f, *for all inputs* x, $\Pr[f(x) = \mathsf{Reconstruct}(\mathsf{out}_1, \mathsf{out}_2, \mathsf{out}_3)] = 1$.

Formally, ZKB++ for the relation $L = \{\text{public}(f, y), \text{private}(x) | y = f(x)\}$ is a non-interactive one round protocol ZKB=(prove, verify) between a prover and a verifier, both having access to y and f with the following requirements. Here ZKB.prove is a PPT algorithm and ZKB.verify is a deterministic algorithm. Let $H_1 : \mathbf{F}^* \to \mathbf{F}^s$, $H_2 : \mathbf{F}^* \to \mathbf{F}^m$, $H_3 : \mathbf{F}^* \to \{1,2,3\}^t$ be three hash functions used by both the prover and verifier.

• $\mathsf{ZKB.prove}(L) \to (\Pi_L)$. The prover samples randomness $\rho_j[\gamma] \in \mathbf{F}^*$ for each party P_j and does the following for $\gamma = 1, 2, \ldots, t$ in parallel.
− compute $(\mathsf{sh}_1^{(0)}[\gamma], \mathsf{sh}_2^{(0)}[\gamma], \mathsf{sh}_3^{(0)}[\gamma]) \leftarrow \mathsf{Share}(x, \rho_1[\gamma], \rho_2[\gamma], \rho_3[\gamma])$ where $\mathsf{sh}_1^{(0)} = H_1(\rho_1)$, $\mathsf{sh}_2^{(0)} = H_1(\rho_2)$, $\mathsf{sh}_3^{(0)} = x - \mathsf{sh}_1^{(0)} - \mathsf{sh}_2^{(0)}$, $\mathsf{sh}_j^{(k+1)}[\gamma] \leftarrow \mathsf{Update}_j^{(k)}(\mathsf{sh}_j^{(k)}[\gamma], \mathsf{sh}_{j+1}^{(k)}[\gamma], \rho_j[\gamma], \rho_{j+1}[\gamma])$, $k = 0, \ldots, n-1$,
$\mathsf{out}_j[\gamma] \leftarrow \mathsf{Output}_j(\mathsf{sh}_j^{(n)}[\gamma])$, $\mathsf{cmt}_j[\gamma] = H_2(\rho_j[\gamma], \mathsf{sh}_j^{(0)}[\gamma], \mathsf{sh}_j^{(n)}[\gamma])$.
− set $a[\gamma] = (\mathsf{out}_1[\gamma], \mathsf{out}_2[\gamma], \mathsf{out}_3[\gamma], \mathsf{cmt}_1[\gamma], \mathsf{cmt}_2[\gamma], \mathsf{cmt}_3[\gamma])$. The prover then computes the challenge $\mathsf{ch} = H_3(a[1], a[2], \ldots, a[t]) \in \{1,2,3\}^t$. Let $\mathsf{ch}[\gamma] \in \{1,2,3\}$ denotes the γ-th digit of ch. For $\gamma = 1, 2, \ldots, t$, the prover sets $\mathsf{com}[\gamma] = (\mathsf{out}_{(j+2 \bmod 3)}[\gamma], \mathsf{cmt}_{(j+2 \bmod 3)}[\gamma])$ where $j = \mathsf{ch}[\gamma]$ and
$$\mathsf{rsp}[\gamma] = \begin{cases} (\mathsf{sh}_2^{(n)}[\gamma], \rho_1[\gamma], \rho_2[\gamma]) & \text{if } \mathsf{ch}[\gamma] = 1 \\ (\mathsf{sh}_3^{(n)}[\gamma], \rho_2[\gamma], \rho_3[\gamma], \mathsf{sh}_3^{(0)}[\gamma]) & \text{if } \mathsf{ch}[\gamma] = 2 \\ (\mathsf{sh}_1^{(n)}[\gamma], \rho_3[\gamma], \rho_1[\gamma], \mathsf{sh}_3^{(0)}[\gamma]) & \text{if } \mathsf{ch}[\gamma] = 3. \end{cases}$$
Finally, the prover outputs the proof $\Pi_L = [\mathsf{ch}, (\mathsf{com}[1], \mathsf{rsp}[1]), \ldots, (\mathsf{com}[t], \mathsf{rsp}[t])]$.

• $\mathsf{ZKB.verify}(y, f, \Pi_L) \to (0 \vee 1)$. For each iteration $\gamma \in [1, t]$, the verifier proceeds as follows where $j = \mathsf{ch}[\gamma]$:
− Reconstruct

$$\mathsf{sh}^{(0)}_j[\gamma] = \begin{cases} H_1(\rho_1[\gamma]) & \text{if } j = 1 \\ H_1(\rho_2[\gamma]) & \text{if } j = 2 \\ \mathsf{sh}^{(0)}_3[\gamma] & \text{if } j = 3 \end{cases}, \ \mathsf{sh}^{(0)}_{j+1 \bmod 3}[\gamma] = \begin{cases} H_1(\rho_2[\gamma]) & \text{if } j = 1 \\ \mathsf{sh}^{(0)}_3[\gamma] & \text{if } j = 2 \\ H_1(\rho_1[\gamma]) & \text{if } j = 3 \end{cases}$$

where $\rho_1[\gamma], \rho_2[\gamma], \rho_3[\gamma], \mathsf{sh}^{(0)}_3[\gamma]$ are extracted from $\mathsf{rsp}[\gamma]$ in Π_L.

− Recompute $\mathsf{sh}^{(n)}_j[\gamma]$ using $\mathsf{sh}^{(0)}_j[\gamma], \mathsf{sh}^{(0)}_{j+1 \bmod 3}[\gamma]$ and extracting $\rho_1[\gamma]$, $\rho_2[\gamma]$, $\rho_3[\gamma]$ from $\mathsf{rsp}[\gamma]$ in Π_L and invoking algorithm

$\mathsf{Update}^{(k)}_j\big(\mathsf{sh}^{(k)}_j[\gamma], \mathsf{sh}^{(k)}_{(j+1 \bmod 3)}[\gamma]), \rho_j[\gamma], \rho_{(j+1 \bmod 3)}[\gamma]\big) \to \mathsf{sh}^{(k+1)}_j[\gamma]$ for $k = 0, 1, \ldots, n-1$.

− Recover $\mathsf{out}_j[\gamma] \leftarrow \mathsf{Output}_j(\mathsf{sh}^{(n)}_j[\gamma])$, $\mathsf{cmt}_j[\gamma] = H_2(\rho_j[\gamma], \mathsf{sh}^{(0)}_j[\gamma], \mathsf{sh}^{(n)}_j[\gamma])$,

$\mathsf{out}_{(j+1 \bmod 3)}[\gamma] \leftarrow \mathsf{Output}_{(j+1 \bmod 3)}(\mathsf{sh}^{(n)}_{(j+1 \bmod 3)}[\gamma])$,

$\mathsf{cmt}_{(j+1 \bmod 3)}[\gamma] = H_2(\rho_{(j+1 \bmod 3)}[\gamma], \mathsf{sh}^{(0)}_{(j+1 \bmod 3)}[\gamma], \mathsf{sh}^{(n)}_{(j+1 \bmod 3)}[\gamma])$.

Note that $\mathsf{com}[\gamma]$ in Π_L contains $\mathsf{out}_{(j+2 \bmod 3)}$ and $\mathsf{cmt}_{(j+2 \bmod 3)}$.

− Set $a'[\gamma] = (\mathsf{out}_1[\gamma], \mathsf{out}_2[\gamma], \mathsf{out}_3[\gamma], \mathsf{cmt}_1[\gamma], \mathsf{cmt}_2[\gamma], \mathsf{cmt}_3[\gamma])$.

The verifier computes $\mathsf{ch}' = H_3(a'[1], a'[2], \ldots, a'[t]) \in \{1,2,3\}^t$. If $\mathsf{ch}' = \mathsf{ch}$, then the verifier accepts the proof Π_L and outputs 1; otherwise it rejects Π_L and outputs 0.

2.2.1 Unruh Transform [24]

Similar to Fiat-Shamir (FS) transform [9], Unruh transform allows to construct non-interactive zero knowledge (NIZK) proofs and signature schemes from Σ-protocol [7] that provide provable security in the quantum random oracle model.

Given a Σ-protocol with challenge space $\mathcal{S} = \{1,2,3\}$, an integer t, two hash functions $H : \mathbf{F}^* \to \{1,2,3\}^t$ and $G : \mathbf{F}^* \to \mathbf{F}^t$, the prover under Unruh's transform will perform the following steps.

− Run the first phase of the Σ-protocol t times to produce $\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_t$.

− For each $\gamma \in [1,t]$ and $j \in \mathcal{S}$, compute the response $\mathsf{rsp}_{\gamma,j}$ for com_γ and challenge $j \in \mathcal{S}$. Compute $\sigma_{\gamma,j} = G(\mathsf{rsp}_{\gamma,j})$.

− Compute $\{j_1, j_2, \ldots, j_t\} = H(x, \mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_t, \sigma_{1,1}, \ldots, \sigma_{t,1}, \sigma_{1,2}, \ldots, \sigma_{t,2}, \sigma_{1,3}, \ldots, \sigma_{t,3})$.

− Output $\Pi_L = (\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_t, \mathsf{rsp}_{1,j_1}, \mathsf{rsp}_{2,j_2}, \ldots, \mathsf{rsp}_{t,j_t}, \sigma_{1,1}, \ldots, \sigma_{t,1}, \sigma_{1,2}, \ldots, \sigma_{t,2}, \sigma_{1,3}, \ldots, \sigma_{t,3})$.

The verifier will verify the hash, verify that the given values match the corresponding σ_{i,j_i} values, and that the rsp_{i,j_i} values are valid responses with respect to the com_i values.

3 Nominative Signature- Syntax and Security

Syntax of nominative signature. At high level, We define a nominative signature scheme NS as four PPT algorithms setup, keygen, neSign, nrSign, a deterministic algorithm verify and a two party interactive protocol cfORds. A trusted third party, called the key generation center (KGC), generates the system parameter $\mathcal{Y} \leftarrow$ NS.setup. A user generates its public-secret key pairs (pk, sk)←NS.keygen. The public keys are made publicly available while the secret keys are kept secret to the respective entities. The nominee uses its own secret key sk to generate a signature (Sig, state) ←NS.neSign and sends Sig to the nominator over a public channel and keeps its updated current internal state state secret to itself. The nominator in turn, issues a nominative signature nsig ←NS.nrSign using its own secret key sk. The nominative signature nsig is verifiable only by the nominee using its internal state state by executing the algorithm NS.verify. Finally, the confirmation (or disavowal) protocol NS.cfORds=(neTM, vrTM) is invoked which is an interactive protocol between the nominee ne and a verifier vr. The aim of this protocol is to make it possible for ne to convince vr that the nominative signature nsig is a valid (or invalid) signature. For this, the nominee ne sets a relation $\mathcal{R}$ using a private witness that is available only to ne and generates $(\mu, \Pi_{\mathcal{R}})$ ←NS.cfORds.neTM. Here $\mu \in \{0,1\}$ and $\Pi_{\mathcal{R}}$ is a zero knowledge proof for a relation $\mathcal{R}$ set by ne. The verifier runs the algorithm NS.cfORds.vrTM on receiving (μ, $\Pi_{\mathcal{R}}$) and returns $\beta \in \{0,1\}$. The bit $\beta = 0$ indicates that the verifier vr is not convinced that the private witness for the relation $\mathcal{R}$ is held by ne itself and disagrees with the proof $\Pi_{\mathcal{R}}$ whereas $\beta = 1$ assures it agrees with the proof $\Pi_{\mathcal{R}}$.

Formally, a nominative signature NS=(setup, keygen, neSign, nrSign, verify, cfORds=(neTM, vrTM)) works as follows.

- $\mathsf{NS.setup}(1^{\lambda}) \to \mathcal{Y}$. On input a security parameter λ, a trusted authority called the key generation center (KGC) generates the system parameter $\mathcal{Y}$.
- $\mathsf{NS.keygen}(\mathcal{Y}, a) \to (\mathsf{pk}_a, \mathsf{sk}_a)$. On given the system parameter $\mathcal{Y}$, the user a generates the public and secret key pair $(\mathsf{pk}_a, \mathsf{sk}_a)$.
- $\mathsf{NS.neSign}(\mathcal{Y}, \mathsf{sk}_v, \mathsf{pk}_v, \mathsf{pk}_u, M) \to (\mathsf{Sig}_{M,v,u}, \mathsf{state}_v)$. On input of the public parameter $\mathcal{Y}$, public-secret key pair $(\mathsf{pk}_v, \mathsf{sk}_v)$ of a nominee v, public key of pk_u the nominator u and the message M, the nominee v runs this algorithm to generate the nominee's signature $\mathsf{Sig}_{M,v,u}$ on a message M with nominator u.
- $\mathsf{NS.nrSign}(\mathcal{Y}, \mathsf{sk}_u, \mathsf{pk}_u, \mathsf{pk}_v, M, \mathsf{Sig}_{M,v,u}) \to \mathsf{nsig}_{M,v,u}$. This algorithm is run by a nominator u on input the public parameter $\mathcal{Y}$, public-secret key pair $(\mathsf{pk}_u, \mathsf{sk}_u)$ of a nominator u, public key pk_v of the nominee v, message M and the nominee's signature $\mathsf{Sig}_{M,v,u}$ and generates the nominative signature $\mathsf{nsig}_{M,v,u}$.
- $\mathsf{NS.verify}(\mathcal{Y}, \mathsf{state}_v, \mathsf{pk}_v, \mathsf{pk}_u, M, \mathsf{nsig}_{M,v,u}) \in \{\mathsf{valid}, \ \mathsf{invalid}\}$. This algorithm is run by the nominee with its state_v and it verifies whether the nominative signature $\mathsf{nsig}_{M,v,u}$ is valid or invalid.
- NS.cfORds=(neTM, vrTM)$\to ((\mu, \Pi_{\mathcal{R}}), \beta)$. This is an interactive protocol between a nominee v and a verifier vr and the nominee returns a bit $\mu \in \{0, 1\}$ and a zero knowledge proof $\Pi_{\mathcal{R}}$. If the verifier agrees with the proof $\Pi_{\mathcal{R}}$, the verifier returns $\beta = 1$. Otherwise vr returns $\beta = 0$.

Security model. Following are the security attributes of a nominative signature.

- Unforgeability against malicious nominee. The nominee ne alone cannot produce a valid nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$ where the nominee ne and the message M both are chosen by the nominator nr. The security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda)$ for unforgeability against malicious nominee between an attacker $\mathcal{F}$ and a simulator $\mathcal{S}$ is provided in Figure 1.

1. The simulator $\mathcal{S}$ generates system parameter $\mathcal{Y}$ and fixes the uncorrupted nominator nr^*. The simulator $\mathcal{S}$ also randomly fixes the public key $\mathsf{pk}_{\mathsf{nr}^*}$ of the nominator nr^* and sends the tuple $(\mathcal{Y}, \mathsf{nr}^*, \mathsf{pk}_{\mathsf{nr}^*})$ to the forger $\mathcal{F}$.
2. The forger $\mathcal{F}$ makes polynomially many, say α, queries to $\mathcal{S}$ for each of the oracles Create, Corrupt, SignNR, SignNE, cfORds. The simulator $\mathcal{S}$ returns $\perp$ when the forger $\mathcal{F}$ makes Corrupt and SignNR queries on a user $a = \mathsf{nr}^*$.
3. Finally, $\mathcal{F}$ outputs a forgery $(M^*, \ \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}})$ on a corrupted nominee ne and the uncorrupted nominator nr such that $\mathsf{pk}_{\mathsf{ne}} \in \mathsf{Lcorrupt}$.
4. The simulator $\mathcal{S}$ returns 1 if the following conditions hold:
   (a) $\mathsf{NS.verify}(\mathcal{Y}, \ \mathsf{state}_{\mathsf{ne}}, \ \mathsf{pk}_{\mathsf{ne}}, \ \mathsf{pk}_{\mathsf{nr}}, \ M^*, \ \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}) \to \mathsf{valid}$,
   (b) $\mathsf{pk}_{\mathsf{nr}} \notin \mathsf{Lcorrupt}$,
   (c) $(\mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}}, \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}) \notin \mathsf{LsignNR}$ and
   (d) $(\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}, \ \mu, \ \Pi_{\mathcal{R}}) \notin \mathsf{LcfORds}$.
   Otherwise, $\mathcal{S}$ returns 0.
5. The forger $\mathcal{F}$ wins the game if $\mathcal{S}$ returns 1.

Figure 1: selective unforgeability game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda)$ against malicious nominee

Definition 9. (Unforgeability against malicious nominee). *We say that a nominative signature is secure under unforgeability against malicious nominee if*

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

for every PPT adversary $\mathcal{F}$ in the experiment $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda)$ *defined in Figure 1 where* $\mathsf{negl}(\lambda)$ *is a negligible function in λ i.e.,* $\mathsf{negl}(\lambda)=\lambda^{-\omega(1)}$.

- $\mathsf{Unforgeability\ against\ malicious\ nominator}$. The nominator nr alone cannot produce a valid nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$ and cannot convince a verifier about the validity or invalidity of the nominative signature. The security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda)$ for unforgeability against malicious nominator between an attacker $\mathcal{F}$ and a simulator $\mathcal{S}$ is provided in Figure 2.

1. The simulator $\mathcal{S}$ generates system parameter $\mathcal{Y} \leftarrow \mathsf{NS.setup}(\lambda)$ and fixes the uncorrupted nominee ne^*. The simulator $\mathcal{S}$ also randomly fixes the public key $\mathsf{pk}_{\mathsf{ne}^*}$ of the nominator ne^* and sends the tuple $(\mathcal{Y}, \mathsf{ne}^*, \mathsf{pk}_{\mathsf{ne}^*})$ to the forger $\mathcal{F}$.
2. The forger $\mathcal{F}$ makes polynomially many, say α, queries to $\mathcal{S}$ for each of the oracles CreateNR, CreateNE, CorruptNR, CorruptNE, SignNR, SignNE, cfORds. The simulator $\mathcal{S}$ returns $\perp$ when the forger $\mathcal{F}$ makes Corrupt, SignNE and cfORds queries on a user $a = \mathsf{ne}^*$.
3. Finally, the forger $\mathcal{F}$ outputs a forgery (M^*, $\mathsf{Sig}^*_{M^*,\mathsf{ne},\mathsf{nr}}$, $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}$) on a corrupted nominator nr and an uncorrupted nominee ne such that $\mathsf{pk}_{\mathsf{nr}} \in \mathsf{LcorruptNR}$, $(\mathsf{pk}_{\mathsf{ne}}, \mathsf{sk}_{\mathsf{ne}}) \in \mathsf{LcreateNE}$.
4. The simulator $\mathcal{S}$ returns 1 if the following holds:
   (a) $\mathsf{NS.verify}(\mathcal{Y}, \mathsf{state}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M^*, \mathsf{nsig}^*_{M^*,\mathsf{ne}^*,\mathsf{nr}^*}) \rightarrow \mathsf{valid}$,
   (b) $\mathsf{pk}_{\mathsf{ne}} \notin \mathsf{LcorruptNE}$,
   (c) $(\mathsf{Sig}^*_{M^*,\mathsf{ne},\mathsf{nr}}, \mathsf{state}_{\mathsf{ne}}) \notin \mathsf{LsignNE}$ and
   (d) $(\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}, \mu, \Pi_{\mathcal{R}}) \notin \mathsf{LcfORds}$.
   Otherwise, $\mathcal{S}$ returns 0.
5. The forger $\mathcal{F}$ wins the game if $\mathcal{S}$ returns 1.

Figure 2: selective unforgeability game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda)$ against malicious nominator

Definition 10. (Unforgeability against malicious nominator). *We say that a nominative signature is secure under unforgeability against malicious nominee if*

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda) = \mathsf{Pr}[\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

for every PPT adversary $\mathcal{F}$ in the experiment $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda)$ defined in Figure 2 where $\mathsf{negl}(\lambda)$ is a negligible function in λ i.e., $\mathsf{negl}(\lambda)=\lambda^{-\omega(1)}$.

- $\mathsf{Security\ under\ invisibility}$. Only the nominee ne can verify the nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$. Let $\mathcal{F}$ be a distinguisher and $\mathcal{S}$ be the challenger. The invisibility game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, b)$ is described in Figure 3.

1. The simulator $\mathcal{S}$ generates system parameter $\mathcal{Y} \leftarrow \mathsf{NS.setup}(\lambda)$ and sends it to the distinguisher $\mathcal{F}$.
2. Next the distinguisher $\mathcal{F}$ makes polynomially many, say α queries to $\mathcal{S}$ for each of the oracles Create, Corrupt, SignNR, SignNE, cfORds.
3. At any point of the game, $\mathcal{F}$ submits a tuple (M^*, ne, nr) where M^* is a message to be signed with ne as the nominee and nr as the nominator such that $\mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}} \in \mathsf{Lcreate}$ but $\mathsf{pk}_{\mathsf{ne}} \notin \mathsf{Lcorrupt}$ i.e., the nominee ne is not corrupted.
4. The simulator chooses a random bit $b \in \{0, 1\}$. If $b = 1$, the simulator $\mathcal{S}$ generates $\mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}} \leftarrow \mathsf{NS.neSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M^*)$, $\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}} \leftarrow \mathsf{NS.nrSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{ne}}, M^*, \mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}})$ and sets $\mathsf{K}_b = \mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$. Else, K_b is generated uniformly.
5. The distinguisher $\mathcal{F}$ observes K_b, outputs a guess b' and wins the game if
   (i) $b' = b$
   (ii) $\mathcal{F}$ does not corrupt $\mathsf{sk}_{\mathsf{ne}}$ i.e., $\mathsf{pk}_{\mathsf{ne}} \notin \mathsf{Lcorrupt}$

Figure 3: Security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, b)$ under invisibility

Definition 11. (Security under invisibility). *A nominative signature scheme is secure under invisibility if*

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{invis}}(\lambda) = |\mathsf{Pr}[\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, 0)] - \mathsf{Pr}[\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, 1)]| \leq \mathsf{negl}(\lambda)$$

for every PPT adversary in the experiment $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, b)$ defined in Figure 3 where $b \in \{0, 1\}$ and $\mathsf{negl}(\lambda)$ is a negligible function in λ.

- $\mathsf{Security\ under\ non\text{-}repudiation}$. If the nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$ is valid then the nominee ne cannot mislead a verifier vr by proving the invalidity of $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$ to the verifier vr and vice versa. Let

$\mathcal{F}$ be a cheating nominee and $\mathcal{S}$ be the simulator. The non-repudiation game $\mathsf{Exp}^{\mathsf{rep}}_{\mathcal{F}}(\lambda)$ is explained in Figure 4.

1. The simulator $\mathcal{S}$ generates $\mathcal{Y} \leftarrow \mathsf{NS.setup}(\lambda)$ and sends it to the adversary $\mathcal{F}$.
2. The adversary $\mathcal{F}$ may make polynomially many, say α, queries to oracles Create, Corrupt, SignNR, SignNE, cfORds.
3. The adversary $\mathcal{F}$ prepares a tuple $(M^*, \mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}, \gamma = 1-\mu)$ where ne is any nominee with $\mathsf{pk}_{\mathsf{ne}} \in$ Lcorrupt, nr is a nominator such that $(\mathsf{pk}_{\mathsf{nr}}, \mathsf{sk}_{\mathsf{nr}}) \in$ Lcreate, $\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$ is a signature on M^* and $(\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}, \mu, \Pi_{\mathcal{R}}) \in$ LcfORds where $\Pi_{\mathcal{R}}$ is a zero knowledge proof for the relation $\mathcal{R}$ and μ is a bit. If NS.verify($\mathcal{Y}$, $\mathsf{state}_{M^*,\mathsf{ne},\mathsf{nr}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M^*,$\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$)$\rightarrow$ valid then $\mu = 1$. Else $\mu = 0$.
4. To mislead, the adversary $\mathcal{F}$ sends the disavowal proof $\Pi_{\mathcal{R}}$ if $\mu = 1$. Otherwise, $\mathcal{F}$ computes the confirmation proof $\Pi_{\mathcal{R}}$. The simulator $\mathcal{S}$ runs NS.cfORds.vrTM($\mathcal{Y}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M^*, $\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$, μ, $\Pi_{\mathcal{R}}$)$\rightarrow \beta$ and returns β.

   The adversary $\mathcal{F}$ wins the game if $\beta = 1$.

Figure 4: Security game $\mathsf{Exp}^{\mathsf{rep}}_{\mathcal{F}}(\lambda)$ under non-repudiation

Definition 12. (Security under non-repudiation). *A nominative signature scheme is secure under non-repudiation if*

$$\mathsf{Adv}^{\mathsf{rep}}_{\mathcal{F}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{rep}}_{\mathcal{F}}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

for every PPT adversary in the experiment $\mathsf{Exp}^{\mathsf{rep}}_{\mathcal{F}}(\lambda)$ *defined in Figure 4 and* $\mathsf{negl}(\lambda)$ *is a negligible function of* λ.

The attacker $\mathcal{F}$ has access to the following oracles in the attack games given in Figures 1-4 and interacts with the simulator $\mathcal{S}$ who generates the public parameter $\mathcal{Y}$ and maintains five private lists: Lcreate, Lcorrupt, LsignNE, LsignNR, LcfORds (each initially empty).

- Create Query: When $\mathcal{F}$ makes this oracle query to $\mathcal{S}$ on a user a, the simulator $\mathcal{S}$ returns pk_a to $\mathcal{F}$ by running NS.keygen($\mathcal{Y}$, a)$\rightarrow$(pk_a, sk_a). The simulator $\mathcal{S}$ stores (pk_a, sk_a) in the list Lcreate.

- Corrupt Query: On receiving this query on a user a from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $(\mathsf{pk}_a, \mathsf{sk}_a) \in$ Lcreate. If not, it returns $\perp$. Otherwise, $\mathcal{S}$ sends sk_a to $\mathcal{F}$ and stores pk_a in the list Lcorrupt.

- SignNE Query: On querying this oracle on a tuple (v, u, M) by $\mathcal{F}$ where v is a nominee, u is a nominator and M is a message, the simulator $\mathcal{S}$ checks whether (pk_u, sk_u), (pk_v, sk_v) $\in$ Lcreate. If not, $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ outputs the signature ($\mathsf{Sig}_{M,v,u}$,state_v) $\leftarrow$ NS.neSign($\mathcal{Y}$, sk_v, pk_v, pk_u, M) of the nominee v on M and stores ($\mathsf{Sig}_{M,v,u}$, state_v) in the list LsignNE where state_v is the current updated internal secret state of the nominee v.

- SignNR Query: In response to this query on $\mathsf{Sig}_{M,v,u}$ from $\mathcal{F}$, the simulator $\mathcal{S}$ verifies whether ($\mathsf{Sig}_{M,v,u}$, state_v) $\in$ LSignNE. If so, $\mathcal{S}$ returns the nominative signature $\mathsf{nsig}_{M,v,u} \leftarrow$ NS.nrSign($\mathcal{Y}$, sk_u, pk_u, pk_v, M, $\mathsf{Sig}_{M,v,u}$) to $\mathcal{F}$ and stores ($\mathsf{Sig}_{M,v,u}$, $\mathsf{nsig}_{M,v,u}$) in the list LsignNR. Otherwise, $\mathcal{S}$ returns $\perp$.

- cfORds Query: On receiving this query on $\mathsf{nsig}_{M,v,u}$ from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether ($\mathsf{Sig}_{M,v,u}$, $\mathsf{nsig}_{M,v,u}$) $\in$ LsignNR. If not, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ extracts state_v from ($\mathsf{Sig}_{M,v,u}$, state_v) $\in$ LSignNE and returns $(\mu, \Pi_{\mathcal{R}}) \leftarrow$ NS.cfORds.neTM($\mathcal{Y}$, state_v, pk_v, pk_u, M, $\mathsf{nsig}_{M,v,u}$) to $\mathcal{F}$ where $\mathcal{R}$ is the relation for the zero knowledge $\Pi_{\mathcal{R}}$. The simulator $\mathcal{S}$ stores ($\mathsf{nsig}_{M,v,u}$, μ, $\Pi_{\mathcal{R}}$) in the list LcfORds.

4 Our Nominative Signature Scheme

Our nominative signature NS=(setup, keygen, neSign, nrSign, verify, cfORds=(neTM, vrTM)) works as follows.

• NS.setup(λ)$\rightarrow \mathcal{Y}$. On input a security parameter $\lambda > 0$, the KGC selects integers n and q of sizes $\mathcal{O}(\lambda)$ and $\mathcal{O}(n^3)$ respectively, sets $m \geq 5n\lceil \log q \rceil$, picks real numbers σ and δ where σ is the standard deviation of the discrete Gaussian distribution $D_{\Lambda,\sigma}$ of size $\Omega(\sqrt{n \log q} \log n)$ and $\delta = 2\sigma\sqrt{m}$ is an error bound. It chooses cryptographically secure hash function $H : \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2 \rightarrow \mathbb{Z}_q^n$ where $\mathcal{M}$ is the message space, selects a CRPRF $f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \rightarrow \mathbb{Z}_q^n$. For our scheme, it is suffice to consider a function that is collision resistant only on one space. That is, given $\mathbf{x}_0, \mathbf{y} \in \mathbb{Z}_q^m$ and $f(\mathbf{x}_0, \mathbf{y}) \in \mathbb{Z}_q^n$, it should be hard to find another $\mathbf{x}_1 \in \mathbb{Z}_q^m$ such that $f(\mathbf{x}_0, \mathbf{y}) = f(\mathbf{x}_1, \mathbf{y})$ and $\mathbf{x}_0 \neq \mathbf{x}_1$. Finally, it publishes the public parameter $\mathcal{Y} = \{n, q, m, \sigma, \delta, H, f\}$.

• $\mathsf{NS.keygen}(\mathcal{Y}, a)\rightarrow (\mathsf{pk}_a, \mathsf{sk}_a)$. The user a invokes the algorithm $\mathsf{TrapGen}(n, m, q)\rightarrow (\mathbf{A}_a, \mathsf{T}_{\mathbf{A}_a})$ (see Lemma 1 in Section 2.1) and sets $\mathsf{pk}_a{=}\mathbf{A}_a \in \mathbb{Z}_q^{n\times m}$, $\mathsf{sk}_a = \mathsf{T}_{\mathbf{A}_a} \in \mathbb{Z}^{m\times m}$. The user a publishes pk_a and keeps sk_a secret to itself.

• $\mathsf{NS.neSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M)\rightarrow (\mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}}{=}(M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f}), \mathsf{state}_{\mathsf{ne}})$. Let $M \in \mathcal{M}$ be a message to be signed. The nominee ne performs the following steps using $\mathcal{Y}{=}(n, q, m, \sigma, \delta, H, f)$, $\mathsf{sk}_{\mathsf{ne}}{=}\mathsf{T}_{\mathbf{A}_{\mathsf{ne}}}$, $\mathsf{pk}_{\mathsf{ne}}{=}\mathbf{A}_{\mathsf{ne}}$ and public key $\mathsf{pk}_{\mathsf{nr}}{=}\mathbf{A}_{\mathsf{nr}}$ of a nominator nr. For our convenience, we write $\mathbf{B}_{\mathsf{ne}} = \mathbf{A}_{\mathsf{ne}}$ and $\mathsf{T}_{\mathbf{B}_{\mathsf{ne}}} = \mathsf{T}_{\mathbf{A}_{\mathsf{ne}}}$.

- Compute $\mathbf{y} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}) \in \mathbb{Z}_q^n$ and generate a short vector $\mathbf{v} \in \mathbb{Z}_q^m$ satisfying $g_{\mathbf{B}_{\mathsf{ne}}}(\mathbf{v}) = \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = \mathbf{y} \bmod q$ with $||\mathbf{v}|| \leq \sigma\sqrt{m}$ by running the algorithm $\mathsf{SamplePre}(\mathbf{B}_{\mathsf{ne}}, \mathsf{T}_{\mathbf{B}_{\mathsf{ne}}}, \mathbf{y}, \sigma)\rightarrow \mathbf{v}$ using the short basis $\mathsf{T}_{\mathbf{B}_{\mathsf{ne}}}$ given in Lemma 1 in Section 2. Note that $||\mathbf{v}||_\infty \leq ||\mathbf{v}|| \leq \sigma\sqrt{m} \leq \delta$ as $\delta = 2\sigma\sqrt{m}$.
- Select randomly $\mathbf{r}_1 \in \mathbb{Z}_q^m$, compute $\mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$ and generate a zero knowledge proof $\Pi_{\mathcal{R}_f}$ for the relation $\mathcal{R}_f = \{\text{public}(f, \mathbf{v}, \mathbf{y}_1, \mathcal{Y}),\ \text{private}\ (\mathbf{r}_1) \mid \mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v})\}$ by executing the algorithm $\mathsf{ZKB.prove}$ using the protocol $\mathsf{ZKB}{+}{+}$ as described in Section 2.2.
- Set the signature $\mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})$, send it to the nominator nr over a two party public channel and update its current internal state $\mathsf{state}_{\mathsf{ne}} = \mathsf{state}_{\mathsf{ne}} \cup (\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr})$ ($\mathsf{state}_{\mathsf{ne}}$ is initially empty).

• $\mathsf{NS.nrSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{ne}}, M, \mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}})\rightarrow (\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v}))$. On receiving the signature $\mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})$ from the nominee ne, the nominator nr executes the following steps using $\mathsf{sk}_{\mathsf{nr}}{=}\mathsf{T}_{\mathbf{A}_{\mathsf{nr}}}$, $\mathsf{pk}_{\mathsf{nr}}{=}\mathbf{A}_{\mathsf{nr}}$ and $\mathsf{pk}_{\mathsf{ne}} = \mathbf{B}_{\mathsf{ne}}$.

- Verify $g_{\mathbf{B}_{\mathsf{ne}}}(\mathbf{v}) = \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}) \bmod q$ with $||\mathbf{v}|| \leq \sigma\sqrt{m}$ and zero knowledge proof $\Pi_{\mathcal{R}_f}$ by invoking the algorithm $\mathsf{ZKB.verify}$ using the $\mathsf{ZKB}{+}{+}$ protocol as described in Section 2.2. The nominator aborts and outputs $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = \perp$ if any of the above verification fails where $\perp$ is a designated symbol indicating verification failure.
- Otherwise, find a short vector $\mathbf{z} \in \mathbb{Z}_q^m$ satisfying $g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{z}) = \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = \mathbf{y}_1 \bmod q$ with $||\mathbf{z}|| \leq \sigma\sqrt{m}$ using the short basis $\mathsf{sk}_{\mathsf{nr}}{=}\mathsf{T}_{\mathbf{A}_{\mathsf{nr}}}$ following the algorithm $\mathsf{SamplePre}(\mathbf{A}_{\mathsf{nr}}, \mathsf{T}_{\mathbf{A}_{\mathsf{nr}}}, \mathbf{y}_1, \sigma)\rightarrow \mathbf{z}$ as in Lemma 1 in Section 2.1 and issues the nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v})$.

• $\mathsf{NS.verify}(\mathcal{Y}, \mathsf{state}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M, \mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}) \in \{\mathsf{valid}, \mathsf{invalid}\}$. This algorithm is executed by the nominee ne with its current internal state $\mathsf{state}_{\mathsf{ne}}$ containing $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr})$ who on receiving $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v})$ uses $\mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}$ and $\mathsf{pk}_{\mathsf{ne}} = \mathbf{B}_{\mathsf{ne}}$ to verify whether $g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{z}) = f(\mathbf{r}_1, \mathbf{v}) \bmod q$ with $||\mathbf{z}|| \leq \sigma\sqrt{m}$ and $g_{\mathbf{B}_{\mathsf{ne}}}(\mathbf{v}) = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}) \bmod q$ with $||\mathbf{v}|| \leq \sigma\sqrt{m}$. Here the nominee ne extracts $\mathbf{v}$, $\mathbf{r}_1$ from its internal secret state $\mathsf{state}_{\mathsf{ne}}$ containing $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr})$. If the verification succeeds, it outputs valid; otherwise it returns $\mathsf{invalid}$.

• $\mathsf{NS.cfORds}{=}(\mathsf{neTM}, \mathsf{vrTM})\rightarrow ((\mu, \Pi_{\mathcal{R}}), \beta)$. This protocol is an interactive protocol between the nominee ne and a verifier vr satisfying the following requirements:

(i) $\mathsf{neTM}(\mathcal{Y}, \mathsf{state}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M, \mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}})\rightarrow (\mu, \Pi_{\mathcal{R}})$. The nominee performs the following steps using $\mathsf{pk}_{\mathsf{ne}} = \mathbf{B}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}$, $\mathsf{state}_{\mathsf{ne}}$ and $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v})$:

- Extract $\mathbf{r}_1, \mathbf{v}$ from $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr})$ contained in $\mathsf{state}_{\mathsf{ne}}$, generate a zero knowledge proof $\Pi_{\mathcal{R}}$ for the relation $\mathcal{R} = \{\text{public}(\mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}, \mathbf{v}, \mathbf{z}, M, \mathcal{Y}),\ \text{private}\ (\mathbf{r}_1) \mid \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}),\ \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\}$ by executing the algorithm $\mathsf{ZKB.prove}$ using the protocol $\mathsf{ZKB}{+}{+}$ as described in Section 2.2.
- Run $\mathsf{NS.verify}(\mathcal{Y}, \mathsf{state}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M, \mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}})$. If the output is valid then return $(\mu = 1, \Pi_{\mathcal{R}})$ to the verifier vr; otherwise, send $(\mu = 0, \Pi_{\mathcal{R}})$ to the verifier vr. The proof $\Pi_{\mathcal{R}}$ is treated as a confirmation (disavowal) proof of the relation $\mathcal{R}$ when $\mu = 1$ ($\mu = 0$).

(ii) $\mathsf{vrTM}(\mathcal{Y}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M, \mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}, \mu, \Pi_{\mathcal{R}}) \rightarrow \beta$. On receiving a pair $(\mu, \Pi_{\mathcal{R}})$ from the nominee ne, the verifier vr verifies the zero knowledge proof $\Pi_{\mathcal{R}}$ by invoking the algorithm $\mathsf{ZKB.verify}$ using the $\mathsf{ZKB}{+}{+}$ protocol as described in Section 2.2. The verifier vr outputs $\beta = 1$ if vr agrees with the confirmation proof $\Pi_{\mathcal{R}}$

i.e., if vr receives $\mu = 1$ and $\mathsf{ZKB.verify}$ passes (or disavowal proof $\Pi_{\mathcal{R}}$ i.e. vr receives $\mu = 0$ and $\mathsf{ZKB.verify}$ succeeds). The verifier vr thus convinced in zero knowledge that neither nominator nor nominee is a cheater. Otherwise, vr disagrees with the confirmation/disavowal proof $\Pi_{\mathcal{R}}$ and returns $\beta = 0$.

Remark 1. *Note that the cheating nominee cannot construct a valid signature on some message $M^* \neq M$. It follows from the zero knowledge proof $\Pi_{\mathcal{R}_f}$ which proves the knowledge of $\boldsymbol{r}_1 \in \mathbb{Z}_q^m$ such that $\boldsymbol{y}_1 = f(\boldsymbol{r}_1, \boldsymbol{v})$ and the collision resistant property of the* PRF f.

Correctness. Let the KGC generates the public parameter $(\mathcal{Y} = (n, q, m, \sigma, \beta, H, f)) \leftarrow \mathsf{NS.setup}(\lambda)$. A nominator nr generates its public-secret key pair $(\mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}, \mathsf{sk}_{\mathsf{nr}} = \mathsf{T}_{\mathbf{A}_{\mathsf{nr}}}) \leftarrow \mathsf{NS.keygen}(\mathcal{Y}, \mathsf{nr})$ and a nominee ne generates its public-secret key pair $(\mathsf{pk}_{\mathsf{ne}} = \mathbf{A}_{\mathsf{ne}}, \mathsf{sk}_{\mathsf{ne}} = \mathsf{T}_{\mathbf{A}_{\mathsf{ne}}}) \leftarrow \mathsf{NS.keygen}(\mathcal{Y}, \mathsf{ne})$. Set $\mathbf{B}_{\mathsf{ne}} = \mathbf{A}_{\mathsf{ne}}$ and $\mathsf{T}_{\mathbf{B}_{\mathsf{ne}}} = \mathsf{T}_{\mathbf{A}_{\mathsf{ne}}}$. Let the nominee ne generates

$$(\mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f}), \mathsf{state}_{\mathsf{ne}})) \leftarrow \mathsf{NS.neSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}}, M)$$

where $\Pi_{\mathcal{R}_f} \leftarrow \mathsf{ZKB.prove}$ is a zero knowledge for the relation $\mathcal{R}_f = \{\text{public}(f, \mathbf{v}, \mathbf{y}_1, \mathcal{Y}),\ \text{private } (\mathbf{r}_1) \mid \mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v})\}$, $\mathbf{v} \in \mathbb{Z}_q^m$ is a short vector satisfying $\mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}})$ with $||\mathbf{v}|| \leq \sigma\sqrt{m}$, $\mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$ with $\mathbf{r}_1 \in \mathbb{Z}_q^m$, $M \in \mathcal{M}$ and $\mathsf{state}_{\mathsf{ne}}$ contains $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr})$. Also, let the nominator nr computes the nominative signature

$$(\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v})) \leftarrow \mathsf{NS.nrSign}(\mathcal{Y}, \mathsf{sk}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}, \mathsf{pk}_{\mathsf{ne}}, M, \mathsf{Sig}_{M,\mathsf{ne},\mathsf{nr}} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f}))$$

where $\mathbf{z}$ is a short vector satisfying the equation $\mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = \mathbf{y}_1 \bmod q$ with $||\mathbf{z}|| \leq \sigma\sqrt{m}$. Let the nominee ne outputs $(\mu, \Pi_{\mathcal{R}}) \leftarrow \mathsf{NS.cfORds.neTM}(\mathcal{Y}, \mathsf{state}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{ne}} = \mathbf{B}_{\mathsf{ne}}, \mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}, M, \mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v}))$ where $\mu \in \{0, 1\}$ and $\Pi_{\mathcal{R}} \leftarrow \mathsf{ZKB.prove}$ is a zero knowledge proof for the relation $\mathcal{R} = \{$ public $(\mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}, \mathbf{v}, \mathbf{z}, M, \mathcal{Y})$, private $(\mathbf{r}_1) \mid \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}), \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\}$ where $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr}) \in \mathsf{state}_{\mathsf{ne}}$.

If the nominee ne, the nominator nr and the verifier vr are honest then we have the following.
(i) The algorithm $\mathsf{NS.verify}(\mathcal{Y}$, $\mathsf{state}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}} = \mathbf{A}_{\mathsf{nr}}$, M, $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{v}))$ run by the honest nominee ne outputs valid as $\mathbf{A}_{\mathsf{nr}} \cdot\ \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})$ where $(\mathbf{r}_1, \mathbf{v}, M, \mathsf{nr}) \in \mathsf{state}_{\mathsf{ne}}$.
(ii) The algorithm $\mathsf{NS.cfORds.vrTM}(\mathcal{Y}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M, $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}}$, μ, $\Pi_{\mathcal{R}})$ run by the honest verifier vr. The verifier vr outputs $\beta = 1$ which follows from the correctness of $\mathsf{ZKB.verify}$ in Section 2.2. In other words, the verifier agrees with the proof $\Pi_{\mathcal{R}}$ and thereby convinced in zero knowledge that ne has the knowledge of the private witness $\mathbf{r}_1$ satisfying $\mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v}) \bmod q$ and $\mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}})$. Note that $\Pi_{\mathcal{R}}$ is a confirmation proof when $\mu = 1$ and disavowal proof when $\mu = 0$.

4.1 Security Proofs

Theorem 6. *The nominative signature scheme* NS=(setup, keygen, neSign, nrSign, verify, cfORds=(neTM, vrTM)) *described in Section 4 is secure under the unforgeability against malicious nominee in the standard model as per Definition 9 for the security game given in Figure 1 assuming the existence of collision-resistant preimage sampleable function (*CRPSF*). Concretely, suppose there is a forger $\mathcal{F}$ that has advantage ϵ against the scheme* NS. *Suppose $\mathcal{F}$ makes atmost $Q_H > 0$ hash queries to H and $Q_f > 0$ pseudorandom queries to f. Then there is an algorithm $\mathcal{S}$ that breaks the collision resistant property of* CRPSF *with probability*

$\mathsf{Pr}\big[\mathcal{F}(\bar{\boldsymbol{A}}, \boldsymbol{x}, n, m, q) \rightarrow \boldsymbol{x}' \ |((g_{\bar{\boldsymbol{A}}}(\boldsymbol{x}) = g_{\bar{\boldsymbol{A}}}(\boldsymbol{x}') \wedge (\boldsymbol{x} \neq \boldsymbol{x}')), \bar{\boldsymbol{A}} \in \mathbb{Z}_q^{n\times m}, \boldsymbol{x} \in \mathbb{Z}_q^m, ||\boldsymbol{x}|| \leq \beta, \beta = 2\sigma\sqrt{m}, \sigma = \Omega(\sqrt{n \log q} \log n), m = 2n \log q, q = \mathcal{O}(n^3), n = \mathcal{O}(\lambda)\big]$

atleast ϵ' where $\epsilon \geq \alpha e^2 \cdot \epsilon'$. Here e is the base of the natural logarithm. The running time of $\mathcal{S}$ is $O(time(\mathcal{F}))$.

Proof. Let a PPT forger $\mathcal{F}$ with the secret key of a nominee forges our proposed nominative signature scheme NS with non-negligible advantage ϵ. We construct an algorithm $\mathcal{S}$ that breaks the collision-resistant property of PSF with probability close to ϵ. The algorithm $\mathcal{S}$ plays the role of the simulator in the security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mne}}(\lambda)$ (Figure 1) and interacts with $\mathcal{F}$ as follows:

1. The simulator $\mathcal{S}$ runs $\mathsf{NS.setup}(\lambda)$ to honestly generate the public parameters $\mathcal{Y} = (n, q, m, \sigma, \delta, H, f)$ with integers n and q of sizes $\mathcal{O}(\lambda)$ and $\mathcal{O}(n^3)$ respectively, real number σ of size $\Omega(\sqrt{n \log q} \log n)$ which

is the standard deviation of the discrete Gaussian distribution $D_{\Lambda,\sigma}$, an error bound $\delta = 2\sigma\sqrt{m}$ where $m \geq 5n\lceil\log q\rceil$, a hash function $H : \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2 \to \mathbb{Z}_q^n$ where $\mathcal{M}$ is the message space and a collision resistant pseudorandom function $f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \to \mathbb{Z}_q^n$. The simulator $\mathcal{S}$ sends $\mathcal{Y}$ to the forger $\mathcal{F}$, fixes the uncorrupted nominator nr^* along with its public key $\mathsf{pk}_{\mathsf{nr}^*} \in \mathbb{Z}_q^{n\times m}$ uniformly at random.

2. The forger $\mathcal{F}$ makes polynomially many, say α, queries to $\mathcal{S}$ for each of the following oracles where Hlist, PRFlist, Successlist, LsignNE, LsignNR, LcfORds are the private lists maintained by the simulator $\mathcal{S}$ (which are initially empty).

– Create Query: On receiving this query on a user $a \neq \mathsf{nr}^*$, the simulator $\mathcal{S}$ runs $\mathsf{NS.keygen}(\mathcal{Y}, a) \to (\mathsf{pk}_a, \mathsf{sk}_a)$ where $\mathsf{pk}_a = \mathbf{A}_a \in \mathbb{Z}_q^{n\times m}$ and $\mathsf{sk}_a = \mathsf{T}_{\mathbf{A}_a} \in \mathbb{Z}^{m\times m}$. For the case $a = \mathsf{nr}^*$, the simulator $\mathcal{S}$ sends the already fixed public key $\mathsf{pk}_{\mathsf{nr}^*}$. In this case $\mathsf{sk}_{\mathsf{nr}^*} = \perp$. The simulator $\mathcal{S}$ stores $(\mathsf{pk}_a, \mathsf{sk}_a) \in \mathsf{Lcreate}$ and sends pk_a to the forger $\mathcal{F}$. When user a plays the role of a nominee, we set $\mathbf{B}_a = \mathbf{A}_a$ and $\mathsf{T}_{\mathbf{B}_a} = \mathsf{T}_{\mathbf{A}_a}$. Observe that the simulation of this query is exactly same as in the original protocol.

– Corrupt Query: On receiving this query on a user a from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $a \neq \mathsf{nr}^*$ and $(\mathsf{pk}_a = \mathbf{A}_a, \mathsf{sk}_a = \mathsf{T}_{\mathbf{A}_a}) \in \mathsf{Lcreate}$. If not, $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ sends sk_a to $\mathcal{F}$ and stores pk_a in the list Lcorrupt.

– Hash Query: To answer hash queries, $\mathcal{S}$ maintains a list Hlist for the hash function H. This list stores records of the form $(\mathbf{x}, H(\mathbf{x}))$ where $\mathbf{x} \in \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2$ and $H(\mathbf{x}) \in \mathbb{Z}_q^n$. The stored value is returned on the queried $\mathbf{x} \in \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2$ if it has already been queried before. Otherwise, a fresh value $H(\mathbf{x}) \in \mathbb{Z}_q^n$ is generated honestly by using the public hash function H and $(\mathbf{x}, H(\mathbf{x}))$ is included in Hlist. The simulator $\mathcal{S}$ returns $H(\mathbf{x})$ to $\mathcal{F}$. Note that the distribution of hash query is same as in the original protocol.

– PRF Query: To respond pseudorandom queries, $\mathcal{S}$ maintains a list PRFlist for the collision resistant pseudorandom function f. Here $f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \to \mathbb{Z}_q^n$ is assumed to be collision resistant only on one space i.e., given $\mathbf{r}_1, \mathbf{v} \in \mathbb{Z}_q^m$, $f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$, it is hard to find $\mathbf{r}_2 \in \mathbb{Z}_q^m$ with $\mathbf{r}_2 \neq \mathbf{r}_1$ and $f(\mathbf{r}_1, \mathbf{v}) = f(\mathbf{r}_2, \mathbf{v})$.

On receiving a query $(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$ from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1) \in \mathsf{PRFlist}$ and returns $\mathbf{y}_1$ to $\mathcal{F}$ if so. Otherwise, $\mathcal{S}$ checks whether $((M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v), H(M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v)) \in \mathsf{Hlist}$ satisfying $\mathbf{B}_v \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v) \bmod q$. If such a pair does not exist in Hlist then $\mathcal{S}$ computes $\mathbf{y}_1 = f(\mathbf{r}_1 \cdot \mathbf{v}) \in \mathbb{Z}_q^n$ honestly similar to that in the real protocol, passes $\mathbf{y}_1 \in \mathbb{Z}_q^n$ to the forger $\mathcal{F}$ and stores $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in PRFlist. If there exists such a pair $((M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v), H(M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v)) \in \mathsf{Hlist}$ satisfying $\mathbf{B}_v \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}^*}, \mathbf{B}_v) \bmod q$ then the simulator $\mathcal{S}$ simulates pseudorandom query on $(\mathbf{r}_1, \mathbf{v})$ as follows.

(i) choose a short vector $\mathbf{d} \leftarrow \mathsf{SampleD}(1^n)$ i.e., $\mathbf{d} \in \mathbb{Z}_q^m$ with $||\mathbf{d}|| \leq \sigma\sqrt{m}$,

(ii) compute $\mathbf{y}_1 = g_{\mathbf{A}_{\mathsf{nr}^*}}(\mathbf{d}) = \mathbf{A}_{\mathsf{nr}^*} \cdot \mathbf{d} \bmod q$ where $g_{\mathbf{A}_{\mathsf{nr}^*}}$ forms a PSF (see Definition 8 in Section 2.2),

(iii) store $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in PRFlist and return $\mathbf{y}_1$ to $\mathcal{F}$,

(iv) maintain a list Successlist and store $(M, \mathbf{r}_1, \mathbf{v}, \mathbf{d}, \mathbf{y}_1)$ in Successlist.

For each distinct query $(\mathbf{r}_1, \mathbf{v})$ to f, the value returned by $\mathcal{S}$ is either the honest value $f(\mathbf{r}_1, \mathbf{v})$ or the value $g_{\mathbf{A}_u}(\mathbf{d})$ where $\mathbf{d} \leftarrow \mathsf{SampleD}(1^n)$; by the uniform output property of the the algorithm SampleD, this is identical to the uniform random value of $f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$. By the pseudorandom property of f, this simulation is same as in the real protocol. We fix the value of $\mathbf{y}_1 \in \mathbb{Z}_q^n$ for the pseudorandom query on $(\mathbf{r}_1, \mathbf{v})$ and for every query on the same tuple $(\mathbf{r}_1, \mathbf{v})$, the simulator returns the same value $\mathbf{y}_1 \in \mathbb{Z}_q^n$. In the original protocol, pseudorandom queries on $(\mathbf{r}_1, \mathbf{v})$ are answered by a single value having the same distribution as that in the simulated answer.

– SignNE Query: On querying this oracle on a tuple $(\mathbf{B}_v, \mathbf{A}_u, M)$ by $\mathcal{F}$ for a nominee v, nominator u and message M, the simulator $\mathcal{S}$ proceed as follows.

(i) checks if $((M, \mathbf{A}_u, \mathbf{B}_v), \mathbf{y}) \notin \mathsf{Hlist}$, then $\mathcal{S}$ queries $(M, \mathbf{A}_u, \mathbf{B}_v)$ to hash query and receives $\mathbf{y} \in \mathbb{Z}_q^n$,

(ii) find $\mathbf{v} \in \mathbb{Z}_q^m$ satisfying $\mathbf{B}_v \cdot \mathbf{v} = \mathbf{y}$,

(iii) check whether $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1) \in \mathsf{PRFlist}$ for some $\mathbf{r}_1 \in \mathbb{Z}_q^m$, $\mathbf{y}_1 \in \mathbb{Z}_q^n$. If no, choose $\mathbf{r}_1 \in \mathbb{Z}_q^m$ randomly and make pseudorandom query on $(\mathbf{r}_1, \mathbf{v})$,

(iv) generate a zero knowledge proof $\Pi_{\mathcal{R}_f}$ using ZKB++ for the relation $\mathcal{R}_f = \big\{$ public $(\mathbf{A}_u, \mathbf{B}_v, \mathbf{v}, \mathbf{z}, M, \mathcal{Y})$, private $(\mathbf{r}_1) \mid \mathbf{A}_u \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\big\}$ and return $\mathsf{Sig}_{M,v,u}$=$(M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})$ to $\mathcal{F}$.

(v) update the current internal state $\mathsf{state}_v = \mathsf{state}_v \cup (\mathbf{r}_1, \mathbf{v}, M, u)$,

(vi) store $(\mathsf{Sig}_{M,v,u}, \mathsf{state}_v)$ in the list LsignNE.

Observe that the distribution of SignNE query is identical to that in the real protocol by the uniform output property of the algorithms SampleD and SamplePre.

– SignNR Query: In response to this query on $\mathsf{Sig}_{M,v,u} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})$ from $\mathcal{F}$, the simulator $\mathcal{S}$ verifies whether the pair $(\mathsf{Sig}_{M,v,u}, \mathsf{state}_v) \in$ LSignNE and $u \neq \mathsf{nr}^*$. If not, $\mathcal{S}$ returns $\perp$. Otherwise, the simulator $\mathcal{S}$ returns the nominative signature $(\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})) \leftarrow$ NS.nrSign$(\mathcal{Y}$, sk_u, pk_u, pk_v, M, $\mathsf{Sig}_{M,v,u})$ to $\mathcal{F}$ and stores $(\mathsf{Sig}_{M,v,u}, \mathsf{nsig}_{M,v,u})$ in the list LsignNR. The distribution of $\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})$ is similar to that in the real protocol.

– cfORds Query: The simulator $\mathcal{S}$, on receiving this query on $\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})$ from $\mathcal{F}$, calls the algorithm NS.cfORds.neTM$(\mathcal{Y}$, state_v, pk_v, pk_u, M, $\mathsf{nsig}_{M,v,u})\rightarrow (\mu, \Pi_{\mathcal{R}})$ to execute the following steps using $\mathsf{pk}_v = \mathbf{B}_v$, $\mathsf{pk}_u = \mathbf{A}_u$ and $(\mathbf{r}_1, \mathbf{v}, M, u)$ contained in state_v.

(i) extract $\mathbf{r}_1, \mathbf{v}$ from $(\mathbf{r}_1, \mathbf{v}, M, u)$ contained in state_v,

(ii) generate a zero knowledge proof $\Pi_{\mathcal{R}}$ for the relation $\mathcal{R} = \big\{$public$(\mathbf{A}_u, \mathbf{B}_v, \mathbf{v}, \mathbf{z}, M, \mathcal{Y})$, private $(\mathbf{r}_1) \mid \mathbf{B}_v \cdot \mathbf{v} = H(M, \mathbf{A}_u, \mathbf{B}_v), \mathbf{A}_u \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\big\}$ with $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in PRFlist and $((M, \mathbf{A}_u, \mathbf{B}_v), H(M, \mathbf{A}_u, \mathbf{B}_v))$ in Hlist using the algorithm ZKB.prove of the non-interactive zero knowledge proof system ZKB++ as described in Section 2.2,

(iii) return $(\mu = 1, \Pi_{\mathcal{R}})$ to the verifier vr if NS.verify$(\mathcal{Y}$, state_v, pk_v, pk_u, M, $\mathsf{nsig}_{M,v,u})$ is valid, otherwise, send $(\mu = 0, \Pi_{\mathcal{R}})$ to vr,

(iv) record $(\mathsf{nsig}_{M,v,u}, \mu, \Pi_{\mathcal{R}})$ in the list LcfORds.

The proof $\Pi_{\mathcal{R}}$ is treated as a confirmation (disavowal) proof of the relation $\mathcal{R}$ when $\mu = 1$ $(\mu = 0)$. Note that, the simulation of $(\mu, \Pi_{\mathcal{R}})$ is exactly same as in the real protocol.

3. Finally, $\mathcal{F}$ produces a forgery $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*), \mathbf{v}^*$ on a message $M^* \in \mathcal{M}$ against an uncorrupted nominator nr and a *corrupted* nominee ne i.e., $\mathsf{pk}_{\mathsf{ne}} \in$ Lcorrupt. If the forgery $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*, \mathbf{v}^*)$ is valid, the following conditions must hold.

(a) NS.verify$(\mathcal{Y}$, $\mathsf{state}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M^*, $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}})\rightarrow$ valid,

(b) $\mathsf{pk}_{\mathsf{nr}} \notin$ Lcorrupt,

(c) $(\mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}}, \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}) \notin$ LsignNR and

(d) $(\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}, \mu, \Pi_{\mathcal{R}})\notin$ LcfORds.

When $\mathsf{nr} = \mathsf{nr}^*$, note that both the equations $\mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z}^* = \mathbf{y}_1^* \bmod q$ with $||\mathbf{z}^*|| \leq \sigma\sqrt{m}$ and $\mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v}^* = \mathbf{y}^* \bmod q$ must be satisfied when $\mathcal{F}$ produces a valid forgery $(M^*, \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*, \mathbf{v}^*))$. This implies that the pair $((M^*, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}), \mathbf{y}^*)) \in$ Hlist, $((\mathbf{r}_1^*, \mathbf{v}^*), \mathbf{y}_1^*) \in$ PRFlist and $(M^*, \mathbf{r}_1^*, \mathbf{v}^*, \mathbf{d}^*, \mathbf{y}_1^*) \in$ Successlist. The simulator $\mathcal{S}$ collects $\mathbf{d}^* \in \mathbb{Z}_q^m$ from Successlist satisfying $g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{d}^*) = f(\mathbf{r}_1^*, \mathbf{v}^*) = \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{d}^*$. This yields a collision in $g_{\mathbf{A}_{\mathsf{nr}}}$ as the forgery $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*, \mathbf{v}^*)$ gives $g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{z}^*) = \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z}^* = \mathbf{y}_1^* = g_{\mathbf{A}_{\mathsf{nr}}}(\mathbf{d}^*)$. From the preimage min-entropy of $\mathbf{d}^*$ (see Definition 8 in Section 2.1) it follows that the signature $\mathbf{z}^* \neq \mathbf{d}^*$, except with negligible probability $2^{-\omega(\log n)}$. Thus $\mathcal{S}$ outputs a valid collision in $g_{\mathbf{A}_{\mathsf{nr}}}$ with probability negligibly close to $1 - 2^{-\omega(\log n)}$.

Now, we show that $\mathcal{S}$ finds the collision in the PSF $g_{\mathbf{A}^*_{\mathsf{nr}}}$ with probability atleast ϵ'. To do so, we analyze the three events required for $\mathcal{S}$ to succeed.

A : $\mathcal{S}$ does not abort as a result of any of $\mathcal{F}$'s Corrupt and SignNR queries.
B : $\mathcal{F}$ generates a valid forgery $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*, \mathbf{v}^*)$.
C : Event B occurs and $\mathsf{nr} = \mathsf{nr}^*$ with $(M^*, \mathbf{r}_1^*, \mathbf{v}^*, \mathbf{d}^*, \mathbf{y}_1^*) \in$ Successlist.

The simulator $\mathcal{S}$ succeeds if all these events occur. Note that

$$\mathsf{Pr}[A \cap B \cap C] = \mathsf{Pr}[A] \cdot \mathsf{Pr}[B|A] \cdot \mathsf{Pr}[C|A \cap B].$$

Then using Lemma 2, Lemma 3 and Lemma 4 described below, the simulator $\mathcal{S}$ produces a collision to the preimage sampleable function with probability atleast $\frac{\epsilon}{\alpha e^2} \geq \epsilon'$. Simulator $\mathcal{S}$'s running time is same as $\mathcal{F}$'s running time together with the time it takes to respond to α many queries to each of the oracles Create, $\mathsf{Corrupt}$, SignNE, SignNR, cfORds, Q_H hash queries and Q_f queries. Each query requires some computation time, let us assume that t_{gen} is the time taken. Hence, the total running time is atmost $t + t_{\mathsf{gen}}(Q_H + Q_f + 5\alpha) \leq t'$. This completes the proof of Theorem 6. □

Lemma 2. $\mathsf{Pr}[A] \geq \frac{1}{e^2}$.

Proof. We prove by induction that after $\mathcal{F}$ makes i many $\mathsf{Corrupt}$ queries, the probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})^i$. The claim is trivially true for $i = 0$. The probability that the forger $\mathcal{F}$ makes a $\mathsf{Corrupt}$ query for the user $u = \mathsf{nr}^*$ is atmost $\frac{1}{\alpha}$ (the total number of allowed $\mathsf{Corrupt}$ query is α). Thus, the probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})$. Using the inductive hypothesis, the probability that $\mathcal{S}$ does not abort after i many $\mathsf{Corrupt}$ queries is atleast $(1 - \frac{1}{\alpha})^i$. Similarly, the probability that $\mathcal{S}$ does not abort after j many SignNR queries is atleast $(1 - \frac{1}{\alpha})^j$. Thus, the total probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})^{i+j}$. Since $\mathcal{F}$ makes atmost α many $\mathsf{Corrupt}$ and SignNR queries each, the probability that $\mathcal{S}$ does not abort as a result of all the $\mathsf{Corrupt}$ and SignNR queries is atleast $(1 - \frac{1}{\alpha})^{2\alpha} \geq \frac{1}{e^2}$. □

The public key given to $\mathcal{F}$ is from the same distribution as a public key produced by algorithm $\mathsf{NS.keygen}$. The distribution of responses to all the queries Create, $\mathsf{Corrupt}$, Hash, $\mathsf{pseudorandom}$, SignNE, SignNR, cfORds are as in the real attack. All responses to $\mathsf{Corrupt}$, SignNE and SignNR are valid. Therefore $\mathcal{F}$ will produce a valid message-signature pair with probability atleast ϵ. Hence, $\mathsf{Pr}[B|A] \geq \epsilon$. Thus we have the following lemma.

Lemma 3. $\mathsf{Pr}[B|A] \geq \epsilon$. □ *(of Lemma 3)*

Given that the events A and B have happened, the simulator $\mathcal{S}$ will declare failure only if $\mathsf{nr} \neq \mathsf{nr}^*$. So, $\mathsf{Pr}[C|A \cap B] = \mathsf{Pr}[\mathsf{nr} = \mathsf{nr}^*] = \frac{1}{\alpha}$. Hence we have the following lemma.

Lemma 4. $\mathsf{Pr}[C|A \cap B] = \frac{1}{\alpha}$. □ *(of Lemma 4)*

Theorem 7. *The* NS=(setup, keygen, neSign, nrSign, verify, cfORds=(neTM, vrTM)) *described in Section 4 is secure under the unforgeability against malicious nominator in the random oracle model as per Definition 10 for the security game given in Figure 2 assuming the existence of collision-resistant preimage sampleable function (*CRPSF*). Concretely, suppose there is a forger $\mathcal{F}$ that has advantage ϵ against the scheme* NS*. Suppose $\mathcal{F}$ makes atmost $Q_H > 0$ hash queries to H and $Q_f > 0$ pseudorandom queries to f. Then there is an algorithm $\mathcal{S}$ that breaks the collision resistant property of* CRPSF *with probability*

$\mathsf{Pr}\big[\mathcal{F}(\bar{\boldsymbol{A}}, \boldsymbol{x}, n, m, q) \rightarrow \boldsymbol{x}' \;|((g_{\bar{\boldsymbol{A}}}(\boldsymbol{x}) = g_{\bar{\boldsymbol{A}}}(\boldsymbol{x}') \wedge (\boldsymbol{x} \neq \boldsymbol{x}')), \bar{\boldsymbol{A}} \in \mathbb{Z}_q^{n\times m}, \boldsymbol{x} \in \mathbb{Z}_q^m, ||\boldsymbol{x}|| \leq \beta, \beta = 2\sigma\sqrt{m}, \sigma = \Omega(\sqrt{n \log q} \log n), m \geq 5n \log q, q = \mathcal{O}(n^3), n = \mathcal{O}(\lambda)\big]$

atleast ϵ' where $\epsilon \geq \alpha e^2 \cdot \epsilon'$. Here e is the base of the natural logarithm. The running time of $\mathcal{S}$ is $O(time(\mathcal{F}))$.

Proof. Let a PPT forger $\mathcal{F}$ with the secret key of a nominator forges our proposed nominative signature scheme NS with non-negligible advantage ϵ. We construct an algorithm $\mathcal{S}$ that breaks the collision-resistant property of PSF with probability close to ϵ. The algorithm $\mathcal{S}$ plays the role of the simulator in the security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{uf-mnr}}(\lambda)$ (Figure 2) and interacts with $\mathcal{F}$ as follows:

1. The simulator $\mathcal{S}$ runs $\mathsf{NS.setup}(\lambda)$ and honestly generates the public parameters $\mathcal{Y} = (n, q, m, \sigma, \delta, H, f)$ where integers n and q are of sizes $\mathcal{O}(\lambda)$ and $\mathcal{O}(n^3)$ respectively, real number σ of size $\Omega(\sqrt{n \log q} \log n)$ is the standard deviation of the discrete Gaussian distribution $D_{\Lambda,\sigma}$, an error bound $\delta = 2\sigma\sqrt{m}$ with $m \geq 5n\lceil \log q \rceil$, a hash function $H : \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2 \rightarrow \mathbb{Z}_q^n$ with $\mathcal{M}$ as the message space and a collision resistant pseudorandom function $f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \rightarrow \mathbb{Z}_q^n$. The simulator $\mathcal{S}$ sends $\mathcal{Y}$ to the forger $\mathcal{F}$, fixes the uncorrupted nominee ne^* and the public key $\mathsf{pk}_{\mathsf{ne}^*} \in \mathbb{Z}_q^{n\times m}$ of the nominee ne^* uniformly at random.

2. The forger $\mathcal{F}$ makes polynomially many, say α, queries to $\mathcal{S}$ for each of the following oracles where Hlist, $\mathsf{PRFlist}$, $\mathsf{Successlist}$, $\mathsf{LsignNE}$, $\mathsf{LsignNR}$, $\mathsf{LcfORds}$ are the private lists maintained by the simulator $\mathcal{S}$ (which are initially empty).

− Create Query: On receiving this query on a user $a \neq \mathsf{ne}^*$, the simulator $\mathcal{S}$ runs $\mathsf{NS.keygen}(\mathcal{Y}, a) \rightarrow (\mathsf{pk}_a, \mathsf{sk}_a)$ where $\mathsf{pk}_a = \mathbf{A}_a \in \mathbb{Z}_q^{n\times m}$ and $\mathsf{sk}_a = \mathsf{T}_{\mathbf{A}_a} \in \mathbb{Z}^{m\times m}$. For the case $a = \mathsf{ne}^*$, the simulator $\mathcal{S}$ sends the already fixed public key $\mathsf{pk}_{\mathsf{ne}^*}$. In this case $\mathsf{sk}_{\mathsf{ne}^*} = \perp$. The simulator $\mathcal{S}$ stores $(\mathsf{pk}_a, \mathsf{sk}_a) \in \mathsf{Lcreate}$ and sends pk_a to the forger $\mathcal{F}$. When user a plays the role of a nominee, we set $\mathbf{B}_a = \mathbf{A}_a$ and $\mathsf{T}_{\mathbf{B}_a} = \mathsf{T}_{\mathbf{A}_a}$. Observe that the simulation of this query is exactly same as in the original protocol.

− $\mathsf{Corrupt}$ Query: On receiving this query on a user a from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $a \neq \mathsf{ne}^*$ and $(\mathsf{pk}_a = \mathbf{A}_a, \mathsf{sk}_a = \mathsf{T}_{\mathbf{A}_a}) \in \mathsf{Lcreate}$. If not, $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ sends sk_a to $\mathcal{F}$ and stores pk_a in the list $\mathsf{Lcorrupt}$.

− Hash Query: To answer hash queries, $\mathcal{S}$ maintains a list Hlist for the hash function H. This list stores records of the form $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = (M, \mathbf{A}_u, \mathbf{B}_v) \in \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2$ and $\mathbf{y} \in \mathbb{Z}_q^n$. The stored value is returned on the queried $\mathbf{x} \in \mathcal{M} \times (\mathbb{Z}_q^{n\times m})^2$ if it has already been queried before. Otherwise, a fresh random value $\mathbf{y} = H(\mathbf{x}) \in \mathbb{Z}_q^n$ is generated when $v \neq \mathsf{ne}^*$. When $v = \mathsf{ne}^*$, the simulator chooses a small random vector $\mathbf{w} \leftarrow \mathsf{SampleD}(1^n)$ i.e., $\mathbf{w} \in \mathbb{Z}_q^m$ with $||\mathbf{w}|| \leq \sigma\sqrt{m}$ and computes $\mathbf{y} = \mathbf{B}_{\mathsf{ne}^*} \cdot \mathbf{w}$ and $(\mathbf{x}, \mathbf{y})$ is included in Hlist. The simulator $\mathcal{S}$ returns $\mathbf{y}$ to $\mathcal{F}$. The simulator stores $(\mathbf{x}, \mathbf{w})$ in the list $\mathsf{Succlist}$. Distributions of hash queries are exactly same as that in the original protocol as long as H is treated as a random oracle.

− PRF Query: To respond pseudorandom queries, $\mathcal{S}$ maintains a list $\mathsf{PRFlist}$ for the collision resistant pseudorandom function f. On receiving a query $(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$ from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1) \in \mathsf{PRFlist}$ and returns $\mathbf{y}_1$ to $\mathcal{F}$ if so. Otherwise, $\mathcal{S}$ computes $\mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$ honestly similar to that in the real protocol, passes $\mathbf{y}_1 \in \mathbb{Z}_q^n$ to the forger $\mathcal{F}$ and stores $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in $\mathsf{PRFlist}$. Distributions of PRF queries are exactly same as that in the original protocol

− SignNE Query: In response to this query on (M, v, u) from $\mathcal{F}$, the simulator $\mathcal{S}$ verifies whether $v \neq \mathsf{ne}^*$. If not, $\mathcal{S}$ returns $\perp$. Otherwise, the simulator $\mathcal{S}$ returns the nominative signature $(\mathsf{Sig}_{M,v,u} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})) \leftarrow \mathsf{NS.neSign}(\mathcal{Y}, \mathsf{sk}_v, \mathsf{pk}_v, \mathsf{pk}_u, M)$ to $\mathcal{F}$ and stores $\mathsf{Sig}_{M,v,u}$ in the list $\mathsf{LsignNE}$. Note that $\Pi_{\mathcal{R}_f}$ is a zero knowledge on the relation $\mathcal{R}_f = \{\text{public}(f, \mathbf{v}, \mathbf{y}_1, \mathcal{Y}),\ \text{private }(\mathbf{r}_1) \mid y_1 = f(\mathbf{r}_1, \mathbf{v})\}$. The distribution of $\mathsf{Sig}_{M,v,u}$ is similar to that in the real protocol. The simulator $\mathcal{S}$ updates the current internal state $\mathsf{state}_v = \mathsf{state}_v \cup (\mathbf{r}_1, \mathbf{v}, M, u)$ and stores $(\mathsf{Sig}_{M,v,u}, \mathsf{state}_v)$ in the list $\mathsf{LsignNE}$.

− SignNR Query: In response to this query on $\mathsf{Sig}_{M,v,u}=(M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f})$ from $\mathcal{F}$, the simulator $\mathcal{S}$ returns the nominative signature $(\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})) \leftarrow \mathsf{NS.nrSign}(\mathcal{Y}, \mathsf{sk}_u, \mathsf{pk}_u, \mathsf{pk}_v, M, \mathsf{Sig}_{M,v,u})$ to $\mathcal{F}$ and stores $(\mathsf{Sig}_{M,v,u}, \mathsf{nsig}_{M,v,u})$ in the list $\mathsf{LsignNR}$. The distribution of $\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})$ is similar to that in the real protocol.

− cfORds Query: The simulator $\mathcal{S}$, on receiving this query on $\mathsf{nsig}_{M,v,u} = (\mathbf{z}, \mathbf{v})$ from $\mathcal{F}$. The simulator $\mathcal{S}$ returns $\perp$ if $v = \mathsf{ne}^*$. Otherwise, $\mathcal{S}$ calls $\mathsf{NS.cfORds.neTM}(\mathcal{Y}, \mathsf{state}_v, \mathsf{pk}_v, \mathsf{pk}_u, M, \mathsf{nsig}_{M,v,u}) \rightarrow (\mu, \Pi_{\mathcal{R}})$ to execute the following steps using $\mathsf{pk}_v = \mathbf{B}_v$, $\mathsf{pk}_u = \mathbf{A}_u$ and $(\mathbf{r}_1, \mathbf{v}, M, u)$ contained in state_v.

(i) extract $\mathbf{r}_1, \mathbf{v}$ from $(\mathbf{r}_1, \mathbf{v}, M, u)$ contained in state_v,

(ii) generate a zero knowledge proof $\Pi_{\mathcal{R}}$ for the relation $\mathcal{R} = \{\text{public}(\mathbf{A}_u, \mathbf{B}_v, \mathbf{v}, \mathbf{z}, M, \mathcal{Y}),\ \text{private }(\mathbf{r}_1) \mid \mathbf{B}_v \cdot \mathbf{v} = H(M, \mathbf{A}_u, \mathbf{B}_v), \mathbf{A}_u \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\}$ with $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in $\mathsf{PRFlist}$ and $((M, \mathbf{A}_u, \mathbf{B}_v), H(M, \mathbf{A}_u, \mathbf{B}_v))$ in Hlist using the algorithm $\mathsf{ZKB.prove}$ of the non-interactive zero knowledge proof system $\mathsf{ZKB}{+}{+}$ as described in Section 2.2,

(iii) return $(\mu = 1, \Pi_{\mathcal{R}})$ to the verifier vr if $\mathsf{NS.verify}(\mathcal{Y}, \mathsf{state}_v, \mathsf{pk}_v, \mathsf{pk}_u, M, \mathsf{nsig}_{M,v,u})$ is valid, otherwise, send $(\mu = 0, \Pi_{\mathcal{R}})$ to vr,

(iv) record $(\mathsf{nsig}_{M,v,u}, \mu, \Pi_{\mathcal{R}})$ in the list $\mathsf{LcfORds}$.

The proof $\Pi_{\mathcal{R}}$ is treated as a confirmation (disavowal) proof of the relation $\mathcal{R}$ when $\mu = 1$ ($\mu = 0$). Note that, the simulation of $(\mu, \Pi_{\mathcal{R}})$ is exactly same as in the real protocol.

3. Finally, $\mathcal{F}$ produces a forgery $\Sigma^* = (\mathsf{Sig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (M^*, \mathbf{v}^*, \mathbf{y}_1^*, \Pi^*_{\mathcal{R}_f}), \mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}^*, \mathbf{v}^*))$ on a message $M^* \in \mathcal{M}$ against an uncorrupted nomiee ne and a *corrupted* nominator nr i.e., $\mathsf{pk_{nr}} \in \mathsf{Lcorrupt}$. If the forgery Σ^* is valid, the following conditions must hold.

(a) $\mathsf{NS.verify}(\mathcal{Y}$, $\mathsf{state_{ne}}$, $\mathsf{pk_{ne}}$, $\mathsf{pk_{nr}}$, M^*, $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}) \rightarrow \mathsf{valid}$,

(b) $\mathsf{pk_{ne}} \notin \mathsf{Lcorrupt}$,

(c) $(\mathsf{Sig}^*_{M^*,\mathsf{ne},\mathsf{nr}}, \mathsf{state_{ne}}) \notin \mathsf{LsignNE}$ and

(d) $(\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}$, μ, $\Pi_{\mathcal{R}}) \notin \mathsf{LcfORds}$.

When $\mathsf{ne} = \mathsf{ne}^*$, note that both the equations $\mathbf{A_{nr}} \cdot \mathbf{z}^* = \mathbf{y}_1^* \bmod q$ and $\mathbf{B_{ne}} \cdot \mathbf{v}^* = \mathbf{y}^* \bmod q$ with $||\mathbf{z}^*||, ||\mathbf{v}^*|| \leq \sigma\sqrt{m}$ must be satisfied when $\mathcal{F}$ produces a valid forgery Σ^*. This implies that the pair $((M^*, \mathbf{A_{nr}}, \mathbf{B_{ne}}), H(M^*, \mathbf{A_{nr}}, \mathbf{B_{ne}})) \in \mathsf{Hlist}$, $((\mathbf{r}_1^*, \mathbf{v}^*), f(\mathbf{r}_1^*, \mathbf{v}^*)) \in \mathsf{PRFlist}$ and $(M^*, \mathbf{A_{nr}}, \mathbf{B_{ne}}, \mathbf{w}^*) \in \mathsf{Succlist}$. The simulator $\mathcal{S}$ collects $\mathbf{w}^* \in \mathbb{Z}_q^m$ from $\mathsf{Succlist}$ satisfying $g_{\mathbf{B_{ne}}}(\mathbf{w}^*) = \mathbf{B_{ne}} \cdot \mathbf{w}^*$. This yields a collision in $g_{\mathbf{B_{ne}}}$ as the forgery Σ^* gives $g_{\mathbf{B_{ne}}}(\mathbf{v}^*) = \mathbf{B_{ne}} \cdot \mathbf{v}^* = g_{\mathbf{B_{ne}}}(\mathbf{w}^*)$. From the preimage min-entropy of $\mathbf{w}^*$ (see Definition 8 in Section 2.1) it follows that the signature $\mathbf{v}^* \neq \mathbf{w}^*$, except with negligible probability $2^{-\omega(\log n)}$. Thus $\mathcal{S}$ outputs a valid collision in $g_{\mathbf{B_{ne}}}$ with probability negligibly close to $1 - 2^{-\omega(\log n)}$.

Now, we show that $\mathcal{S}$ finds the collision in PSF with probability atleast ϵ'. To do so, we analyze the three events required for $\mathcal{S}$ to succeed.

A : $\mathcal{S}$ does not abort as a result of any of $\mathcal{F}$'s $\mathsf{Corrupt}$ and SignNE queries.
B : $\mathcal{F}$ generates a valid forgery Σ^*.
C : Event B occurs and $\mathsf{ne} = \mathsf{ne}^*$ with $(M^*, \mathbf{A_{nr}}, \mathbf{B_{ne}}, \mathbf{w}^*) \in \mathsf{Succlist}$.

The simulator $\mathcal{S}$ succeeds if all these events occur. Note that

$$\mathsf{Pr}[A \cap B \cap C] = \mathsf{Pr}[A] \cdot \mathsf{Pr}[B|A] \cdot \mathsf{Pr}[C|A \cap B].$$

Then using the Lemma 2, Lemma 3 and Lemma 4 described below, the simulator $\mathcal{S}$ produces a collision to the PSF with probability atleast $\frac{\epsilon}{\alpha e^2} \geq \epsilon'$. Simulator $\mathcal{S}$'s running time is same as $\mathcal{F}$'s running time together with the time it takes to respond to α many Create, $\mathsf{Corrupt}$, SignNE, SignNR, cfORds queries, Q_H hash queries and Q_f pseudorandom queries. Each query requires some computation time, let us assume that t_{gen} is the time taken. Hence, the total running time is atmost $t + t_{\mathsf{gen}}(Q_H + Q_f + 5\alpha) \leq t'$. This completes the proof of Theorem 7. □

Lemma 5. $\mathsf{Pr}[A] \geq \frac{1}{e^2}$.

Proof. We prove by induction that after $\mathcal{F}$ makes i many $\mathsf{Corrupt}$ queries, the probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})^i$. The claim is trivially true for $i = 0$. The probability that the forger $\mathcal{F}$ makes a $\mathsf{Corrupt}$ query for the user $u = \mathsf{ne}^*$ is atmost $\frac{1}{\alpha}$ (the total number of allowed $\mathsf{Corrupt}$ query is α). Thus, the probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})$. Using the inductive hypothesis, the probability that $\mathcal{S}$ does not abort after i many $\mathsf{Corrupt}$ queries is atleast $(1 - \frac{1}{\alpha})^i$. Similarly, the probability that $\mathcal{S}$ does not abort after j many SignNE queries is atleast $(1 - \frac{1}{\alpha})^j$. Thus, the total probability that $\mathcal{S}$ does not abort is atleast $(1 - \frac{1}{\alpha})^{i+j}$. Since $\mathcal{F}$ makes atmost α many $\mathsf{Corrupt}$ and SignNE queries each, the probability that $\mathcal{S}$ does not abort as a result of all the $\mathsf{Corrupt}$ and SignNE queries is atleast $(1 - \frac{1}{\alpha})^{2\alpha} \geq \frac{1}{e^2}$. □

The public key given to $\mathcal{F}$ is from the same distribution as a public key produced by algorithm $\mathsf{NS.keygen}$. The distribution of responses to all the queries Create, $\mathsf{Corrupt}$, Hash, $\mathsf{pseudorandom}$, SignNE, SignNR, cfORds are as in the real attack. All responses to $\mathsf{Corrupt}$, SignNE and SignNR are valid. Therefore $\mathcal{F}$ will produce a valid message-signature pair with probability atleast ϵ. Hence, $\mathsf{Pr}[B|A] \geq \epsilon$. Thus we have the following lemma.
Lemma 6. $\mathsf{Pr}[B|A] \geq \epsilon$. □ *(of Lemma 6)*

Given that the events A and B have happened, the simulator $\mathcal{S}$ will declare failure only if $\mathsf{ne} \neq \mathsf{ne}^*$. So, $\mathsf{Pr}[C|A \cap B] = \mathsf{Pr}[\mathsf{ne} = \mathsf{ne}^*] = \frac{1}{\alpha}$. Hence we have the following lemma.

Lemma 7. $\mathsf{Pr}[C|A \cap B] = \frac{1}{\alpha}$. □ *(of Lemma 7)*

Theorem 8. *Assume that* Problem-I *4 defined in Section 2 is hard. Then the construction of our nominative signature scheme* NS={setup, keygen, neSign, nrSign, verify, cfORds =(neTM, vrTM)} *described in Section 4 is secure under invisibility as per the Definition 11 for the security game given in Figure 3.*

Proof. Let $\mathcal{F}$ be a PPT distinguisher that breaks the invisibility security game $\mathsf{Exp}_{\mathcal{F}}^{\mathsf{invis}}(\lambda, b)$ defined in Figure 3 with non-negligible advantage ϵ. We will construct a simulator $\mathcal{S}$ that breaks an instance of Problem-I using $\mathcal{F}$ as a subroutine i.e., given two vectors $\mathbf{d} \in \mathbb{Z}_q^m$, $\mathbf{w} \in \mathbb{Z}_q^n$ with $q = \mathcal{O}(n^3)$, $m \geq 5n\lceil\log q\rceil$ and $\sigma = \Omega(\sqrt{n \log q} \log n)$, the simulator $\mathcal{S}$ interacts with $\mathcal{F}$ and distinguishes whether $\mathbf{w} = f(\mathbf{k}, \mathbf{d})$ for some $\mathbf{k} \in \mathbb{Z}_q^m$ or $\mathbf{w}$ is uniformly chosen from $\mathbb{Z}_q^n$. The simulator $\mathcal{S}$ sets $\mathcal{Y}$=(n, q, m, σ, β, H, f) where $\beta = 2\sigma\sqrt{m}$, $f : \mathbb{Z}_q^m \times \mathbb{Z}_q^m \to \mathbb{Z}_q^n$ and $H : \{0,1\}^* \to \mathbb{Z}_q^n$ are cryptographically secure hash functions.

The simulator $\mathcal{S}$ speculates an index $k \in \{1, 2, \ldots, q_H\}$. More specifically, $\mathcal{S}$ guesses that $\mathcal{F}$ makes k-th H_1 query that is used by $\mathcal{F}$ to output a challenged tuple.

1. The simulator $\mathcal{S}$ sends $(\mathcal{Y}, \mathsf{pk}_{\mathsf{ne}}^*)$ to the forger $\mathcal{F}$.

2. The simulator responds to the distinguisher $\mathcal{F}$ on queries Create, CorruptNR, Hash, PRF, SignNE, SignNR, cfORds as follows and maintains seven lists Lcreate, Lcorrupt, Hlist, PRFlist, LsignNE, LsignNR, LcfORds.

- Create Query. On receiving this query for a user a from $\mathcal{F}$, the simulator $\mathcal{S}$ runs NS.keygen($\mathcal{Y}$, a)→(pk_a, sk_a) where pk_a=$\mathbf{B}_a \in \mathbb{Z}_q^{n\times m}$ and sk_a=$\mathsf{T}_{\mathbf{B}_a} \in \mathbb{Z}_q^{m\times m}$ by calling the algorithm TrapGen(n, m, q)→ ($\mathbf{B}_a$, $\mathsf{T}_{\mathbf{B}_a}$). The simulator $\mathcal{S}$ sends the public key pk_a to $\mathcal{F}$ and stores the public-secret key pair (pk_a, sk_a) in its private list Lcreate.

- Corrupt Query. On receiving the query on a user a from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $(\mathsf{pk}_a = \mathbf{B}_a, \mathsf{sk}_a = \mathsf{T}_{\mathbf{B}_a}) \in$ Lcreate. If not, $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ sends sk_a to $\mathcal{F}$ and stores pk_a in the list Lcorrupt.

- Hash query. If the answer to the received tuple $\mathbf{x} = (M, \mathbf{A}_u, \mathbf{B}_v)$ has already made, the simulator returns the same value stored in Lhash. For k-th H_1 query, the simulator responds as $H_1(\mathbf{x}) = \mathbf{y}^* = \mathbf{B}_{\mathsf{ne}^*} \cdot \mathbf{d} \bmod q$ and stores $(\mathbf{x}, H_1(\mathbf{x}), \mathbf{d})$ in the list Lsuccess. Otherwise, the simulator $\mathcal{S}$ honestly generates as in the real protocol. The simulator finally sends $(\mathbf{x}, H_1(\mathbf{x}))$ to the forger $\mathcal{F}$ and stores $(\mathbf{x}, H_1(\mathbf{x}))$ in the list Lhash and $(\mathbf{x}, H_1(\mathbf{x}), \mathbf{d})$ in the list Lsuccess.

- PRF query. To respond pseudorandom queries, $\mathcal{S}$ maintains a list PRFlist for the collision resistant pseudorandom function f. On receiving a query $(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$ from $\mathcal{F}$, the simulator $\mathcal{S}$ checks whether $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1) \in$ PRFlist and returns $\mathbf{y}_1$ to $\mathcal{F}$ if so. Otherwise, $\mathcal{S}$ computes $\mathbf{y}_1 = f(\mathbf{r}_1, \mathbf{v}) \in \mathbb{Z}_q^n$ honestly similar to that in the real protocol, passes $\mathbf{y}_1 \in \mathbb{Z}_q^n$ to the forger $\mathcal{F}$ and stores $((\mathbf{r}_1, \mathbf{v}), \mathbf{y}_1)$ in PRFlist. Distributions of PRF queries are exactly same as that in the original protocol.

- SignNE Query. In response to the tuple (v, u, M) by $\mathcal{F}$, the challenger $\mathcal{S}$ checks whether $(\mathsf{pk}_u = \mathbf{A}_u, \mathsf{sk}_u = \mathsf{T}_{\mathbf{A}_u}), (\mathsf{pk}_v = \mathbf{B}_v, \mathsf{sk}_v = \mathsf{T}_{\mathbf{B}_v}) \in$ Lcreate. If not, $\mathcal{S}$ returns $\perp$. Otherwise, $\mathcal{S}$ returns the signature $\mathsf{Sig}_{M,v,u} = (M, \mathbf{v}, \mathbf{y}_1, \Pi_{\mathcal{R}_f}) \leftarrow$ NS.neSign($\mathcal{Y}$, sk_v, pk_v, pk_u, M) to $\mathcal{D}$ and stores ($\mathsf{Sig}_{M,v,u}$, state_v) in the list LsignNE where $\mathsf{state}_{\mathsf{ne}}$ contains tuple of the form $(\mathbf{r}_1, \mathbf{v}, M, v)$.

- SignNR Query. In response to this query on $\mathsf{Sig}_{M,v,u}$ from $\mathcal{F}$, the challenger $\mathcal{S}$, returns the nominative signature $\mathsf{nsig}_{M,v,u} \leftarrow$ NS.nrSign($\mathcal{Y}$, sk_u, pk_u, pk_v, M, $\mathsf{Sig}_{M,v,u}$) to $\mathcal{F}$ and stores ($\mathsf{Sig}_{M,v,u}$, $\mathsf{nsig}_{M,v,u}$) in the list LsignNR.

- cfORds Query. The challenger $\mathcal{S}$ responses on receiving this query on $\mathsf{nsig}_{M,v,u}$ from $\mathcal{F}$ by checking if ($\mathsf{Sig}_{M,v,u}$, $\mathsf{nsig}_{M,v,u}$) $\in$ LsignNR. If not, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ extracts state_v from ($\mathsf{Sig}_{M,v,u}$, state_v) $\in$ LSignNE and returns $(\mu, \Pi_{\mathcal{R}}) \leftarrow$ NS.cfORds.neTM($\mathcal{Y}$, state_v, pk_v, pk_u, M, $\mathsf{nsig}_{M,v,u}$) to $\mathcal{F}$. The challenger $\mathcal{F}$ stores ($\mathsf{nsig}_{M,v,u}$, μ, $\Pi_{\mathcal{R}}$) in the list LcfORds.

3. At any point of the game, $\mathcal{F}$ submits a *challenge* tuple (M, ne, nr) where M is a message to be signed with ne as the nominee and nr as the nominator such that $\mathsf{pk}_{\mathsf{ne}} \in$ Lcreate and $\mathsf{pk}_{\mathsf{nr}} \in$ Lcorrupt.

4. The challenger chooses a random bit $b \in \{0,1\}$. If $b = 1$, the challenger $\mathcal{S}$ generates $\mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}} \leftarrow$ NS.neSign($\mathcal{Y}$, $\mathsf{sk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M^*), $\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}} \leftarrow$ NS.nrSign($\mathcal{Y}$, $\mathsf{sk}_{\mathsf{nr}}$, $\mathsf{pk}_{\mathsf{nr}}$, $\mathsf{pk}_{\mathsf{ne}}$, M^*, $\mathsf{Sig}_{M^*,\mathsf{ne},\mathsf{nr}}$) and sets $\mathsf{K}_b = \mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$. If the bit $b = 0$ then $\mathcal{S}$ finds $\mathbf{z} \in \mathbb{Z}_q^m$ satisfying $\mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = \mathbf{w} \bmod q$ and returns the nominative signature $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{d}, \mathbf{w})$.

5. The distinguisher $\mathcal{F}$ observes K_b, outputs a guess b' and wins the game if $b' = b$ and $\mathsf{pk}_{\mathsf{ne}} \notin$ Lcorrupt.

Note that one of the verification steps of $\mathsf{nsig}_{M,\mathsf{ne},\mathsf{nr}} = (\mathbf{z}, \mathbf{d}, \mathbf{w})$ is to check whether $\mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{d}) \bmod q$ which in turn checks whether $\mathbf{w} = f(\mathbf{r}_1, \mathbf{d})$ for some $\mathbf{r}_1 \in \mathbb{Z}_q^m$. Thus the simulator $\mathcal{S}$ will correctly solve the given problem-I whenever $\mathcal{F}$ correctly solves the invisibility challenge. □

Theorem 9. *Our nominative signature scheme* NS={setup, keygen, neSign, nrSign, verify, cfORds =(neTM, vrTM)} *described in Section 4 is secure under non-repudiation if no PPT cheating nominee has a non negligible advantage in game 4 assuming the completeness and soundness properties of the zero knowledge proof system* ZKB++.

Proof. Let $\mathcal{F}$ be the cheating nominee and $\mathcal{S}$ be the challenger as given in the game experiment 4.

1. The simulator $\mathcal{S}$ runs NS.setup(λ) $\rightarrow \mathcal{Y}$ =(n, q, m, σ, β, H, f).

2. The simulator responds to the distinguisher $\mathcal{F}$ on queries Create, CorruptNR, Hash, PRF, SignNE, SignNR, cfORds exactly same as in the proof of Theorem 8.

3. The cheating nominee prepares the tuple $(M^*, \mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}, \gamma = 1 - \mu)$ where ne is any nominee with $\mathsf{pk}_{\mathsf{ne}} \in$ Lcorrupt, nr is a nominator such that ($\mathsf{pk}_{\mathsf{nr}}$, $\mathsf{sk}_{\mathsf{nr}}$)$\in$ Lcreate, $\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}$ is a signature on M^* and $(\mathsf{nsig}_{M^*,\mathsf{ne},\mathsf{nr}}, \mu, \Pi_{\mathcal{R}}) \in$ LcfORds where $\Pi_{\mathcal{R}}$ is a zero knowledge proof for the relation $\mathcal{R}$ and μ is a bit. Recall that $\mu = 1$ if NS.verify($\mathcal{Y}$, $\mathsf{state}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{ne}}$, $\mathsf{pk}_{\mathsf{nr}}$, M, $\mathsf{nsig}^*_{M^*,\mathsf{ne},\mathsf{nr}}$)$\rightarrow$ valid; otherwise $\mu = 0$.

4. If $\mu = 1$, the cheating nominee executes the disavowal proof. The probability that the verifier accepts the proof is $\epsilon \in [0, \frac{1}{2})$ by the *completeness* property of the zero knowledge proof $\Pi_{\mathcal{R}}$. Here $\mathcal{R}$ is a relation $\mathcal{R} = \big\{\text{public}(\mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}, \mathbf{v}, \mathbf{z}, M, \mathcal{Y}),\ \text{private }(\mathbf{r}_1) \mid \mathbf{B}_{\mathsf{ne}} \cdot \mathbf{v} = H(M, \mathbf{A}_{\mathsf{nr}}, \mathbf{B}_{\mathsf{ne}}), \mathbf{A}_{\mathsf{nr}} \cdot \mathbf{z} = f(\mathbf{r}_1, \mathbf{v})\big\}$.

If $\mu = 0$ then the cheating nominee executes the confirmation proof. The probability that the veifier accepts the proof is $\epsilon \in [0, \frac{1}{2})$ by the *soundeness* property of the zero knowledge proof $\Pi_{\mathcal{R}}$. □

5 Conclusion

In summary, the goal of this work is the construction of a nominative signature scheme secure against quantum adversary. Our construction provides unforgeability, invisibility, impersonation and non-repudiation. The scheme is secure under non-repudiation in the quantum random oracle model. The proposed construction is computationally efficient as lattice computations (addition and multiplication of vectors) are very efficient as compared to exponentiation and bilinear pairing.

References

[1] Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 99–108. ACM (1996)

[2] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. ACM (2017)

[3] Chaum, D.: Designated confirmer signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 86–91. Springer (1994)

[4] Chaum, D., Van Antwerpen, H.: Undeniable signatures. In: Conference on the Theory and Application of Cryptology. pp. 212–216. Springer (1989)

[5] Cramer, R., Damgård, I., MacKenzie, P.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: International Workshop on Public Key Cryptography. pp. 354–372. Springer (2000)

[6] Dennis, Y., CHANG, S., et al.: A more efficient convertible nominative signature. In: SECRYPT 2007-International Conference on Security and Cryptography (2007)

[7] Derler, D., Ramacher, S., Slamanig, D.: Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. In: International Conference on Post-Quantum Cryptography. pp. 419–440. Springer (2018)

[8] Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 416–426. Citeseer (1990)

[9] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 186–194. Springer (1986)

[10] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the fortieth annual ACM symposium on Theory of computing. pp. 197–206. ACM (2008)

[11] Guo, L., Wang, G., Wong, D.S.: Further discussions on the security of a nominative signature scheme (2007)

[12] Huang, Q., Liu, D.Y., Wong, D.S.: An efficient one-move nominative signature scheme. International Journal of Applied Cryptography 1(2), 133–143 (2008)

[13] Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 143–154. Springer (1996)

[14] Kansal, M., Dutta, R., Mukhopadhyay, S.: Construction for a nominative signature scheme from lattice with enhanced security. In: International Conference on Codes, Cryptology, and Information Security. pp. 72–91. Springer (2019)

[15] Kim, S.J., Park, S.J., Won, D.H.: Nominative signatures. In: ICEIC: International Conference on Electronics, Informations and Communications. pp. 68–71 (1995)

[16] Libert, B., Ling, S., Mouhartem, F., Nguyen, K., Wang, H.: Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II 22. pp. 373–403. Springer (2016)

[17] Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–31. Springer (2016)

[18] Liu, D.Y., Wong, D.S., Huang, X., Wang, G., Huang, Q., Mu, Y., Susilo, W.: Formal definition and construction of nominative signature. In: International Conference on Information and Communications Security. pp. 57–68. Springer (2007)

[19] Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. In: Advances in Cryptology–CRYPTO. vol. 85, p. 447 (1985)

[20] Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: International workshop on fast software encryption. pp. 371–388. Springer (2004)

[21] Schuldt, J.C., Hanaoka, G.: Non-transferable user certification secure against authority information leaks and impersonation attacks. In: International Conference on Applied Cryptography and Network Security. pp. 413–430. Springer (2011)

[22] Steinfeld, R., Bull, L., Wang, H., Pieprzyk, J.: Universal designated-verifier signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 523–542. Springer (2003)

[23] Susilo, W., Mu, Y.: On the security of nominative signatures. In: Australasian Conference on Information Security and Privacy. pp. 329–335. Springer (2005)

[24] Unruh, D.: Quantum proofs of knowledge. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 135–152. Springer (2012)

[25] Wang, G., Bao, F.: Security remarks on a convertible nominative signature scheme. In: IFIP International Information Security Conference. pp. 265–275. Springer (2007)

[26] Yuval, G.: How to swindle rabin. Cryptologia 3(3), 187–191 (1979)

[27] Zhao, W., Ye, D.: Pairing-based nominative signatures with selective and universal convertibility. In: International Conference on Information Security and Cryptology. pp. 60–74. Springer (2009)