SCHOOL OF
COMPUTING

Title: Specification of the Graph Signature Cryptographic Library and the PRISMACLOUD Topology Certification
Version 0.9.2

Names: Thomas Gross and Ioannis Sfyrakis

TECHNICAL REPORT SERIES

No. CS-TR- 1523 July 2018

TECHNICAL REPORT SERIES

No: CS-TR- 1523 Date: July 2018

Title: Specification of the Graph Signature Cryptographic Library and the PRISMACLOUD Topology Certification
Version 0.9.2

Authors: Thomas Gross, Ioannis Sfyrakis

Abstract:

Topology certification aims at offering an approach for integrity and privacy for inter-connected systems, such as clouds, by certifying and proving properties of infrastructure topologies without disclosing the confidential information, such as the blueprint of the system. Cryptographically, this is enabled by a signature scheme and corresponding honest-verifier zero-knowledge proofs of knowledge on graphs.

In this document, we describe the architecture and design of a topology certification tool along with a detailed systematic specification of the underlying graph signature cryptographic library. The graph signatures and topology certification could be thought of as an attribute-based credential system on abstract graph data structures and are designed to be compatible with existing attribute-based credentials on user attributes.

This document specifies core components of the Topology Certification (called TOPOCERT) of the EU Horizon 2020 project PRISMACLOUD (GA no644962).

Printed and published by Newcastle University,
School of Computing, Urban Sciences Building,
Newcastle upon Tyne, NE4 5TG, England.

Bibliographical Details

NEWCASTLE UNIVERSITY
School of Computing. Technical Report Series. CS-TR- 1523

Title and Authors : Specification of the Graph Signature Cryptographic Library and the PRISMACLOUD Topology Certification
Version 0.9.2

Thomas Gross, Ioannis Sfyrakis

Abstract:

Topology certification aims at offering an approach for integrity and privacy for inter-connected systems, such as clouds, by certifying and proving properties of infrastructure topologies without disclosing the confidential information, such as the blueprint of the system. Cryptographically, this is enabled by a signature scheme and corresponding honest-verifier zero-knowledge proofs of knowledge on graphs.
In this document, we describe the architecture and design of a topology certification tool along with a detailed systematic specification of the underlying graph signature cryptographic library. The graph signatures and topology certification could be thought of as an attribute-based credential system on abstract graph data structures and are designed to be compatible with existing attribute-based credentials on user attributes.
This document specifies core components of the Topology Certification (called TOPOCERT) of the EU Horizon 2020 project PRISMACLOUD (GA no644962).

About the authors

Current Research: Cyber Security, Privacy and Evidence-based Methods for Security I'm a **Tenured Reader** in **System Security** (Associate Professor) at the **Newcastle University.** I'm the **Director** of the **Centre for Cybercrime and Computer Security (CCCS),** a UK **Academic Centre of Excellence in Cyber Security Research (ACE-CSR).** I'm a member of the Secure and Resilient Systems group and the Centre for Software Reliability (CSR).

Ioannis Sfyrakis is a Research Associate in the Secure & Resilient Systems group, School of Computing at Newcastle University. He is currently involved in the EU/PrismaCloud research project for the implementation of the graph signatures scheme. He submitted his PhD thesis in September 2017. His doctoral research is about protecting lightweight virtual machines in the cloud using access-control and trusted execution environments. His work has been published in different conferences and workshops including IEEE Cloud Engineering. His research interests are virtualization security, access control, trusted execution, unikernels, smart contracts and implementing cryptographic schemes.

Suggested keywords

graph signatures, cryptographic library

Specification of the Graph Signature Cryptographic Library and the PRISMACLOUD Topology Certification

Version 0.9.2

Thomas Groß* and Ioannis Sfyrakis

July 9, 2018

Abstract

Topology certification aims at offering an approach for integrity and privacy for inter-connected systems, such as clouds, by certifying and proving properties of infrastructure topologies without disclosing the confidential information, such as the blueprint of the system. Cryptographically, this is enabled by a signature scheme and corresponding honest-verifier zero-knowledge proofs of knowledge on graphs. In this document, we describe the architecture and design of a topology certification tool along with a detailed systematic specification of the underlying graph signature cryptographic library. The graph signatures and topology certification could be thought of as an attribute-based credential system on abstract graph data structures and are designed to be compatible with existing attribute-based credentials on user attributes.

This document specifies core components of the Topology Certification (called TOPOCERT) of the EU Horizon 2020 project PRISMACLOUD (GA n°644962).

*Contact e-mail: thomas.gross@newcastle.ac.uk

Contents

1 Preliminaries

1.1 Assumptions

Special RSA Modulus: A special RSA modulus has the form $N = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, the corresponding group is called *Special RSA group*.

Strong RSA Assumption: Given an RSA modulus N and a random element $g \in Z_N^*$, it is hard to compute $h \in Z_N^*$ and integer $e > 1$ such that $h^e \equiv \ mod\ N$. The modulus N is of special form pq, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Quadratic Residues: The set QR_N is the set of Quadratic Residues of a special RSA group with modulus N.

2 Cryptography Utilities

In this section, we define utilities for computations in the underlying group structure, especially QR_N. Algorithms presented here are largely adapted from Victor Shoup's excellent book *A Computational Introduction to Number Theory and Algebra* [Sho09].

2.1 Special RSA modulus

Algorithm 1: generateSpecialRSAModulus(): Generate Special RSA Modulus N.

Input: candidate integer a, prime factors of positive, odd integer N: $q_1, \ldots, q_r$.
Output: N, p, q, p', q'

1 $(p, p') \leftarrow generateRandomSafePrime()$
2 $(q, q') \leftarrow generateRandomSafePrime()$
3 $N \leftarrow pq$
4 **return** (N, p, q, p', q')

2.2 Generate Random Safe Prime

The algorithm for generating safe primes is adapted from [MVOV96, Section 4.6.1].

A fast algorithm for generating primes is presented in [CS99].

Algorithm 2: generateRandomSafePrime(): Generate random safe prime.

Input: required k bit-length of the prime.
Output: safe prime p, Sophie Germain prime p'

do
  Select a random (k-1)-bit prime p'
  $p \leftarrow 2p' + 1$
while not *isPrime(p)*
return (p, p')

Algorithm 3: generateRandomSafePrimeFast(): Generate random safe prime (fast algorithm).

Input: required k bit-length of the prime.
Output: safe prime e

Select a random (k+1)-bit prime P
do
  $R \leftarrow computeRandomNumber((2^k - 1)/2P, (2^{k+1} - 1/2P))$
  $e \leftarrow 2PR + 1$
while not *isPrime(e)*
return e

2.3 Commitment group

Algorithm 4: generateNumberSequence(): Compute number sequence [Sho09, Section 9.5].

Input: integer number $m \geq 2$
Output: number sequence $(n_1, \ldots, n_k)$

$n_0 \leftarrow m$
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  $n_k \xleftarrow{R} \{1, \ldots, n_{k-1}\}$
until $n_k = 1$
return $(n_1, \ldots, n_k)$

Algorithm 5: generateRandomNumberWithFactors(): Compute random number in factored form [Sho09, Section 9.6].

Input: integer number $m \geq 2$
Output: random number prime factorization $(p_1, \ldots, p_k)$

while *true* **do**
  $(n_1, \ldots, n_k) \leftarrow$ generateNumberSequence(m)
  let $(p_1, \ldots, p_r)$ be the subsequence of primes in $(n_1, \ldots, n_k)$
  $y \leftarrow \prod_{i=1}^{r} p_i$
  if $y \leq m$ **then**
    $x \xleftarrow{R} \{1, \ldots, m\}$
    if $x \leq y$ **then**
      return $(p_1, \ldots, p_r)$

Algorithm 6: generateRandomPrimeWithFactors(): Compute random prime number in factored form [Sho09, Section 11.1].

Input: integer number $m \geq 2$
Output: prime number p factorization $(p_1, \ldots, p_k)$

do
  $(p_1, \ldots, p_k) \leftarrow generateRandomNumberWithFactors(m)$
  $p \leftarrow \prod_{i=1}^{k} p_i + 1$
while not *isPrime*(p)
return $(p_1, \ldots, p_k)$

Algorithm 7: createZPSGenerator(): Compute generator for $\mathbb{Z}_p^*$ [Sho09, Section 11.1].

Input: prime factorization of the order of an odd prime p $(p - 1 = \prod_{i=1}^{r} q_i^{e_i})$
Output: generator g for $\mathbb{Z}_p^*$

for $i \leftarrow 1$ *to* r **do**
  repeat
    choose $\alpha \in \mathbb{Z}_p^*$ at uniformally random
    compute $\beta \leftarrow \alpha^{(p-1)/q_i}$
  until $\beta \neq 1$
  $\gamma_i \leftarrow \alpha^{(p-1)/q_i^{e_i}}$
$\gamma \leftarrow \prod_{i=1}^{r} \gamma_i$
return γ

2.4 Chinese Remainder Theorem

Algorithm 8: Compute the Chinese Remainder Theorem algorithm (CRT) [KL14, Section 8.1.5].

Input: integers x_p, x_q, primes p, q with $gcd(p, q) = 1$.
Output: the integer x ($0 \leq x < p \cdot q,\ x \equiv x_p \pmod{p},\ x \equiv x_q \pmod{q}$)

Compute X, Y using Extended Euclidean algorithm such that $Xp + Yq = 1$
$1_p \leftarrow [Yq \bmod N]$
$1_q \leftarrow [Xp \bmod N]$
$x \leftarrow [(x_p \cdot 1_p + x_q \cdot 1_q) \bmod N]$
return x

When the factorization of N is known, in order to map x mod N to the mod p and mod q representation, the element x relates to $([x \bmod p], [x \bmod q])$

2.5 Extended Euclidean algorithm

Algorithm 9: Compute the Extended Euclidean algorithm (EEA) [Sho09, Section 4.2].

Input: integer a and odd integer b, $a >= b >= 0$
Output: integers d, s, t such that $d = gcd(a, b)$ and $as + bt = d$.

$r \leftarrow a$
$r' \leftarrow b$
$s \leftarrow 1$
$s' \leftarrow 0$
$t \leftarrow 0$
$t' \leftarrow 1$
while $r'! = 0$ **do**
  $q \leftarrow \lfloor r/r' \rfloor$
  $r'' \leftarrow r \bmod r'$
  $(r, s, t, r', s', t') \leftarrow (r', s', t', r'', s - s'q, t - t'q)$
d ← r
return (d, s, t)

2.6 Quadratic Residues Under Composite Modules QR_N

2.6.1 The Jacobi Symbol $(a|N)$

We will use the Jacobi symbol to establish whether a group element is part of QR_N. We adapt the definition from Shoup [Sho09, Section 12.2]:

Definition 2.1 (Jacobi Symbol)**.** *Let a, N be integers, where N is positive and odd, so that $N = q_1, \ldots, q_k$, where the q_i's are odd primes, not necessarily distinct. Then the* Jacobi symbol $(a|N)$ *is defined as $(a|N) := (a|q_1) \cdots (a|q_k)$, where $(a|q_i)$ is the Legendre symbol (cf. [Sho09, Section 12.1]).*

When it comes to computing the Jacobi symbol, this can be done using Euler's criterion computing:

$$a^{(q_i-1)/2} \pmod{q_i}$$

for each prime factor q_i of N. However, this approach has an asymptotic complexity of $O(r \cdot \mathsf{len}(q_i)^3)$ for a composite of r odd primes q_i: $N = q_1 \cdots q_r$.

Shoup [Sho09, Section 12.3] specified an efficient algorithm similar to the Euclidian Algorithm with asymptotic complexity $O(\mathsf{len}(a)\mathsf{len}(N))$. We will use said Algorithm 10 to compute the Legendre symbol $(a|q_i)$.

Algorithm 10: Compute the Jacobi symbol $(A|N)$ [Sho09, Section 12.3].

Input: candidate integer a, positive odd integer N.
Output: Jacobi symbol $(a|N)$.
Invariant: N is odd and positive.
Dependencies: splitPowerRemainder()

```
1  σ ← 1
2  repeat
3      // Loop invariant:  N is odd and positive.
4
5      a ← a mod N
6      if a = 0 then
7          if N = 1 then return σ else return 0
8      compute a', h such that a = 2^h a' and a' is odd
9      if h ≢ 0 mod 2 and N ≢ ±1 mod 8 then σ ← −σ
10     if a' ≢ 1 mod 4 and N ≢ 1 mod 4 then σ ← −σ
11     (a, N) ← (N, a')
12 forever
```

2.6.2 Testing Membership of QR_N

It is intractable to determine the membership in QR_N without knowledge of the factorization of N.

Given the factorization of $N = q_1 \cdots q_r$, we can determine whether $a \in \mathbb{Z}_N^*$ is a quadratic residue in QR_N by checking that

$$(a|q_i) = 1 \text{ for all } q_i \in \{q_1, \ldots, q_r\}.$$

Consequently, for the setup of the graph signature library with a special RSA modulus of two distinct primes $N = pq$, we require

$$(a|p) = 1 \wedge (a|q) = 1.$$

2.6.3 Generating a Generator of QR_N

We adapting the following definition from Shoup [Sho09, Section 2.7].

Algorithm 11: elementOfQR(): Determines if integer a is an element of QR_N.

Input: candidate integer a, prime factors of positive, odd integer N: $q_1, \ldots, q_r$.
Output: true if $a \in \mathrm{QR}_N$, false if $a \notin \mathrm{QR}_N$.
Dependencies: jacobiSymbol()

```
1 o ← true
2 for all q_i:
3       if (a|q_i) ≠ 1 then o ← false
4 end
5 return o
```

Definition 2.2 (Primitive Root)**.** *For a given positive integer N, we say that $a \in \mathbb{Z}$ with* $\gcd(a, N) = 1$ *is a* primitive root *modulo N, if the multiplicative order of a modulo N is equal to $\phi(N)$.*

Generating an element of QR_N, in general, is simply achieved by squaring uniformly-chosen random element of $\mathbb{Z}_N^*$. An integer a is a group element of $\mathbb{Z}_N^*$ if and only if $\gcd(N, a) = 1$.

We need to ensure that the created element is not a generator of the sub-group of size 2. That is, the resulting quadratic residue must not equal 1.

Algorithm 12: createElementOfZNS(): Generate S' number.

Input: Special RSA modulus N.
Output: random number S' of QR_N.
Dependencies: isElementOfZNS()

```
1 do
2 |   Choose at random S' ∈_R {2, N − 1}
3 while not isElementOfZNS(S')          /* check gcd(S', N) = 1 */
4 return S'
```

Algorithm 13: isElementOfZNS(): Check if number is member of $\mathbb{Z}_N^*$.

Input: number a, modulus N.
Output: boolean: true or false.
Dependencies: isElementOfZNS()

```
1 if gcd(N, a) = 1 then
2 |   return true
3 else
4 |   return false
```

2.6.4 Number Representation.

splitPowerRemainder() presented in Algorithm 17 computes the greatest power of base 2

Algorithm 14: verifySGeneratorOfQRN(): evaluate generator S properties.

Input: generator S, p', q'.
Output: boolean: true or false

```
1 if S ≠ 1 (mod N) then
2 |   if S^p' ≠ 1 (mod N) ∧ S^q' ≠ 1 (mod N) then
3 |   |   return true
4 else
5 |   return false
```

Algorithm 15: verifySGeneratorOfQRN(): evaluate generator S properties (alternative implementation) [Cam17]. The test is based on $S \equiv 1 \pmod{N}$ iff $N|(S-1)$ and $S = 1 \pmod{p}$ and $S = 1 \pmod{q}$. Testing that the $\mathsf{gcd}(S-1, N) \not> 1$ rules out sub-groups with p' or q' members.

Input: generator S, modulus N.
Output: boolean: true or false

```
1 if gcd(S − 1, N) = 1 then
2 |   return true
3 else
4 |   return false
```

Algorithm 16: createQRNGenerator(): Generate generator of QR_N.

Input: Special RSA modulus N, p', q'.
Output: generator S of QR_N.
Dependencies: createElementOfZNS(), verifySGenerator()

```
1 do
2 |   S' ← createElementOfZNS(N)
3 |   S ← S'^2 (mod N)
  |   /* all p'q' elements of QR_N apart from p' + q' elements are
  |      generators                                                  */
4 while not verifySGeneratorOfQRN(S)   /* check properties of generator S
  */
5 return S
```

contained in an odd integer a and its remainder a'. We note that in Java BigInteger the most significant bit of a positive integer can be computed with getBitlength().

Algorithm 17: splitPowerRemainder(): Compute the $2^h a'$ representation of integer a.

Input: odd integer a.
Output: Integers h and a' such that $a = 2^h a'$.
Post-conditions: $a = 2^h a'$ and a' is odd

1 $h \leftarrow \mathsf{mostSignificantBit}(a)$
2 $a' \leftarrow a - 2^h$
3 return (h, a')

2.7 Camenisch-Lysyanskaya Signatures

Algorithm 18: generateCLSignature(): Simplified generation of a Camenisch-Lysyanskaya signature

Input: message m
Output: signature σ

1 $e \leftarrow \mathsf{createRandomPrime}(l_\rho, gs\_params)$
2 $v \leftarrow \mathsf{createRandomNumber}(l_v, gs\_params)$
3 $A = \left(\frac{Z}{R_0^m S^v}\right)^{1/e} \mod N$
4 $\sigma \leftarrow (e, A, v)$
5 **return** σ

3 Graph Signature Library

The graph signature library implements the corresponding signature scheme (GRS) specified by Groß [Gro15].

The library realizes the interactions between a signer and a recipient, meant to create a signature on a hidden committed graph, and the interactions between a prover and a verifier, meant to prove properties of a graph signature in zero-knowledge.

Graph signatures can be formed by combining a committed (hidden) sub-graph from the recipient and a issuer-known sub-graph from the signer. For the sake of the PRISMACLOUD project, it is sufficient to realize issuer-known graphs, as the graphs will be known by the signer (auditor).

3.1 Signer S

The signer is responsible to generate an appropriate key setup, to certify an encoding scheme, and to sign graphs. In the HiddenSign() protocol the signer accepts a graph

commitment from the recipient, adds an issuer-known sub-graph and completes the signature with his secret key sk_{S}. The signer outputs a partial graph signature, subsequently completed by the recipient.

3.2 Recipient R

The recipient initializes the HiddenSign() protocol by creating a graph commitment and retaining randomness R, possibly only containing his master secret key, but no sub-graph. In this case, it is assumed that the signer knows the graph to be signed. Once the signer sends his partial signature, the recipient completes the signature with his randomness R.

3.3 Prover P

The prover role computes zero-knowledge proofs of knowledge with a policy predicate $\mathfrak{P}$ on graph signatures. These proofs can either be interactive or non-interactive.

3.4 Verifier V

The verifier role interacts with a prover to verify a policy predicate $\mathfrak{P}$. The verifier initializes the interaction, sending the policy predicate $\mathfrak{P}$ as well as a nonce that binds the session context.

3.5 Proof Context

The different prover and verifier algorithms co-create, amend and draw upon a joint proof context. The proof context is specific for a session of a zero-knowledge proof. It contains the entire proof state, that is,

- Integer and graph commitments,
- witness commitments,
- challenge,
- responses.

The prover's proof context contains additional secrets:

- The randomness of integer and graph commitments,
- the randomness corresponding to the secrets of the ZKPoK, and
- the secrets themselves (especially the actual graph and its encoding).

3.6 Abstract Description

Parameters. We offer the description of the parameters used for the graph signature scheme in Table 1. We use the same notation as the Identity Mixer credential system, the standard realization of the Camenisch-Lysyanskaya signature scheme [IBM13].

Core Interface. The graph signature library draws upon an interface with multiple operations. We first specify the abstract interface itself in Definition 3.1 and then discuss the inputs and outputs subsequently.

Definition 3.1 (Graph Signature Scheme)**.** *The graph signature scheme consists of the following algorithms:*

$((pk_{\mathsf{S}}, sk_{\mathsf{S}}), \sigma_{\mathsf{S,kg}}) \leftarrow$ **Keygen**$(1^{\lambda}, gs\_params)$ *A probabilistic polynomial-time algorithm which computes the key setup of the graph signature scheme and corresponding commitment scheme.*

$((pk_{\mathsf{S,E}}, sk_{\mathsf{S,E}}), \sigma_{\mathsf{S,es}}) \leftarrow$ **GraphEncodingSetup**$((pk_{\mathsf{S}}, sk_{\mathsf{S}}), \sigma_{\mathsf{S,kg}}, enc\_params)$ *A probabilistic polynomial-time algorithm which computes the setup of the graph encoding, especially, a reserved certified set of bases which are meant to hold the vertex and edge messages.*

$C \leftarrow$ **Commit**$(\mathcal{G}; R)$ *A probabilistic polynomial-time algorithm computing an Integer commitment on a graph.*

$(\sigma; \epsilon) \leftarrow$ **HiddenSign**$(pk_{\mathsf{S,E}}, C, V_{\mathsf{R}}, V_{\mathsf{S}})$ *An interactive probabilistic polynomial-time algorithm between a recipient and a signer which signs a committed graph. We note that both parties can have common inputs, namely a commitment* $C = \mathsf{Commit}(\mathcal{G}_{\mathsf{R}}; R)$ *encoding the recipient's graph and disclosed connections points* $V_{\mathsf{R}}, V_{\mathsf{S}}$*, and the Signer's extended public key* $pk_{\mathsf{S,E}}$*. Hence, the signer and the recipient can contribute sub-graphs to be combined.* Private inputs: *Recipient* R*:* $\mathcal{G}_{\mathsf{R}}$*, commitment randomness* R*; Signer* S*:* $sk_{\mathsf{S,E}}$*,* $\mathcal{G}_{\mathsf{S}}$*.*

0 or 1 $\leftarrow$ **Verify**$(pk_{\mathsf{S}}, C, R', \sigma)$ *A verification algorithm on graph commitment* C *and signature* σ*.*

Algorithm Input/Output Specification. We define the inputs and outputs for the abstract interface as follows.

Definition 3.2 $(((pk_{\mathsf{S}}, sk_{\mathsf{S}}), \sigma_{\mathsf{S,kg}}) \leftarrow \mathsf{Keygen}(1^{\lambda}, gs\_params))$**.** *A probabilistic polynomial-time algorithm which computes the key setup of the graph signature scheme and corresponding commitment scheme.*

Inputs:

- general security parameter (1^{λ})
- key generation parameters of the graph signature scheme (*gs_params*) described in Table 1a.

Outputs:

- secret key (sk_{S}) :
  - factorization of a special RSA group with modulus bit length ℓ_n
  - group setup of the special RSA group
  - group setup of the commitment group Γ
  - foundational generator S for the Quadratic Residues under the given Special RSA modulus QR_N.
  - dedicated base for the Recipient's master key R_0.
- public key pk_{S}:
  - group setup of the special RSA group
  - group setup of the commitment group Γ
- digitally sign the given public outputs and make the signature $\sigma_{\mathsf{S,kg}}$ public
- the parameters specified in *gs_params* are stored for this instantiation of the Signer S.

Group Setup:

Algorithm 19: commitmentGroupSetup(): Group setup for commitment group

Input: size of the prime order subgroup of Γ (l_ρ), size of the commitment group modulus (l_Γ), *gs_params*

Output: order of the subgroup of the commitment group ρ, commitment group modulus Γ, generators g and h

1 $\rho \leftarrow$ generateRandomPrime(ℓ_ρ, gs_params)
2 $\Gamma \leftarrow$ generateGroupModulus(ρ, gs_params)
3 $g \leftarrow$ createGenerator$(\rho, \Gamma, gs_params)$
4 $h \leftarrow g^r$
5 return (ρ, Γ, g, h)

Definition 3.3 $(((pk_{\mathsf{S,E}}, sk_{\mathsf{S,E}}), \sigma_{\mathsf{S,es}}) \leftarrow \mathsf{GraphEncodingSetup}((pk_{\mathsf{S}}, sk_{\mathsf{S}}), \sigma_{\mathsf{S,kg}}, enc\_params))$**.** *A probabilistic polynomial-time algorithm which computes the setup of the graph encoding, especially, a reserved certified set of bases which are meant to hold the vertex and edge messages.*

Inputs:

- Signer S's secret key (sk_{S})
- Signer S's public key (pk_{S})
- encoding setup parameters (*enc_params*)

Algorithm 20: KeyGen(): Main key generation algorithm for the graph signature scheme and commitment scheme

Input: size of RSA modulus (ℓ_n), gs_params
Pre-conditions: ℓ_n must be at least 2048.
Output: public key pk, secret key sk, signature Σ

$(N, p, p', q, q') \leftarrow \mathsf{generateSpecialRSAModulus}(\ell_n, l_{pt})$
$S \leftarrow \mathsf{createQRGenerator}(N)$
$interval \leftarrow [2 \dots p'q' - 1]$
$x_Z \leftarrow \mathsf{createRandomNumber}(interval, gs\_params)$ /* $x_Z \in_R [2, \dots, p'q' - 1]$ */
$Z \leftarrow S^{x_Z} \ mod \ N$
$x_{R_0} \leftarrow \mathsf{createRandomNumber}(interval, gs\_params)$ /* $x_{R_0} \in_R [2, \dots, p'q' - 1]$ */
$R_0 \leftarrow S^{x_{R_0}} \ mod \ N$
$(\rho, \Gamma, g, h) \leftarrow \mathsf{commitmentGroupSetup}(\ell_\rho, l_\Gamma, gs\_params)$
$sk \leftarrow (p', q', x_{R_0}, x_Z)$
$pk \leftarrow (N, R_0, S, Z)$
return (sk,pk)

Outputs:

- Public Output: bases reserved to hold vertex and edge encodings, $R_i | i \in \{1, \dots, \ell_\mathcal{V}\}$ for vertices and $R_j | j \in \{1, \dots, \ell_\mathcal{E}\}$ for edges.
- Sign the generators, proving knowledge of their representation and binding them to public key pk_S.
- Signer's *extended public key* $pk_{\mathsf{S,E}}$ certified combination of original public key pk_S and the vertex and edge encoding bases R_i, R_j
- Signer's *extended secret key* $sk_{\mathsf{S,E}}$ combination of original secret key sk_S and the discrete logarithms $\mathsf{log}_S(R_k)$
- Signature $\sigma_{\mathsf{S,es}}$
- Private Output: discrete logarithms of all produced bases with respect to generator S, $\mathsf{log}_S(R_k)$.

Definition 3.4 ($C \leftarrow \mathsf{Commit}(\mathcal{G}; R)$)**.** *A probabilistic polynomial-time algorithm computing an Integer commitment on a graph.*

Inputs:

- graph $\mathcal{G}$
- randomness R

Computations: It commits to the graph in an appropriate encoding, that is, holding vertex and edge representations in different bases. As specified in the graph signature definition [Gro15], the algorithm will establish a commitment as follows:

$$C = \underbrace{\cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k} \cdots}_{\forall \text{ edges } (i,j)} S^r \bmod N,$$

where e_i and e_j are vertex representatives. The label representatives e_k are obtained with the vertex mappings $f_{\mathcal{V}}(i)$ and edge mappings $f_{\mathcal{E}}(i,j)$.

Outputs:

- commitment C
- Committer retains the randomness R for future commitment opening or proofs of representation

Definition 3.5 $((\epsilon;\sigma) \leftarrow \mathsf{HiddenSign}(pk_{\mathsf{S,E}}, C, V_{\mathsf{R}}, V_{\mathsf{S}}))$**.** *An interactive probabilistic polynomial-time algorithm between a recipient and a signer which signs a committed graph. We note that both parties can have common inputs, namely a commitment* $C = \mathsf{Commit}(\mathcal{G}_{\mathsf{R}}; R)$ *encoding the recipient's graph and disclosed connections points* $V_{\mathsf{R}}, V_{\mathsf{S}}$*, and the Signer's extended public key* $pk_{\mathsf{S,E}}$*. Hence, the signer and the recipient can contribute sub-graphs to be combined.* Private inputs: *Recipient* R*:* $\mathcal{G}_{\mathsf{R}}$*, commitment randomness* R*; Signer* S*:* $sk_{\mathsf{S,E}}$*,* $\mathcal{G}_{\mathsf{S}}$*.*

The abstract interface specification for the interactive algorithm decomposes into two interfaces for Signer S and Recipient R.

$$\mathsf{Signer.HiddenSign}(pk_{\mathsf{S,E}}, C, V_{\mathsf{R}}, V_{\mathsf{S}}; sk_{\mathsf{S,E}}, \mathcal{G}_{\mathsf{S}}), \text{and}$$

$$\mathsf{Recipient.HiddenSign}(pk_{\mathsf{S,E}}, C, V_{\mathsf{R}}, V_{\mathsf{S}}; \mathcal{G}_{\mathsf{R}}, R).$$

Let us first discuss public and private inputs. While the Integer commitment C is publicly known, it is usually computed by the Recipient R for the the given HiddenSign() operation with the corresponding randomness R with Commit(). The Recipient will be required to offer a proof of representation of the commitment as part of the interactive protocol.

Note that the key-pair inputs $(pk_{\mathsf{S,E}}, sk_{\mathsf{S,E}})$ refer to extended keys, that is, public and private keys that contain the information on encoding bases of GraphEncodingSetup().

While the HiddenSign() algorithm allows for either both private input graphs $\mathcal{G}_{\mathsf{R}}$ and $\mathcal{G}_{\mathsf{S}}$ to be present or absent, the standard case realized in TOPOCERT is that we have a signer-known graph $\mathcal{G}_{\mathsf{S}}$, but no hidden/committed graph of the Recipient R.

In most cases the connection points V_{R} and V_{S} will be equal, but that is not necessary.[1]

As output on the Recipient side, R obtains a graph signature $\sigma_{\mathsf{S},\mathcal{G}}$ (or short σ) on the combined graph $\mathcal{G} = \mathcal{G}_{\mathsf{R}} \cup \mathcal{G}_{\mathsf{S}}$ valid with respect to $pk_{\mathsf{S,E}}$.

The Signer S does not produce an output.

[1] The first graph signature [Gro15] proposal referred to the connection points as $\mathcal{V}_{\mathsf{R}}$, $\mathcal{V}_{\mathsf{S}}$, which would mean the set of all vertices and not a subset.

Definition 3.6 (0 or $1 \leftarrow \mathsf{Verify}(pk_{\mathsf{S,E}}, C, R', \sigma)$)**.** *A verification algorithm on graph commitment C and signature σ.*

The algorithm $\mathsf{Verify}()$ takes as inputs the Signer's extended public key $pk_{\mathsf{S,E}}$, a signed graph commitment C and its randomness R' and the graph signature σ. The algorithm outputs either 0 or 1, signifying that σ is either invalid or valid as graph signature on C.

We note that usually graph signatures, such as σ are used in proof of possessions and further zero-knowledge proofs of knowledge to show certain properties. In such cases, we can use $\mathsf{Verify}()$ to signify a proof of representation that the secrets encoded in commitment C are equal to the secret messages of the graph signature σ.

Then we have a zero-knowledge proof of knowledge defined as follows:

$$
\begin{aligned}
&PK\{(e_i, e_j, e_k, e, v, r) : \\
&\quad Z \equiv \pm \underbrace{\cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k} \cdots}_{\forall \text{ edges } (i,j)} A^e S^v \pmod N \wedge \\
&\quad C \equiv \pm \underbrace{\cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k} \cdots}_{\forall \text{ edges } (i,j)} S^r \pmod N \\
&\}.
\end{aligned}
$$

Here the first equation proves the representation of the graph signature σ and the second equation proves the representation of the commitment C, yielding equality over the secrets e_i, e_j and e_k.

We note that the proofs of knowledge on graph properties between prover and verifier are specified as standard Σ-proofs.

3.7 Preliminaries

Notation. We specify parameters with the same conventions as the Specification of the Identity Mixer Cryptographic Library [IBM13]. For lengths of bitstrings and numbers specified by a positive integer ℓ, $\{0,1\}^\ell$ denotes the set of $\{0, 1, \ldots, 2^\ell - 1\}$, while $\pm\{0,1\}^\ell$ denotes the set of integers $\{-2^\ell + 1, \ldots, 2^\ell - 1\}$.

Proof Specification. We specify zero-knowledge proofs of knowledge (and corresponding signature proofs of knowledge) in the Camenisch-Stadler notation [CS97]. Proofs specified as such can be directly compiled into honest-verifier Σ-proofs as Schnorr protocols.

In this specification, we will present the high-level PK/SPK specification as well as its implementation, especially to make clear how commitments and differential randomnesses will be computed to be subsequently proven. While the original Camenisch-Stadler notation used the Greek alphabet to denote secret values, we use the Latin alphabet for secrets in the implementation specification.

Proof Realization. In the implementation specification of Schnorr proofs, we use different diacritical marks above the variable symbol to denote types of variables (chosen randomness as well as computed values):

- the tilde (˜) diacritic denotes witnesses (chosen randomness or witness for a particular equation), and
- the hat (ˆ) diacritic denotes responses (solutions to proofs for particular secrets).

In the case that the secret symbol already carries a diacritical mark, that original diacritical mark is maintained and the diacritical mark for the variable type added on top.

Secrets and public values receive diacritical marks to denote particular purposes in a proof system:

- the breve (˘) diacritic refers to commitments and randomness for element representations stripped of labels.
- the dot (˙) diacritic refers to commitments and randomness for secondary element representations stripped of labels.

Commitments without diacritical mark will usually refer to the entire representation.

Java Variable Naming. In the Java implementation of the graph signature library, we have the notation that the attributes/variables are named following closely the LaTeX encoding of the specification.

The following words are reserved: 1. `tilde`, 2. `hat`, 3. `breve`, 4. `dot`, 5. `bar`, 6. `prime`.

Here we maintain the order:

1. Type the value (e.g., `tilde`, `hat`),
2. The governing secret (e.g., `m`),
3. Possible modifiers (e.g., `prime`)
4. The index,

Java variables are names according to the Java naming conventions with exceptions:

- The variables are transformed in mixed camel case, that is, starting lower-case word and continuing with capitalizing each following word.
- Variables must retain their original case, e.g., we write $\tilde{m}$ as `tildem`, not `tildeM`
- Indices are prefixed with a single underscore "_"

Here are a few examples:

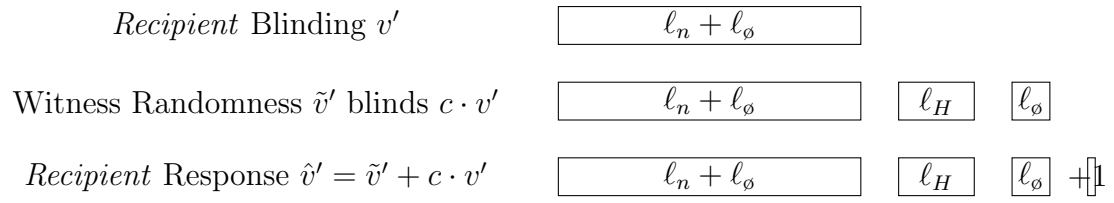
- $\tilde{m}_0$ = `tildem_0`
- r'_i = `rPrime_i`
- $\hat{a}_{\bar{i},\bar{i}}$ = `hata_BariBarj`

Element Ordering. For the computation of the Fiat-Shamir heuristic and messages sent over the wire, we maintain the following order. The most general element comes first, followed by the elements created in subsequent issuing/proof stages.

Specifically, that principle implies the following framework order:

1. *context*,
2. Group modulus N,
3. QR_N generator S,
4. Target value for RSA-based signatures Z,
5. Long-term parameters/bases of the graph signature scheme
   - 5.1 Master secret key base R_0,
   - 5.2 Bases reserved for vertex representation R_i,
   - 5.3 Bases reserved for edge representation R_j.
6. Public values and commitments present for the given proof,

Figure 1: Illustration of the relationship of different exponent length in a Schnorr proof, with the *Recipient* blinding v' as example. The length of the secret to be proven, here v', determines the lengths of the other exponents.



7. Witnesses computed for the given proof $\tilde{T}$ (tilde-values, ordered according to the principles above),
8. Responses computed for the given proof (hat-values, ordered according to the principles above),
9. Proof challenge c.

Parameter Length. In Figure 1, we give an overview of how different exponent lengths relate in Schnorr proofs. We consider the example of the *Recipient* blinding randomness v', which will be proven in an response equation

$$\hat{v}' \leftarrow \tilde{v}' + c \cdot v',$$

using the witness randomness $\tilde{v}'$ as blinding.

Crucially, the length of the witness randomness needs to be $\ell_\emptyset$ longer than the product of challenge c and secret, here ℓ_H and $\ell_n + \ell_\emptyset$ to guarantee statistical zero-knowledge property. In the given example, thereby, the witness randomness has a length of $\ell_n + 2\ell_\emptyset + \ell_H$. Finally, the response will be one bit longer than this due to the sum of two values of equal length.

4 Key Generation

4.1 Group Setup

Compute bases Z, R, R_0, R_i, R_j using key generation algorithm 20. We refer to the randomness used in this algorithm as r_Z, r, r_0, r_i, r_j, all positive integers with at most length ℓ_n. The signer stores these respective random exponents as $\mathsf{dlog}_S(Z)$, $\mathsf{dlog}_S(R_0)$, $\mathsf{dlog}_S(R_i)$, and $\mathsf{dlog}_S(R_j)$ for all bases of the extended public key.

In order to guarantee that the elements of the public key lie in the correct subgroups, we construct a proof of representation on the base generation using a non-interactive proof, which is called a *Signature Proof of Knowledge* (SPK). The *Prover* constructs a proof and publishes the proof to the *Verifier*, who can verify the generated proof.

- The following proof and verification computations prove knowledge of the discrete logarithms $\mathsf{dlog}_S(Z)$, $\mathsf{dlog}_S(R_0)$, $\mathsf{dlog}_S(R_i)$, and $\mathsf{dlog}_S(R_j)$. They iterate over all bases reserved for vertex and edge representation. It
  $\forall i \in \{1, \ldots, \ell_\mathcal{V}\}$; $\forall j \in \{1, \ldots, \ell_\mathcal{E}\}$.

- Compute the following Signature Proof of Knowledge:

$$
\begin{aligned}
SPK\{&\forall i\{1,\ldots,\ell_{\mathcal{V}}\}, \forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : (r_Z, r, r_0, r_i, r_j) : \\
&Z \equiv \pm S^{r_Z} \pmod{N} \wedge \\
&R \equiv \pm S^{r} \pmod{N} \wedge \\
&R_0 \equiv \pm S^{r_0} \pmod{N} \wedge \\
&R_i \equiv \pm S^{r_i} \pmod{N} \ \wedge \ R_j \equiv \pm S^{r_j} \pmod{N} \\
\}&
\end{aligned}
$$

  1. Create uniformly-chosen witness randomness:
     $\tilde{r}_Z \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H}$
     $\tilde{r} \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H}$
     $\tilde{r}_0 \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H}$
     $\forall i \in \{1,\ldots,\ell_{\mathcal{V}}\} : \tilde{r}_i \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H}$
     $\forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : \tilde{r}_j \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H}$

  2. Compute witnesses:
     $\tilde{Z} \leftarrow S^{\tilde{r}_Z} \bmod N$
     $\tilde{R} \leftarrow S^{\tilde{r}} \bmod N$
     $\tilde{R}_0 \leftarrow S^{\tilde{r}_0} \bmod N$
     $\forall i \in \{1,\ldots,\ell_{\mathcal{V}}\} : \tilde{R}_i \leftarrow S^{\tilde{r}_i} \bmod N$
     $\forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : \tilde{R}_j \leftarrow S^{\tilde{r}_j} \bmod N$

  3. Compute challenge:
     $c \leftarrow \forall i \in \{1,\ldots,\ell_{\mathcal{V}}\}, \forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : \mathcal{H}(\mathit{context}, N, S, Z, R, R_0, R_i, R_j, \tilde{Z}, \tilde{R}, \tilde{R}_0, \tilde{R}_i, \tilde{R}_j)$

  4. Compute responses:
     $\hat{r}_Z \leftarrow \tilde{r}_Z + c \cdot r_Z$
     $\hat{r} \leftarrow \tilde{r} + c \cdot r$
     $\hat{r}_0 \leftarrow \tilde{r}_0 + c \cdot r_0$
     $\forall i \in \{1,\ldots,\ell_{\mathcal{V}}\} : \hat{r}_i \leftarrow \tilde{r}_i + c \cdot r_i$
     $\forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : \hat{r}_j \leftarrow \tilde{r}_j + c \cdot r_j$

  5. Output proof signature:
     $P \leftarrow \forall i \in \{1,\ldots,\ell_{\mathcal{V}}\}, \forall j \in \{1,\ldots,\ell_{\mathcal{E}}\} : (N, S, Z, R, R_0, R_i, R_j, \hat{r}_Z, \hat{r}, \hat{r}_0, \hat{r}_i, \hat{r}_j, c)$

- The *Verifier* computes verification:

  1. Check lengths:
     $\hat{r}_Z \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
     $\hat{r} \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
     $\hat{r}_0 \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
     $\forall i \in \pm\{1,\ldots,\ell_{\mathcal{V}}\} : \hat{r}_i \in \{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
     $\forall j \in \pm\{1,\ldots,\ell_{\mathcal{E}}\} : \hat{r}_j \in \{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
     if any length check fails **then abort** and **reject**, outputting $\perp$.

2. Compute $\hat{t}$-values:
   $\hat{Z} \leftarrow Z^{-c} S^{\hat{r}_Z} \bmod N$
   $\hat{R} \leftarrow R^{-c} S^{\hat{r}} \bmod N$
   $\hat{R}_0 \leftarrow R_0^{-c} S^{\hat{r}_0} \bmod N$
   $\forall i \in \{1, \ldots, \ell_{\mathcal{V}}\} : \hat{R}_i \leftarrow R_i^{-c} S^{\hat{r}_i} \bmod N$
   $\forall j \in \{1, \ldots, \ell_{\mathcal{E}}\} : \hat{R}_j \leftarrow R_j^{-c} S^{\hat{r}_j} \bmod N$
3. Compute the verification challenge:
   $\hat{c} \leftarrow \forall i \in \{1, \ldots, \ell_{\mathcal{V}}\}, \forall j \in \{1, \ldots, \ell_{\mathcal{E}}\} :$
   $\mathcal{H}(\texttt{context}, N, S, Z, R, R_0, R_i, R_j, \hat{Z}, \hat{R}, \hat{R}_0, \hat{R}_i, \hat{R}_j)$
4. Verify equality of challenge:
   if $c = \hat{c}$ **then**
   accept
   else
   reject, outputting $\perp$.

5 Issuing Specification

We specify the issuing of graph signatures based on the underlying SRSA Camenisch-Lysyanskaya signature scheme. Given the use of the underlying signature scheme, the issuing specification draws upon the original Camenisch-Lysyanskaya SRSA scheme [CL02], the Identity Mixer Cryptographic Library Specification [IBM13], and the issuing specification offered by Groß [Gro14].

The given issuing protocol differs from existing CL-Signature implementations in its accounting for graphs as input of both parties.

The issuing protocol is executed between two parties, a *Signer* opening the protocol with a nonce, and a *Recipient* possibly inputting a committed (hidden) graph and committing to the *Recipient* master secret. The protocol is executed in four rounds, 0 to 3.

5.1 Protocol: Signer Specification

5.1.1 Round 0: Nonce Generation

Inputs

Designated recipient.

1. *Signer* chooses a random nonce $n_1 \in_R \{0,1\}^{\ell_H}$.
2. *Signer* stores the random nonce for future reference.
3. *Signer* $\xrightarrow{n_1}$ *Recipient*

Outputs

Signer nonce n_1

5.1.2 Graph Signature Computation

Inputs

Signer Extended Secret Key: $sk_{\mathsf{S,E}}$
Recipient Commitment: U and corresponding proof P_1
Recipient nonce n_2

1. *Signer* verifies proof signature P_1, iterating over responses representing vertices i and edges (i,j) in $G' = (V', E')$:

   1.1 Check lengths of integer responses:
   $\hat{v}' \in \pm\{0,1\}^{\ell_n+2\ell_\emptyset+\ell_H+1}$,
   $\hat{m}_0 \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$,
   $\forall i \in V' : \hat{m}_i \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$,
   $\forall (i,j) \in E' : \hat{m}_{(i,j)} \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$.

   1.2 **if** any length check fails **then reject** P_1 and **abort** issuing, outputting $\perp$.

   1.3 Compute:

$$\hat{U} \leftarrow U^{-c}(S^{\hat{v}'})R_0^{\hat{m}_0} \underbrace{\cdots R_{\pi(i)}^{\hat{m}_i} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{\hat{m}_{(i,j)}} \cdots}_{\forall \text{ edges } (i,j)} \bmod N$$

   1.4 Verify challenge:
   $\hat{c} \leftarrow \forall i \in V', \forall (i,j) \in E' : \mathcal{H}(\texttt{context}, N, S, Z, R_0, R_{\pi(i)}, R_{\pi(i,j)}, U, \hat{U}, n_1)$

   1.5 **if** $\hat{c} \neq c$ **then reject** P_1 and **abort** issuing, outputting $\perp$.

2. *Signer* generates partial graph signature to be completed by the *Recipient*. These computations iterate over the $i \in V''$ and $(i,j) \in E''$ of the *Signer* graph $G = (V'', E'')$.

   2.1 Choose a random prime:
   $e \in_R [2^{\ell_e-1}, 2^{\ell_e-1} + 2^{\ell'_e-1}]$

   2.2 Choose a random integer $\bar{v} \in_R \pm\{0,1\}^{\ell_v-1}$

   2.3 Compute: $v'' \leftarrow 2^{\ell_v-1} + \bar{v}$

   2.4 Compute:

$$Q \leftarrow \forall i \in V'', \forall (i,j) \in E'' : \frac{Z}{US^{v''} \cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k}} \bmod N$$

   Specifically, the *Signer* uses the knowledge of the discrete logarithms $\mathsf{dlog}_S(Z)$, $\mathsf{dlog}_S(R_0)$, $\mathsf{dlog}_S(R_i)$, and $\mathsf{dlog}_S(R_j)$ to facilitate this computation in the exponent. We break this computation down into vertex and edge exponents:

$$\forall i \in V'' : \bar{e}_i \leftarrow (\mathsf{dlog}_S(R_{\pi(i)}) e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k)$$

$$\forall (i,j) \in E'' : \bar{e}_{(i,j)} \leftarrow (\mathsf{dlog}_S(R_{\pi(i,j)})e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k)$$

Using these exponents, the *Signer* completes the computation:

$$Q \leftarrow \forall i \in V'', \forall (i,j) \in E'' : \left(US^{v''+\ldots+\bar{e}_i+\ldots+\bar{e}_{(i,j)}+\ldots-\mathsf{dlog}_S(Z)}\right)^{-1} \bmod N$$

2.5 Compute A:

i. Knowing the group order $\#\,\mathrm{QR}_N = p'q'$, the issuer computes the multiplicative inverse of e under $p'q'$. As e is co-prime with $p'q'$, e^{-1} exists. For subsequent proofs we refer to e^{-1} as d.

ii. $A \leftarrow Q^{e^{-1} \bmod p'q'} \bmod N$

2.6 *Signer* stores: $Q, v'', \texttt{context}$

3. *Signer* creates a proof of correctness and knowledge of $d = e^{-1}$:

$$SPK\{(d) : A \equiv \pm Q^d \pmod N\}(n_2)$$

3.1 Choose witness randomness:
$\tilde{d} \in_R [2, p'q' - 1]$

3.2 Compute witness:
$\tilde{A} \leftarrow Q^{\tilde{d}} \bmod N$

3.3 Compute challenge:
$c' \leftarrow \mathcal{H}(\texttt{context}, Q, A, \tilde{A}, n_2)$

3.4 Compute response: $\hat{d} \leftarrow \tilde{d} - c' \cdot d \bmod p'q'$

3.5 The proof consists of $P_2 \leftarrow (\hat{d}, c')$

4. *Signer* sends $(A, e, v''), P_2$ and the graph encoding for G'' to the *Recipient*

Outputs

Preliminary graph signature: (A, e, v'') and corresponding proof P_2
Graph encoding for G''

5.2 Protocol: Recipient Specification

5.2.1 Round 1: Commitment to Master Secret Key and Hidden Graph

Inputs

Signer nonce n_1
Signer extended public key: $pk_{\mathsf{S,E}}$
Recipient graph: G' (optional)
Recipient master secret key: *msk*

1. *Recipient* chooses a uniformly-chosen random integer $v' \in_R \pm\{0,1\}^{\ell_n + \ell_\emptyset}$

2. *Recipient* commits to the graph with an appropriate encoding following the Commit() algorithm:

$$U \leftarrow \underbrace{\cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k} \cdots}_{\forall \text{ edges } (i,j)} S^{v'} \bmod N \tag{1}$$

3. *Recipient* computes an non-interactive proof that the commitment is correctly computed (proof of representation) using a Signature Proof of Knowledge (SPK):

$\forall i : i \in V'$, the set of vertices of *Recipient* graph $G' = (V', E')$
$\forall (i,j) : (i,j) \in E'$, the set of edges of *Recipient* graph $G' = (V', E')$

In the following proof, the m_i and $m_{(i,j)}$ contain the full encoding of vertices and edges (incl. labels) of *Recipient* graph G', respectively. The base R_0 is reserved to encode the *Recipient*'s master secret key *msk* as m_0, used to bind graph signatures to the same entity.

$$\begin{aligned} &SPK\{\forall i \in V', \forall (i,j) \in E' : (m_i, m_{(i,j)}, v') : \\ &\quad U \equiv \pm R_0^{m_0} \underbrace{\cdots R_{\pi(i)}^{m_i} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{m_{(i,j)}} \cdots}_{\forall \text{ edges } (i,j)} S^{v'} \pmod N \wedge \\ &\quad v' \in \pm\{0,1\}^{\ell_n + \ell_\emptyset} \ \wedge \ m_0 \in \pm\{0,1\}^{\ell_m} \wedge \\ &\quad \forall i \in V' : m_i \in \pm\{0,1\}^{\ell_m} \ \wedge \ \forall (i,j) \in E' : m_{(i,j)} \in \pm\{0,1\}^{\ell_m} \\ &\}(n_1) \end{aligned}$$

3.1 Create uniformly-chosen witness randomness:
$\tilde{v}' \in_R \pm\{0,1\}^{\ell_n + 2\ell_\emptyset + \ell_H}$
$\tilde{m}_0 \in_R \pm\{0,1\}^{\ell_m + \ell_\emptyset + \ell_H + 1}$
$\forall i \in V' : \tilde{m}_i \in_R \pm\{0,1\}^{\ell_m + \ell_\emptyset + \ell_H + 1}$
$\forall (i,j) \in E' : \tilde{m}_{(i,j)} \in_R \pm\{0,1\}^{\ell_m + \ell_\emptyset + \ell_H + 1}$

3.2 compute witness:
$$\tilde{U} \leftarrow R_0^{\tilde{m}_0} \underbrace{\cdots R_{\pi(i)}^{\tilde{m}_i} \cdots}_{\forall \text{ vertices } i} \underbrace{\cdots R_{\pi(i,j)}^{\tilde{m}_{(i,j)}} \cdots}_{\forall \text{ edges } (i,j)} S^{\tilde{v}'} \bmod N$$

3.3 compute challenge:
$c \leftarrow \mathcal{H}(\texttt{context}, N, S, Z, R_0, R_{\pi(i)}, R_{\pi(i,j)}, U, \tilde{U}, n_1)$

3.4 compute responses:
$\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$
$\hat{m}_0 \leftarrow \tilde{m}_0 + c \cdot m_0$
$\forall i \in V' : \hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
$\forall (i,j) \in E' : \hat{m}_{(i,j)} \leftarrow \tilde{m}_{(i,j)} + c \cdot m_{(i,j)}$

3.5 output proof signature: $P_1 \leftarrow \forall i \in V', \forall (i,j) \in E' : (c, \hat{v}', \hat{m}_i, \hat{m}_{(i,j)})$

4. *Recipient* chooses a random nonce $n_2 \in_R \{0,1\}^{\ell_H}$

5. *Recipient* $\xrightarrow{U,\ P_1,\ n_2}$ *Signer*: commitment U, proof signature P_1, nonce n_2

6. *Recipient* persists the following structures: `context`, randomness v'

Outputs

Recipient graph commitment: U with corresponding proof P_1
Recipient nonce: n_2

5.2.2 Round 3: Signature Completion

Inputs

Preliminary graph signature: (A, e, v'') and corresponding proof P_2
Graph encoding for G''

1. Compute $v \leftarrow v'' + v'$

2. Recipient verifies graph signature (A, e, v), iterating over all $i \in V$ and $(i,j) \in E$ of the combined graph $G = (V, E)$:

   2.1 Check that e is prime and $e \in [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$.

   2.2 **if** this length check fails **then abort** the issuing protocol, outputting $\perp$.

   2.3 Compute:

$$Q \leftarrow \forall i \in V, \forall (i,j) \in E : \frac{Z}{S^v R_0^{m_0} \cdots R_{\pi(i)}^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} \cdots R_{\pi(i,j)}^{e_i e_j \Pi_{k \in f_{\mathcal{E}}(i,j)} e_k}} \bmod N$$

   2.4 Compute

$$\hat{Q} \leftarrow A^e \pmod N$$

   2.5 **if** $\hat{Q} \neq Q \bmod N$, **then abort** the issuing protocol, outputting $\perp$.

3. *Recipient* verifies P_2

   3.1 Compute

$$\hat{A} \leftarrow A^{c' + \hat{d} \cdot e} \bmod N, \text{where}$$
$$\hat{A} \equiv A^{c'} Q^{\hat{d}} \pmod N$$

   3.2 Compute $\hat{c} \leftarrow \mathcal{H}(\texttt{context}, Q, A, \hat{A}, n_2)$

   3.3 **if** $\hat{c} \neq c'$ **then abort** the issuing protocol, outputting $\perp$.

Given:

$d \equiv e{-}1 \pmod{p'q'}; \qquad \tilde{d} \in_R \mathbb{Z}^*_{p'q'}$

$\tilde{A} \equiv Q^{\tilde{d}} \pmod{N}; \qquad c' \in \{0,1\}^{\ell_H}$

$\hat{d} \equiv \tilde{d} - c' \cdot d \pmod{p'q'}$

$$\begin{aligned}
\hat{A} &\equiv A^{c'+\hat{d}\cdot e} \pmod{N} \\
&\equiv A^{c'} A^{(\tilde{d}-c'\cdot d)\cdot e} \pmod{N} \\
&\equiv A^{c'} A^{\tilde{d}\cdot e} A^{-c'\cdot d\cdot e} \pmod{N} \\
&\equiv A^{c'} (A^e)^{\tilde{d}} A^{-c'(e^{-1}\cdot e)} \pmod{N} \\
&\equiv Q^{\tilde{d}} A^{c'} A^{-c'\cdot 1} \pmod{N} \\
&\equiv Q^{\tilde{d}} \stackrel{!}{\equiv} \tilde{A} \pmod{N} \qquad \square
\end{aligned}$$

Figure 2: Correctness proof for the verification of P_2.

4. **if** the steps 2 and 3 are successful **then** store the graph signature $\sigma = (A, e, v)$ along with the corresponding graph encoding.

Outputs

Final graph signature: (A, e, v)
Corresponding graph encoding for G

We include the straight-forward, yet non-standard correctness proof of the verification of P_2 in Figure 2.

6 Overall Proof of Geo-Location Separation

Note that the proof protocol for geo-location separation is exemplary for other proofs, incl. proof of possession and pair-wise difference. We include a monolithic specification of the required proofs to offer an intuition what is computed and ascertain correctness.

The proof will iterate over all vertices $i \in V$ and all edges $(i, j) \in E$ for the entire graph $G = (V, E)$. As such the proof will disclose the maximal size of the graph in the proof of possession.

We note that the proofs have been written separately in the corresponding publications for clarity, asking for a compound execution for referential integrity. Here, we execute the entire proof protocol in one go.

What is proven. On an input by the *Verifier* with a subset of known vertex identifier $\bar{V}$, the *Prover* engages in an SPK to prove that the geo-location labels associated with

those vertices $v_i \in \bar{V}$ are pair-wise different. Notably, this proof requires that all vertices of the graph are labeled with their respective geo-location. The proof does not disclose anything else.

6.1 Geo-Location Separation Proof

- *Verifier* chooses a random nonce $n_3 \in_R \{0,1\}^{\ell_H}$.
- *Verifier* $\xrightarrow{n_3}$ *Prover*
- *Prover* computes commitments on all vertex representations:
  $C_i \leftarrow R^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} S_i^r \bmod N$
- The *Prover* iterates over pair-wise different vertex encodings $\bar{i}, \bar{j} \in \bar{V} \subseteq V$, solving the Extended Euclidian Algorithm $\mathsf{EEA}()$ for arguments for Bézout's Identity:

  $$(d_{\bar{i},\bar{j}}, a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}}) \leftarrow \mathsf{EEA}(m_i, m_j), \text{ such that,}$$

  $$d_{\bar{i},\bar{j}} = a_{\bar{i},\bar{j}} m_i + b_{\bar{i},\bar{j}} m_j.$$

  If m_i and m_j are indeed separate, they must be co-prime and $d_{\bar{i},\bar{j}} = 1$. If and only if this is true, $a_{\bar{i},\bar{j}}$ and $b_{\bar{i},\bar{j}}$ exist to fulfil Bézout's Identity for 1. The *Prover* stores the values $a_{\bar{i},\bar{j}}$ and $b_{\bar{i},\bar{j}}$ for pair-wise different vertices $\bar{i}$, $\bar{j}$ in $\bar{V}$.
- The *Prover* computes and stores the differential randomness for pairs of commitments $C_{\bar{i}}$ and $C_{\bar{j}}$ for pair-wise different vertices $\bar{i}$, $\bar{j}$ in $\bar{V}$:

  $$r_{\bar{i},\bar{j}} \leftarrow -r_{\bar{i}} \cdot a_{\bar{i},\bar{j}} - r_{\bar{j}} \cdot b_{\bar{i},\bar{j}}.$$

- *Prover* establishes a Proof of Geo-Separation over the entire graph $G = (V, E)$, which entails the steps to (a) prove possession of the graph signature σ, (b) prove representation of the commitments C_i on the graph representations, incl. equality to the message exponents, and (c) prove pair-wise difference over the vertices in $\bar{V}$.

  $\forall i \in V, (i,j) \in E, \forall \bar{i}, \bar{j} \in \bar{V} \subseteq V :$

  $$\begin{aligned}
  &SPK\{(m_0, m_i, m_{(i,j)}, e, v, r_i, a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}}, r_{\bar{i},\bar{j}}) : \\
  &\quad Z \equiv \pm R_0^{m_0} R_{\pi(i)}^{m_i} R_{\pi(i,j)}^{m_{(i,j)}} A'^e S^v \pmod N \wedge && (\mathsf{GSPossessionProver}()) \\
  &\quad \forall i \in V : C_i \equiv \pm R^{m_i} S^{r_i} \pmod N \wedge && (\mathsf{CommitmentProver}()) \\
  &\quad \forall \bar{i}, \bar{j} \in \bar{V} : R \equiv \pm C_{\bar{i}}^{a_{\bar{i},\bar{j}}} C_{\bar{j}}^{b_{\bar{i},\bar{j}}} S^{r_{\bar{i},\bar{j}}} \pmod N \wedge && (\mathsf{PairWiseDifferenceProver}()) \\
  &\quad e \in \{0,1\}^{\ell_e} \wedge v \in \pm\{0,1\}^{\ell_v} \wedge m_0 \in \pm\{0,1\}^{\ell_m} \wedge \\
  &\quad \forall i \in V : m_i \in \pm\{0,1\}^{\ell_m} \wedge \\
  &\quad \forall (i,j) \in E : m_{(i,j)} \in \pm\{0,1\}^{\ell_m} \\
  &\}
  \end{aligned}$$

  1. Randomize A

1.1 Choose $r_A \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset}$

1.2 Compute the randomized signature (A', e, v')
$A' \leftarrow AS^{r_A} \pmod N$

1.3 Compute $v' \leftarrow v - e \cdot r_A$

1.4 Compute $e' \leftarrow e - 2^{\ell_e - 1}$

2. Choose witness randomness uniformly at random:
$\tilde{e} \in_R \pm\{0,1\}^{\ell'_e+\ell_\emptyset+\ell_H+1}$
$\tilde{v} \in_R \pm\{0,1\}^{\ell_v+\ell_\emptyset+\ell_H+1}$
$\tilde{m}_0 \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
$\forall i \in V$:
$\tilde{m}_i \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
$\tilde{r}_i \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
$\forall (i,j) \in E$:
$\tilde{m}_{(i,j)} \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
$\forall \bar{i}, \bar{j} \in \bar{V}$:
$\tilde{a}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
$\tilde{b}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
$\tilde{r}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$

3. Compute witnesses:
$\tilde{Z} \leftarrow (A')^{\tilde{e}} R_0^{\tilde{m}_0} R_{\pi(i)}^{\tilde{m}_i} \cdots R_{\pi(i,j)}^{\tilde{m}_{(i,j)}} S^{\tilde{v}'} \bmod N$
$\forall i \in V : \tilde{C}_i \leftarrow R^{\tilde{m}_i} S^{\tilde{r}_i} \bmod N$
$\forall \bar{i}, \bar{j} \in \bar{V} : \tilde{R}_{\bar{i},\bar{j}} \leftarrow C_{\bar{i}}^{\tilde{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\tilde{b}_{\bar{i},\bar{j}}} S^{\tilde{r}_{\bar{i},\bar{j}}} \bmod N$

4. Compute hash:
$c \leftarrow \mathcal{H}(\texttt{context}, A', Z, C_i, \tilde{Z}, \tilde{C}_i, \tilde{R}_{\bar{i},\bar{j}}, n_3)$

5. Compute responses:
$\hat{e} \leftarrow \tilde{e} + c \cdot e'$
$\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$
$\hat{m}_0 \leftarrow \tilde{m}_0 + c \cdot m_0$
$\forall i \in V$:
$\hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
$\hat{r}_i \leftarrow \tilde{r}_i + c \cdot r_i$
$\forall (i,j) \in E$:
$\hat{m}_{(i,j)} \leftarrow \tilde{m}_{(i,j)} + c \cdot m_{(i,j)}$
$\forall \bar{i}, \bar{j} \in \bar{V}$:
$\hat{a}_{\bar{i},\bar{j}} \leftarrow \tilde{a}_{\bar{i},\bar{j}} + c \cdot a_{\bar{i},\bar{j}}$
$\hat{b}_{\bar{i},\bar{j}} \leftarrow \tilde{b}_{\bar{i},\bar{j}} + c \cdot b_{\bar{i},\bar{j}}$
$\hat{r}_{\bar{i},\bar{j}} \leftarrow \tilde{r}_{\bar{i},\bar{j}} + c \cdot r_{\bar{i},\bar{j}}$

6. Output $P_3 \leftarrow (c, A', \hat{e}, \hat{v}', \hat{m}_0, \hat{m}_i, \hat{m}_{(i,j)}, \hat{r}_i, \hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$

6.2 Geo-Location Separation Verification

- Proof of Geo-Separation (Verifying)

- Proving Knowledge of a Signature (GSPossessionVerifier())
- For the verification of the proof of geo-location separation, the *Verifier* operates over pair-wise different vertex encodings $\bar{i}$, $\bar{j}$ in $\bar{V}$.
- The *Verifier* receives an input:

$$P_3 = (c, A', \hat{e}, \hat{v}', \hat{m}_0, \hat{m}_i, \hat{m}_{(i,j)}, \hat{r}_i, \hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$$

1. Check lengths:
   $\hat{e} \in \pm\{0,1\}^{l'_e+\ell_\varnothing+\ell_H+1}$
   $\hat{v} \in \pm\{0,1\}^{\ell_v+\ell_\varnothing+\ell_H+1}$
   $\hat{m}_0 \in \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+2}$
   $\forall i \in V$:
   $\hat{m}_i \in \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+2}$
   $\hat{r}_i \in \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
   $\forall (i,j) \in E$:
   $\hat{m}_{(i,j)} \in \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+2}$
   $\forall \bar{i}, \bar{j} \in \bar{V}$:
   $\hat{a}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
   $\hat{b}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
   $\hat{r}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
2. **if** any length check fails **then reject** the proof and **abort** verification, outputting $\perp$.
3. Compute $\hat{t}$-values
   $\hat{Z} \leftarrow \left(\frac{Z}{A'^{2^{\ell_e}-1}}\right)^{-c} R_0^{\hat{m}_0}(R_{\pi(i)}^{\hat{m}_i} \cdots R_{\pi(i,j)}^{\hat{m}_{(i,j)}}) A'^{\hat{e}} S^{\hat{v}'} \bmod N$
   $\forall i \in V : \hat{C}_i \leftarrow C_i^{-c} R^{\hat{m}_i} S^{\hat{r}_i} \bmod N$
   $\forall \bar{i}, \bar{j} \in \bar{V} : \hat{R}_{\bar{i},\bar{j}} \leftarrow R^{-c} C_{\bar{i}}^{\hat{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\hat{b}_{\bar{i},\bar{j}}} S^{\hat{r}_{\bar{i},\bar{j}}} \bmod N$
4. Compute the verification challenge:
   $\hat{c} \leftarrow \mathcal{H}(\texttt{context}, A', Z, C_i, \hat{Z}, \hat{C}_i, \hat{R}_{\bar{i},\bar{j}}, n_3)$
5. Verify equality of challenge
   if $c = \hat{c}$ **then**
   accept
   else
   reject, outputting $\perp$.

We include the correctness proof for the proof of possession of P_3 in Figure 3. While the proof is straight-forward, we pay attention to the transformation between the blinded graph signature (A', e', v') used in the proof and the originally signed graph signature (A, e, v).

Figure 4 contains the correctness proof for the coprimality statement. We draw attention to the use of Bézout's Identity to prove that vertex representations $m_{\bar{i}}$ and $m_{\bar{j}}$ must be coprime:

$$1 = a_{\bar{i},\bar{j}} \cdot m_{\bar{i}} + b_{\bar{i},\bar{j}} \cdot m_{\bar{j}}$$

holds if and only if $m_{\bar{i}}$ and $m_{\bar{j}}$ are coprime. In a setting with unknown group order $\#\,\mathrm{QR}_N$, this yields an equation over integer exponents:

$$R \equiv R^{(a_{\bar{i},\bar{j}} \cdot m_{\bar{i}})} R^{b_{(\bar{i},\bar{j}} \cdot m_{\bar{j}})} \pmod{N},$$

which is used in Figure 3, Equation †.

7 Component Provers

Provers are organized to have a *Pre-Challenge Phase* and a *Post-Challenge Phase*. In the Pre-Challenge Phase, the *Prover* receives as input the reference of the values being proven (a graph signature, a commitment, etc.) and outputs the witnesses (t-values). In this phase, the prover computes and stores the witness randomness.

In the Post-Challenge Phase, the *Prover* takes as input the common challenge c and outputs the responses (hat-values) matching its secrets and witness randomness.

The sub-ordinate prover objects stay alive between Pre-Challenge and Post-Challenge Phase and keep their internal state.

Algorithm 21: blindGS(): Blinding of a given graph signature (A, e, v).

Input: Graph signature (A, e, v), *Signer* public key pk, gs_params
Pre-conditions: (A, e, v) is a valid graph signature under *Signer* public key pk.
Output: Blinded graph signature (A', e', v').
Post-conditions: $A', e', v')$ is a valid graph signature on the same graph G.

1 Choose blinding randomness uniformly at random $r_A \in_R \pm\{0,1\}^{\ell_n+\ell_\emptyset}$
2 Get base S and N from pk
3 $A' \leftarrow AS^{r_A} \pmod{N}$
4 $v' \leftarrow v - e \cdot r_A$
5 $e' \leftarrow e - 2^{\ell_e - 1}$
6 return (A', e', v')

7.1 Protocol: ProverOrchestrator()

7.1.1 Pre-Challenge Phase

Inputs

Verifier nonce $n_3 \in \{0,1\}^{\ell_H}$

1. Computation of commitments (delegated to Commitment() factory:

   1.1 The *Prover* computes commitments on all vertex representations:

$$C_i \leftarrow R^{e_i \Pi_{k \in f_{\mathcal{V}}(i)} e_k} S_i^r \bmod N$$

Given:

$P_3 = (c, A', \hat{e}, \hat{v}', \hat{m}_0, \hat{m}_i, \hat{m}_{(i,j)}, \hat{r}_i, \ldots)$

$A' \equiv AS^{r_A} \pmod N; \qquad v' = v - e \cdot r_A$

$e' = e - 2^{\ell_e - 1}$

$$\begin{aligned}
\hat{Z} &\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} R_0^{\hat{m}_0} \left(R_{\pi(i)}^{\hat{m}_i} \cdots R_{\pi(i,j)}^{\hat{m}_{(i,j)}}\right) A'^{\hat{e}} S^{\hat{v}'} \pmod N \\
&\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} R_0^{(\tilde{m}_0 + c \cdot m_0)} \left(R_{\pi(i)}^{(\tilde{m}_i + c \cdot m_i)} \cdots R_{\pi(i,j)}^{(\tilde{m}_{(i,j)} + c \cdot m_{(i,j)})}\right) A'^{(\tilde{e} + c \cdot e')} S^{(\tilde{v}' + c \cdot v')} \pmod N \\
&\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} R_0^{\tilde{m}_0} R_0^{c \cdot m_0} R_{\pi(i)}^{\tilde{m}_i} R_{\pi(i)}^{c \cdot m_i} \cdots R_{\pi(i,j)}^{\tilde{m}_{(i,j)}} R_{\pi(i,j)}^{c \cdot m_{(i,j)}} A'^{\tilde{e}} A'^{c \cdot e'} S^{\tilde{v}'} S^{c \cdot v'} \pmod N \\
&\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} \underbrace{\left(R_0^{\tilde{m}_0} R_{\pi(i)}^{\tilde{m}_i} \cdots R_{\pi(i,j)}^{\tilde{m}_{(i,j)}} A'^{\tilde{e}} S^{\tilde{v}'}\right)}_{\equiv \tilde{Z} \pmod N} \left(R_0^{c \cdot m_0} R_{\pi(i)}^{c \cdot m_i} \cdots R_{\pi(i,j)}^{c \cdot m_{(i,j)}} A'^{c \cdot e'} S^{c \cdot v'}\right) \pmod N \\
&\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} \tilde{Z} \left(R_0^{c \cdot m_0} R_{\pi(i)}^{c \cdot m_i} \cdots R_{\pi(i,j)}^{c \cdot m_{(i,j)}} A'^{c \cdot e'} S^{c \cdot v'}\right) \pmod N \\
&\equiv \left(\frac{Z}{A'^{2^{\ell_e-1}}}\right)^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} A'^{e'} S^{v'}\right)^c \pmod N
\end{aligned}$$

$$\begin{aligned}
&\equiv \left(\frac{Z}{(AS^{r_A})^{2^{\ell_e-1}}}\right)^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} (AS^{r_A})^{(e - 2^{\ell_e-1})} S^{(v - e \cdot r_A)}\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} (AS^{r_A})^{(e - 2^{\ell_e-1})} S^{-e \cdot r_A} S^v\right)^c \left((AS^{r_A})^{(2^{\ell_e-1})}\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} (AS^{r_A})^{(e - 2^{\ell_e-1})} S^{-e \cdot r_A} S^v (AS^{r_A})^{(2^{\ell_e-1})}\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} (AS^{r_A})^{e} S^{-e \cdot r_A} S^v\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} \left(R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} A^e S^{e \cdot r_A} S^{-e \cdot r_A} S^v\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} \left(\underbrace{R_0^{m_0} R_{\pi(i)}^{m_i} \cdots R_{\pi(i,j)}^{m_{(i,j)}} A^e S^v}_{\equiv Z \pmod N}\right)^c \pmod N \\
&\equiv Z^{-c} \tilde{Z} (Z)^c \overset{!}{\equiv} \tilde{Z} \pmod N \qquad \square
\end{aligned}$$

Figure 3: Correctness proof for the verification of the proof of possession of P_3. The second part of the proof transforms the blinded graph signature (A', e', v') to the original graph signature (A, e, v).

Given:

$$
\begin{aligned}
P_3 &= (c, \ldots, \hat{m}_i, \ldots, \hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}}) \\
r_{\bar{i},\bar{j}} &= -r_i \cdot a_{\bar{i},\bar{j}} - r_j \cdot b_{\bar{i},\bar{j}} \\
1 &= a_{\bar{i},\bar{j}} \cdot m_{\bar{i}} + b_{\bar{i},\bar{j}} \cdot m_{\bar{j}}
\end{aligned}
$$

$$
\begin{aligned}
\hat{R}_{\bar{i},\bar{j}} &\equiv R^{-c} C_{\bar{i}}^{\hat{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\hat{b}_{\bar{i},\bar{j}}} S^{\hat{r}_{\bar{i},\bar{j}}} \pmod N \\
&\equiv R^{-c}\ C_{\bar{i}}^{(\tilde{a}_{\bar{i},\bar{j}} + c \cdot a_{\bar{i},\bar{j}})} C_{\bar{j}}^{(\tilde{b}_{\bar{i},\bar{j}} + c \cdot b_{\bar{i},\bar{j}})} S^{(\tilde{r}_{\bar{i},\bar{j}} + c \cdot r_{\bar{i},\bar{j}})} \pmod N \\
&\equiv R^{-c}\ \underbrace{\left(C_{\bar{i}}^{\tilde{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\tilde{b}_{\bar{i},\bar{j}}} S^{\tilde{r}_{\bar{i},\bar{j}}} \right)}_{\equiv \tilde{R}_{\bar{i},\bar{j}} \pmod N} \left(C_{\bar{i}}^{(c \cdot a_{\bar{i},\bar{j}})} C_{\bar{j}}^{(c \cdot b_{\bar{i},\bar{j}})} S^{(c \cdot r_{\bar{i},\bar{j}})} \right) \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left(C_{\bar{i}}^{(c \cdot a_{\bar{i},\bar{j}})} C_{\bar{j}}^{(c \cdot b_{\bar{i},\bar{j}})} S^{(c \cdot r_{\bar{i},\bar{j}})} \right) \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left(C_{\bar{i}}^{a_{\bar{i},\bar{j}}} C_{\bar{j}}^{b_{\bar{i},\bar{j}}} S^{r_{\bar{i},\bar{j}}} \right)^c \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left((R^{m_{\bar{i}}} S^{r_{\bar{i}}})^{a_{\bar{i},\bar{j}}}\ (R^{m_{\bar{j}}} S^{r_{\bar{j}}})^{b_{\bar{i},\bar{j}}}\ S^{r_{\bar{i},\bar{j}}} \right)^c \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left(R^{(a_{\bar{i},\bar{j}} \cdot m_{\bar{i}})} R^{(b_{\bar{i},\bar{j}} \cdot m_{\bar{j}})}\ S^{(a_{\bar{i},\bar{j}} \cdot r_{\bar{i}})} S^{(b_{\bar{i},\bar{j}} \cdot r_{\bar{j}})} S^{r_{\bar{i},\bar{j}}} \right)^c \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left(R^{(a_{\bar{i},\bar{j}} \cdot m_{\bar{i}})} R^{(b_{\bar{i},\bar{j}} \cdot m_{\bar{j}})}\ \underbrace{S^{(a_{\bar{i},\bar{j}} \cdot r_{\bar{i}})} S^{(b_{\bar{i},\bar{j}} \cdot r_{\bar{j}})} S^{(-r_i \cdot a_{\bar{i},\bar{j}} - r_j \cdot b_{\bar{i},\bar{j}})}}_{\equiv 1 \pmod N} \right)^c \pmod N \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ \left(\underbrace{R^{(a_{\bar{i},\bar{j}} \cdot m_{\bar{i}})} R^{(b_{\bar{i},\bar{j}} \cdot m_{\bar{j}})}}_{\equiv R \pmod N} \right)^c \pmod N \qquad (\dagger) \\
&\equiv R^{-c}\ \tilde{R}_{\bar{i},\bar{j}}\ R^c \overset{!}{\equiv} \tilde{R}_{\bar{i},\bar{j}} \qquad \square
\end{aligned}
$$

Figure 4: Correctness proof for the verification of the coprimality proof of P_3. Its final equation † uses Bézout's Identity and the proven coprimality of $m_{\bar{i}}$ and $m_{\bar{j}}$.

1.2 The *Prover* stores the public values and commitment randomness in the common values store:
- $\forall i \in V$:
  $\mathsf{store}(C_i, r_i)$

2. The *Prover* obtains blinded graph signature (A', e', v'), running Algorithm 21.

3. The *Prover* stores the blinded graph signature
   $\mathsf{store}(A', e', v')$

4. The *Prover* calls upon the component provers to execute their pre-computations.
   - The PairWiseDifferenceProver() will compute the coprimality evidence for its proofs based on the commitments C_i stored.

5. The *Prover* calls the *Pre-Challenge Phase* of component provers in turn, with references to messages addressed, each returning a $\tilde{t}$-value as output of their first phase.

   5.1 $(\tilde{Z}) \leftarrow \mathsf{GSPossessionProver}()$

   5.2 $\forall i \in V$:
   $(\tilde{C}_i) \leftarrow \mathsf{CommitmentProver}(i)$

   5.3 $\forall \bar{i}, \bar{j} \in \bar{V}$:
   $(\tilde{R}_{\bar{i},\bar{j}}) \leftarrow \mathsf{PairWiseDifferenceProver}(\bar{i}, \bar{j})$

   The component prover instances are kept alive for the *Post-Challenge Phase*.

6. **if** any prover returns $\perp$ **then abort** the proof, outputting $\perp$.

7. Compute the common challenge:

   7.1 The *Prover* gathers the ordered list of all public values and all $\tilde{t}$-values provided by component provers.

   7.2 The *Prover* canonicalizes and serializes the concatenation of proof values according to a pre-defined order: `context` first, then general public values (A', Z), then commitments and other public values, then witnesses, nonce n_3 finally.

   7.3 The *Prover* computes the hash of this string to produce the Fiat-Shamir challenge:
   $$c \leftarrow \mathcal{H}(\texttt{context}, A', Z, C_i, \tilde{Z}, \tilde{C}_i, \tilde{R}_{\bar{i},\bar{j}}, n_3)$$

8. The *Prover* calls the *Post-Challenge Phase* of component provers in turn, each prover returning its responses, the hat-values.

   8.1 $(\hat{e}, \hat{v}', \hat{m}_0, \forall i \in V : \hat{m}_i, \forall (i,j) \in E : \hat{m}_{(i,j)}) \leftarrow \mathsf{GSPossessionProver}()$

   8.2 $\forall i \in V$:
   $(\hat{r}_i) \leftarrow \mathsf{CommitmentProver}(i)$

   8.3 $\forall \bar{i}, \bar{j} \in \bar{V}$:
   $(\hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}}) \leftarrow \mathsf{PairWiseDifferenceProver}(\bar{i}, \bar{j})$

9. **if** any component prover response is $\perp$ **then abort** the proof, outputting $\perp$.

10. The *Prover* assembles the final proof.

    10.1 The *Prover* creates an ordered list prefixed by (c, A') followed by the ordered lists of responses of the component provers.

    10.2 The *Prover* canonicalizes and serializes the proof:

$$P_3 \leftarrow (c, A', \hat{e}, \hat{v}', \hat{m}_0, \hat{m}_i, \hat{m}_{(i,j)}, \hat{r}_i, \hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$$

11. The *Prover* sends P_3 to the *Verifier*.

Outputs

Proof:
Challenge c
Blinded graph signature A'
Responses/hat-values $\hat{s}$

7.2 Protocol: GSPossessionProver()

Governed State

Witness randomness:
- $\tilde{e}, \tilde{v}, \tilde{m}_0$
- $\forall i \in V$:
  $\tilde{m}_i \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
- $\forall (i,j) \in E$:
  $\tilde{m}_{(i,j)} \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$

7.2.1 Pre-Challenge Phase

Inputs

Blinded graph signature (A', e', v')

1. Choose witness randomness uniformly at random:
   - $\tilde{e} \in_R \pm\{0,1\}^{\ell'_e+\ell_\emptyset+\ell_H+1}$
   - $\tilde{v} \in_R \pm\{0,1\}^{\ell_v+\ell_\emptyset+\ell_H+1}$
   - $\tilde{m}_0 \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
   - $\forall i \in V$:
     $\tilde{m}_i \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$
   - $\forall (i,j) \in E$:
     $\tilde{m}_{(i,j)} \in_R \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+1}$

2. Store the witnesses as common values:

- $\mathsf{store}(\tilde{e}, \tilde{v}, \tilde{m}_0)$
- $\forall i \in V : \mathsf{store}(\tilde{m}_i)$
- $\forall (i,j) \in E : \mathsf{store}(\tilde{m}_{(i,j)})$

3. Compute $\mathsf{GSPossessionProver}()$ witness:

$$\tilde{Z} \leftarrow (A')^{\tilde{e}} R_0^{\tilde{m}_0} R_{\pi(i)}^{\tilde{m}_i} \cdots R_{\pi(i,j)}^{\tilde{m}_{(i,j)}} S^{\tilde{v}'} \bmod N$$

4. Output $(\tilde{Z})$.

Outputs

Witness $\tilde{Z}$

7.2.2 Post-Challenge Phase

Inputs

Challenge c

1. Retrieve witnesses from the common values store:
   - $\mathsf{retrieve}(\tilde{e}, \tilde{v}, \tilde{m}_0)$
   - $\forall i \in V : \mathsf{retrieve}(\tilde{m}_i)$
   - $\forall (i,j) \in E : \mathsf{retrieve}(\tilde{m}_{(i,j)})$
2. Retrieve the secrets of the blinded graph signature from the common values store:
   - $\mathsf{retrieve}(e', v', m_0)$
   - $\forall i \in V : \mathsf{retrieve}(m_i)$
   - $\forall (i,j) \in E : \mathsf{retrieve}(m_{(i,j)})$
3. Compute the $\mathsf{GSPossessionProver}()$ responses:
   - $\hat{e} \leftarrow \tilde{e} + c \cdot e'$
   - $\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$
   - $\hat{m}_0 \leftarrow \tilde{m}_0 + c \cdot m_0$
   - $\forall i \in V :$
     $\hat{m}_i \leftarrow \tilde{m}_i + c \cdot m_i$
   - $\forall (i,j) \in E :$
     $\hat{m}_{(i,j)} \leftarrow \tilde{m}_{(i,j)} + c \cdot m_{(i,j)}$
4. Output list of responses:

$$(\hat{e}, \hat{v}', \hat{m}_0, \forall i \in V : \hat{m}_i, \forall (i,j) \in E : \hat{m}_{(i,j)})$$

Outputs

Responses $(\hat{e}, \hat{v}', \hat{m}_0, \hat{m}_i, \hat{m}_{(i,j)})$

7.3 Protocol: CommitmentProver()

Governed State

Witness randomness:
- $\tilde{r}_i$

7.3.1 Pre-Challenge Phase

Inputs

Message reference m_i

1. Choose witness randomness uniformly at random:
   - $\tilde{r}_i \in_R \pm\{0,1\}^{\ell_n+\ell_\phi+\ell_H+1}$
2. Store the witness:
   - $\mathsf{store}(\tilde{r}_i)$
3. Retrieve the witness randomness of committed messages from the common values store:
   $\mathsf{retrieve}(\tilde{m}_i)$
4. Compute CommitmentProver() witness $\tilde{C}_i$:
$$\tilde{C}_i \leftarrow R^{\tilde{m}_i} S^{\tilde{r}_i} \bmod N$$
5. Output $\tilde{C}_i$.

Outputs

Witness $\tilde{C}_i$

7.3.2 Post-Challenge Phase

Inputs

Challenge c

1. Retrieve witnesses from the common values store:
   - $\mathsf{retrieve}(\tilde{r}_i)$
2. Retrieve the commitment randomness from the common values store:
   - $\mathsf{retrieve}(r_i)$

3. Compute the CommitmentProver() response:
   $\hat{r}_i \leftarrow \tilde{r}_i + c \cdot r_i$

4. Output $\hat{r}_i$

Outputs

Responses $(\hat{r}_i)$

7.4 Protocol: PairWiseDifferenceProver()

Governed State

Coprimality Secrets:
- $a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}}$
- $r_{\bar{i},\bar{j}}$

Witness randomness:
- $\tilde{a}_{\bar{i},\bar{j}}, \tilde{b}_{\bar{i},\bar{j}}$
- $\tilde{r}_{\bar{i},\bar{j}}$

7.4.1 Pre-Challenge Phase

Inputs

Pair-wise different commitment references $\bar{i}$ and $\bar{j}$

1. Precomputation:

   1.1 Retrieve the commitment messages and randomness from the common values store:
   - $\mathsf{retrieve}(m_i, r_i)$
   - $\mathsf{retrieve}(m_j, r_j)$

   1.2 Solve the Extended Euclidian Algorithm for m_i and m_j:

   $$(d_{\bar{i},\bar{j}}, a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}}) \leftarrow \mathsf{EEA}(m_i, m_j)$$

   1.3 **if** $d_{\bar{i},\bar{j}} \neq 1$ **then abort** the proof, because m_i and m_j are not coprime.

   1.4 Compute the differential commitment randomnesss:

   $$r_{\bar{i},\bar{j}} \leftarrow -r_{\bar{i}} \cdot a_{\bar{i},\bar{j}} - r_{\bar{j}} \cdot b_{\bar{i},\bar{j}}$$

   1.5 Store the coprimality secrets and differential randomness:
   - $\mathsf{store}(a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}})$
   - $\mathsf{store}(r_{\bar{i},\bar{j}})$

2. Choose witness randomness uniformly at random:
   - $\tilde{a}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
   - $\tilde{b}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
   - $\tilde{r}_{\bar{i},\bar{j}} \in_R \pm\{0,1\}^{\ell_n+\ell_\varnothing+\ell_H+1}$
3. Store witness randomness:
   - $\mathsf{store}(\tilde{a}_{\bar{i},\bar{j}}, \tilde{b}_{\bar{i},\bar{j}})$
   - $\mathsf{store}(\tilde{r}_{\bar{i},\bar{j}})$
4. Compute PairWiseDifferenceProver() Witness:

$$\tilde{R}_{\bar{i},\bar{j}} \leftarrow C_{\bar{i}}^{\tilde{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\tilde{b}_{\bar{i},\bar{j}}} S^{\tilde{r}_{\bar{i},\bar{j}}} \bmod N$$

5. Output $\tilde{R}_{\bar{i},\bar{j}}$

Outputs

Witness: $\tilde{R}_{\bar{i},\bar{j}}$

7.4.2 Post-Challenge Phase

Inputs

Challenge c

1. Retrieve coprimality secrets:
   - $\mathsf{retrieve}(a_{\bar{i},\bar{j}}, b_{\bar{i},\bar{j}})$
   - $\mathsf{retrieve}(r_{\bar{i},\bar{j}})$
2. Retrieve witness randomness:
   - $\mathsf{retrieve}(\tilde{a}_{\bar{i},\bar{j}}, \tilde{b}_{\bar{i},\bar{j}})$
   - $\mathsf{retrieve}(\tilde{r}_{\bar{i},\bar{j}})$
3. Compute the PairWiseDifferenceProver() responses:
   - $\hat{a}_{\bar{i},\bar{j}} \leftarrow \tilde{a}_{\bar{i},\bar{j}} + c \cdot a_{\bar{i},\bar{j}}$
   - $\hat{b}_{\bar{i},\bar{j}} \leftarrow \tilde{b}_{\bar{i},\bar{j}} + c \cdot b_{\bar{i},\bar{j}}$
   - $\hat{r}_{\bar{i},\bar{j}} \leftarrow \tilde{r}_{\bar{i},\bar{j}} + c \cdot r_{\bar{i},\bar{j}}$
4. Output $(\hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$

Outputs

Responses: $(\hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$

8 Component Verifiers

Each *Verifier* is responsible for taking the common values, the public values and the responses (hat-values) of its protocol as input and to output the candidate verification witness (hat-value) as output.

8.1 VerifierOrchestrator()

Inputs

Proof P_3, incl.
Challenge c
Public values Z, A', C_i

1. The *Verifier* populates its common values store with the public values of the proof.
   - $\mathsf{store}(Z, A')$
   - $\mathsf{store}(C_i)$
2. The *Verifier* calls the component verifiers in turn, offering as input the challenge c along with the corresponding component prover responses. Each component verifier will return a single $\hat{t}$ value, a verifier witness.
   2.1 $\hat{Z} \leftarrow \mathsf{GSPossessionVerifier}(c, A', (\hat{e}, \hat{v}', \hat{m}_0, \forall i \in V : \hat{m}_i, \forall (i,j) \in E : \hat{m}_{(i,j)}))$
   2.2 $\forall i \in V$:
   $\hat{C}_i \leftarrow \mathsf{CommitmentVerifier}(c, (\hat{r}_i))$
   2.3 $\forall \bar{i}, \bar{j} \in \bar{V}$:
   $\hat{R}_{\bar{i},\bar{j}} \leftarrow \mathsf{PairwiseDifferenceVerifier}(c, (\hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}}))$

   2.1 The *Verifier* gathers the ordered list of all public values and all $\hat{t}$-values provided by component verifiers.
   2.2 The *Verifier* canonicalizes and serializes the concatenation of the $\hat{t}$-values according to a pre-defined order: `context` first, then general public values (A', Z), then commitments and other public values, then $\hat{t}$-values, nonce n_3 finally.
   2.3 The *Verifier* computes the hash of this string to produce the *Verifier*'s version of the Fiat-Shamir challenge $\hat{c}$:

$$\hat{c} \leftarrow \mathcal{H}(\texttt{context}, A', Z, C_i, \hat{Z}, \hat{C}_i, \hat{R}_{\bar{i},\bar{j}}, n_3)$$

3. Verify equality of challenge
   if $c = \hat{c}$ **then**
   accept
   else
   reject, outputting $\perp$.

Outputs

8.2 Protocol: GSPossessionVerifier()

Inputs

Challenge c
Blinded graph signature A'
GSPossessionProver() responses: $(\hat{e}, \hat{v}', \hat{m}_0, \forall i \in V : \hat{m}_i, \forall (i,j) \in E : \hat{m}_{(i,j)})$

1. Check lengths:
   - $\hat{e} \in \pm\{0,1\}^{l'_e+\ell_\emptyset+\ell_H+1}$
   - $\hat{v} \in \pm\{0,1\}^{\ell_v+\ell_\emptyset+\ell_H+1}$
   - $\hat{m}_0 \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$
   - $\forall i \in V$:
     $\hat{m}_i \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$
   - $\forall (i,j) \in E$:
     $\hat{m}_{(i,j)} \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$
2. **if** any length check fails **then reject** the proof and **abort** verification, outputting $\perp$.
3. Compute GSPossessionVerifier() $\hat{t}$-value:

$$\hat{Z} \leftarrow \left(\frac{Z}{A'^{2^{\ell_e}-1}}\right)^{-c} R_0^{\hat{m}_0}(R_{\pi(i)}^{\hat{m}_i} \cdots R_{\pi(i,j)}^{\hat{m}_{(i,j)}}) A'^{\hat{e}} S^{\hat{v}'} \bmod N$$

4. Output $\hat{Z}$.

Outputs

Verification witness: $\hat{Z}$

8.3 Protocol: CommitmentVerifier()

Inputs

Challenge c
CommitmentProver() response: $\hat{r}_i$

1. Retrieve corresponding message secrets.
   retrieve($\hat{m}_i$)
2. Check lengths:
   - $\hat{r}_i \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
   - $\hat{m}_i \in \pm\{0,1\}^{\ell_m+\ell_\emptyset+\ell_H+2}$

3. **if** any length check fails **then reject** the proof and **abort** verification, outputting $\perp$.

4. Compute the CommitmentVerifier() $\hat{t}$-value:

$$\hat{C}_i \leftarrow C_i^{-c} R^{\hat{m}_i} S^{\hat{r}_i} \bmod N$$

5. Output $\hat{C}_i$.

Outputs

Verifier witness: $\hat{C}_i$

8.4 Protocol: PairWiseDifferenceVerifier()

Inputs

Challenge c
PairWiseDifferenceProver() responses: $(\hat{a}_{\bar{i},\bar{j}}, \hat{b}_{\bar{i},\bar{j}}, \hat{r}_{\bar{i},\bar{j}})$

1. Check lengths:
   - $\hat{a}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
   - $\hat{b}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$
   - $\hat{r}_{\bar{i},\bar{j}} \in \pm\{0,1\}^{\ell_n+\ell_\emptyset+\ell_H+1}$

   if any length check fails **then reject** the proof and **abort** verification, outputting $\perp$.

2. Compute the PairWiseDifferenceVerifier() $\hat{t}$ value:

$$\hat{R}_{\bar{i},\bar{j}} \leftarrow R^{-c} C_{\bar{i}}^{\hat{a}_{\bar{i},\bar{j}}} C_{\bar{j}}^{\hat{b}_{\bar{i},\bar{j}}} S^{\hat{r}_{\bar{i},\bar{j}}} \bmod N$$

3. Output $\hat{R}_{\bar{i},\bar{j}}$

Outputs

Verifier witness: $\hat{R}_{\bar{i},\bar{j}}$

9 Recommendations

To securely use the graph signature scheme and the TOPOCERT tool, it is necessary to follow key size requirements specified for SRSA Camenisch-Lysyanskaya signatures [CL02, IBM13]. In general, the TOPOCERT tool is meant to operate in an implementation with a 2048-bits key strength and appropriately selected parameters, which

Table 1: Parameters of the graph signature scheme *gs_params* and encoding setup *enc_params*. Parameters for the underlying Camenisch-Lysyanskaya signature scheme are largely adapted from the Identity Mixer Specification [IBM13]. In the implementation, this table is referred to as `table:params`.

(a) Parameters of Keygen()

Parameter	Description	Bit-length
ℓ_n	Bit length of the special RSA modulus	2048
ℓ_Γ	Bit length of the commitment group	1632
ℓ_ρ	Bit length of the prime order of the subgroup of Γ	256
ℓ_m	Maximal bit length of messages encoding vertices and edges	256
ℓ_{res}	Number of reserved messages	1†
ℓ_e	Bit length of the certificate component e	597
ℓ'_e	Bit length of the interval the e values are taken from	120
ℓ_v	Bit length of the certificate component v	2724
$\ell_\varnothing$	Security parameter for statistical zero-knowledge	80
ℓ_H	Bit length of the cryptographic hash function used for the Fiat-Shamir Heuristic	256
ℓ_r	Security parameter for the security proof of the CL-scheme	80
ℓ_{pt}	The prime number generation to have an error probability to return a composite of $1 - 1/2^{\ell_{pt}}$	80†

Note: † refers to numbers that are integers, not bit lengths.

(b) Parameters of GraphEncodingSetup()

$\ell_\mathcal{V}$	Maximal number of vertices to be encoded	1000†‡
$\ell'_\mathcal{V}$	Reserved bit length for vertex encoding (bit length of the largest encodable prime representative)	120
$\ell_\mathcal{E}$	Maximal number of edges to be encoded	50.000†‡
$\ell_\mathcal{L}$	Maximal number of labels to be encoded	256†‡
$\ell'_\mathcal{L}$	Reserved bit length for label encoding	16

Note: † refers to numbers that are integers, not bit lengths; ‡ refers to the default parameter, not the theoretical maximum.

implies parameters as defined in Table 1. For a detailed specification of the parameter selection for the underlying Camenisch-Lysyanskaya signature scheme, we refer to Tables 2 and 3 of the Specification of the Identity Mixer Cryptographic Library, Version 2.3.40, on p. 43 [IBM13].

Remark 1 (Security Parameter). The security parameters, especially bit length for the group setups, flow from the specification of the bit length of the special RSA modulus ℓ_n and the message space ℓ_m. The constraints placed on the respective bit lengths are crucial to maintain the soundness of the security proof of the underlying Camenisch-Lysyanskaya signature scheme (cf. Table 3 of the Specification of the Identity Mixer Cryptographic Library, Version 2.3.40, on p. 43 [IBM13]).

Remark 2 (Encoding Parameters). We consider the choices made for the graph encoding scheme.

Encoding Defaults The bit length parameters for the prime encoding $\ell'_{\mathcal{V}}$ and $\ell'_{\mathcal{L}}$ follow from the available message bit length, assuming that the labels are encoded as the lowest prime representatives. However, the given defaults for number of vertices, edges, labels to be encoded $\ell_{\mathcal{V}}$, $\ell_{\mathcal{E}}$, and $\ell_{\mathcal{L}}$ are not the theoretical maxima.

Maximal Number of Labels For a single-labeled graph with $\ell'_{\mathcal{L}} = 16$, the maximal encodable number of labels is 6542. The restrictions of the number of labels is in place to allow for multi-labeled graphs, in which case the product of the label identifiers occupies the reserved space.

Maximal Number of Vertices The maximal number of vertices for the reserved bit length $\ell'_{\mathcal{V}} = 120$ is 1.59810^{34}. The limiting factor for the number of encoded vertices, however, is *not* the reserved bit length of the message space, but the space required to store the corresponding based dedicated vertex and edge encoding. For each possibly encodable vertex and edge the graph signature scheme needs to reserve a group element with an bit length of $\ell_n = 2048$. A encoding for fully connected graphs with $\ell_{\mathcal{V}} = 1000$ and $\ell_{\mathcal{E}} = \ell_{\mathcal{V}}(\ell_{\mathcal{V}} - 1) = 999000$ would consume 244.28 kBytes for vertices and 243.89 MBytes for the edges.

Remark 3 (Signature Size). A signature of the graph signature scheme consists of one group element and two exponents A, e, v). A single signature has the following bit length for the default parameters in Table 1:

$$|(A, e, v)|_2 = \ell_n + \ell_v + \ell_e = 5369 \text{ bits.}$$

Remark 4 (Base Randomization). We note here that the graph signature scheme proposed by Groß [Gro15] requires a base randomization for multi-use confidentiality of graph elements. This is because the bases referenced in the ZKPoK are public knowledge and each proof reveals which exponents are harbored by which base.

The base randomization asks that random permutations $\pi_{\mathcal{V}}$ and $\pi_{\mathcal{E}}$ be applied to the vertex and edge bases respectively. A space-efficient solution for that requirement could use keyed pseudorandom permutations.

Let an appropriate family of pseudorandom permutations $\mathcal{F}$ on group elements in QR_N be given, where pseudorandom permutations (PRPs) are defined as by Katz and

Lindell [KL14]. Theoretical work on constructions of pseudorandom permutations from pseudorandom functions was spawned by the seminal work of Luby and Rackoff [LR88]. As an alternative approach, we also refer to constructions of Verifiable Secret Shuffles, such as Neff [Nef01], which allow a Prover in a honest-verifier zero-knowledge proof scenario to convince a Verifier that a secret shuffled was computed correctly.

1. During the Signer's round of HiddenSign(), S chooses a uniformly random permutation key k with appropriate bit length.
2. S applies the pseudorandom permutations $(\pi_{\mathcal{V}}, \pi_{\mathcal{E}})$ with common key k to the certified base sets, obtaining permuted base sets.
3. Signer S then encodes graph $\mathcal{G}$ on the derived base sets.
4. Signer S shares permutation key k with the Recipient together with the corresponding signature $\sigma_k = (A, e, v)_k$ along with a proof of representation that σ_k indeed fulfills the CL-equation on the derived bases.

Hence, the Signer will issue multiple signatures, one for each permutation. Each signature has a size of one group element, two exponents, and one permutation key.

References

[Cam17] Jan Camenisch. private communication, 2017.

[CL02] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN*, LNCS. Springer, 2002.

[CS97] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In *CRYPTO*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[CS99] Ronald Cramer and Victor Shoup. Signature Schemes Based on the Strong RSA Assumption. *IACR Cryptology ePrint Archive*, 1999.

[Gro14] Thomas Groß. Rapid issuing of Camenisch-Lysyanskaya signatures: Signing with a constant number of exponentiations. Technical Report CS-TR-1418, Newcastle University, May 2014.

[Gro15] Thomas Groß. Signatures and efficient proofs on committed graphs and NP-statements. In *19th International Conference on Financial Cryptography and Data Security (FC 2015)*, pages 293–314, 2015.

[IBM13] IBM. Specification of the Identity Mixer cryptographic library, v. 2.3.40. Specification, IBM Research, January 2013. http://prime.inf.tu-dresden.de/idemix/.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[MVOV96] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[Nef01] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[Sho09] Victor Shoup. *A computational introduction to number theory and algebra (2nd ed.)*. Cambridge university press, 2009.