

Anonyma: Anonymous Invitation-Only Registration in Malicious Adversarial Model*

SANAZ TAHERI BOSHROOYEH, Koç University, Turkey

ALPTEKİN KÜPÇÜ, Koç University, Turkey

ÖZNUR ÖZKASAP, Koç University, Turkey

In invitation-based systems, a new user can register upon having a certain number of invitations (i.e., t) issued by the existing members. The newcomer hands his invitations to the system administrator who verifies whether the invitations are issued by legitimate members. This causes the administrator to be aware of who is invited by whom. However, the inviter-invitee relationship is privacy-sensitive information and can lead to inference attacks where the invitee's profile (e.g., political view or location) can be extracted through the profiles of his inviters. Addressing this problem, we propose *Anonyma*, an anonymous invitation-based system where a corrupted administrator who may even collude with a subset of existing members is not able to figure out who is invited by whom. We formally define and prove the inviter anonymity as well as unforgeability of invitations against a *malicious adversary*. Our design only incurs a constant cost to authenticate a new registration. This is significantly better than the similar works where the generation of invitations and verification of new registration cause an overhead linear in the total number of existing members. Besides, *Anonyma* is efficiently scalable in the sense that once a user joins the system, the administrator can instantly, and without re-keying the existing members, issue credentials for the newcomer to be able to act as an inviter. We additionally design *AnonymaX*, an anonymous cross-network invitation-based system empowering third-party authentication where the invitations issued by the members of one system can be used for registering to another system.

Additional Key Words and Phrases: Invitation-Based System, Anonymity, Unforgeability, Integrity, Cross-Network Invitation, Third-party Authentication, Malicious Adversary.

ACM Reference Format:

Sanaz Taheri Boshrooyeh, Alptekin Küpçü, and Öznur Özkasap. 2021. *Anonyma*: Anonymous Invitation-Only Registration in Malicious Adversarial Model. *ACM Trans. Priv. Sec.* 1, 1 (February 2021), 35 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Motivation: Invitation-based systems (or invitation-only registration systems, interchangeably) are services where registration is possible only through getting invitations from the current members of the system. Many reasons encourage invitation-only registration policy, e.g., the lack of sufficient resources to cover arbitrary many users, improving the quality of service by constraining the number of members, and to protect the system against spammers or undesirable users¹.

*This is an extension to our prior publication Inonymous [7]

¹<https://pieregister.com/features/invitation-based-registrations/>

Authors' addresses: Sanaz Taheri Boshrooyeh, Koç University, Rumelifeneri Yolu, Sariyer, İstanbul, Turkey, staheri14@ku.edu.tr; Alptekin Küpçü, Koç University, Rumelifeneri Yolu, Sariyer, İstanbul, Turkey, akupcu@ku.edu.tr; Öznur Özkasap, Koç University, Rumelifeneri Yolu, Sariyer, İstanbul, Turkey, oozkasap@ku.edu.tr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

2471-2566/2021/2-ART \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Invitation-only registration overview: The ecosystem of an invitation-only system, depicted in Figure 1, is comprised of an *administrator/server*, a set of *members*, who act as *inviters*, and a new user, called an *invitee*, who wants to register to the system. At the beginning of the system lifetime, the server registers an initial set of members who shall start inviting outsiders. For instance, in the Google Inbox service (which was initially started on an invitation-only policy), the employees of Google can be the initial members. The invitee receives invitations from a subset of existing members, i.e., inviters. The invitee knows the inviters beforehand via some other means outside the network to be joined to (e.g., Google employees invite their families/friends). The invitee submits the invitations to the server. The server verifies the invitation and accepts or rejects the registration request accordingly. Note that during the verification process the server gets to learn the exact inviters of the invitee.

Note that registration is a one-time action that is regulated by invitation-policy, however, further logins to the system are administrated by other authentication techniques like a password. Thus, being an inviter of someone does not give access to the account of the invited person.

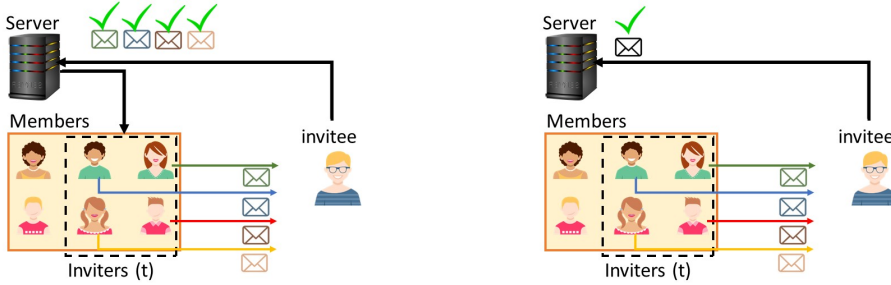


Fig. 1. The workflow of an invitation-only registration system. The invitee collects the individual invitations and hands them to the group administrator to verify they are issued by legit members. Thus, the administrator knows the identity of the inviters.

Current applications of invitation-only registration: Google initiated this invitation-only policy when deploying services such as Google Inbox, Orkut, and Google Wave². Another successful system with an invitation-only registration is *Spotify*³. *Facebook* also has secret groups in which new users can participate upon getting invitations from other group members⁴. Similarly, messengers such as *WhatsApp*⁵ and *Telegram*⁶ offer private groups running on the invite-only basis. A similar approach is sought in *PIE Register*⁷ where it enables users to set up exclusive websites whose contents are only visible to the visitors invited by the website administrator or authorized members. Another closely relevant example is the trustee-based social authentication deployed by Facebook as a backup authentication method [9, 22, 33]. A backup authentication method is used when the user fails to pass the primary authentication, e.g., forgetting the password [22]. The account holder determines a set of trustees and informs the server in advance. When the user loses access to his account, the server sends recovery codes to the trustees (equivalent to inviters in the invitation-only registration scenario). Upon collection of enough number (recovery threshold) of codes from the trustees and handing the codes to the server, the user regains access to his account.

²<http://www.macworld.com/article/1055383/gmail.html>

³<https://community.spotify.com/t5/Accounts/Spotify-Family-Q-amp-A/td-p/988520>

⁴<https://blog.hootsuite.com/facebook-secret-groups/>

⁵<https://faq.whatsapp.com/en/android/26000123/?category=5245251>

⁶<https://telegram.org/tour/groups>

⁷<https://pieregister.com/features/invitation-based-registrations>.

Security issues in invitation-only registration: Apart from the benefits of invitation-only systems, they are prone to inference attacks which were initially discussed in Inonymous [7]. In invitation-based systems, the administrator receives all the invitations issued for an invitee to check whether they are originated from legitimate members. As such, the system administrator knows who is invited by whom. In some other cases like the Telegram chat service, the situation is worse and indeed all the members of the group receive a notification of who is invited by whom. The inviters are mostly among the newcomer's acquaintances, e.g., colleagues, home mates, family members, and close friends who usually have many common features with the invitee. Several studies have shown that information such as location, religious beliefs, sexual orientation, and political views can be extracted about an individual by analyzing the common features among his connections, namely, inviters in our case [12, 21, 32]. This leakage is known as an inference attack. Due to this issue, inviter-invitee relation counts as privacy-sensitive information.

Related Work: The problem of inference attack in the invitation-based system has been addressed in our prior work, Inonymous [7]. That proposal enables an invitee to prove to the group administrator the possession of enough legitimate invitations without disclosing the identity of her inviters, thus protecting the anonymity of her inviters. Additionally, Inonymous guarantees the unforgeability of invitations in the presence of an adversarial invitee. That is, no invitee with insufficient inviters would be able to join the system. The anonymity of inviters and unforgeability of invitations are formally proved under an *honest but curious (HbC)* adversarial setting, namely, all the parties follow the protocol descriptions honestly. However, the HbC secure protocol of Inonymous cannot stand an active adversarial model where parties may deviate from protocols' descriptions. In specific, if a corrupted administrator disregards the protocol description, she can damage inviter anonymity and identify who is invited by whom. In *Anonyma*, we extend the proposal of Inonymous to withstand an *active adversarial model*.

In addition to Inonymous, some other lines of research like group signature schemes, threshold ring-based signature schemes, selective-disclosure credentials, and electronic voting systems share some-what similar system model and security objectives i.e., inviter anonymity and invitation unforgeability. However, they suffer from two major issues when utilized for the invitation-only registration scenario 1) computational and memory inefficiencies due to their application-specific design choices 2) the lack of *inviter anonymity*. Note that the term *anonymity* has been used in anonymous credentials, however, anonymity in that context refers to *attribute anonymity* and should not be confused by *inviter anonymity* which is required in an anonymous invitation-based system. Attribute anonymity stands for the unlinkability of a credential holder (i.e., an invitee) to a set of attributes in her credentials (i.e., her invitation) whereas inviter anonymity aims at hiding the connection between the issuing authority (i.e., the inviter) and the recipient (i.e., the invitee). As such, attribute anonymity does not imply inviter anonymity. An extensive comparison of *Anonyma* with related studies is provided in section 7.

Anonyma: An overview of *Anonyma* is depicted in Figure 2. The high-level idea is that the group administrator designates a set of initial members and hands them some private credentials by which they can generate invitations for their intended invitees. Members are allowed to generate arbitrary many invitations. However, it is guaranteed that no inviter can generate more than one valid invitation for a particular invitee. The invitee must collect t invitations from t distinct members. Next, the invitee aggregates the invitations to a consolidated version in which any identifiable information about the individual inviters is removed. The final aggregate of invitations that is given to the administrator conveys no information regarding the individual inviters; hence inviter anonymity is preserved. The administrator can then verify the integrity of the invitations without knowing who the inviters exactly are.

Note that *Anonyma* preserves the inviter anonymity during the registration phase. However, it does not cover the case that the inviter and invitee may reveal their relation through their interaction inside the service. For instance, in the Google Inbox example, Bob can get invited to the service by Alice using an anonymous invitation-only registration (i.e., no one knows Alice has invited Bob). However, later on, Bob may exchange an email with Alice, which would imply a relationship between Alice and Bob. We emphasize that any interaction of this type that occurs after the registration phase must be considered out of the scope of this paper.

In summary, the contributions of *Anonyma* fall into two major categories namely, security and efficiency which are outlined below.

Security:

- **Inviter Anonymity:** No adversarial entity would be able to identify who is invited by whom. This property holds even if the group administrator colludes with a subset of inviters of an invitee and attempts to discover the identity of the other inviters of that invitee. Both group administrators and the inviters are active adversaries which may not follow the protocol specifications. The formal security definition and proof are supplied in Section 6.
- **Invitation Unforgeability:** An adversarial invitee cannot register to the system unless he has t many legitimate invitations. This property holds even if the invitee colludes with $t - 1$ inviters (clearly the registration of an invitee with t invitations is legitimate). The formal security definition and proof are supplied in Section 6. The provided definition also implies the following properties.
 - (1) **Invitation non-exchangability:** This property indicates that invitations issued for a particular invitee are not reusable for another user i.e., each invitation is tied to its intended invitee.
 - (2) **Preventing double invitation:** This feature indicates that an inviter cannot issue more than one valid invitation for a single invitee. This is essential since otherwise an invitee with insufficient inviters (i.e., $t' < t$) can obtain multiple invitations (e.g., $t - t'$) from one of her inviters and successfully register.

Note that in an invitation-based system, any collection of t members can decide to invite arbitrary many invitees. This is a legit property that complies with the invitation-only policy and should not count as a violation of invitation-unforgeability.

Efficiency:

- **Short invitations:** Each individual invitation, as well as the aggregated version, embodies only two group elements regardless of the threshold t and the current number of system members. Additionally, the complexity of communication among all the parties is of constant overhead.
- **Non-Interactive:** The invitation generation is a stand-alone non-interactive procedure that each inviter can run needless to the presence of other inviters and the group administrator. The same holds for the verification of the final aggregated invitation that shall be executed by the group administrator.
- **Light computations:** All the computations in *Anonyma* are light and efficient. An inviter has to carry out a constant number of group exponentiation for the invitation generation. This is in contrast to the related work [8, 10, 31, 34] where the inviter's computation overhead is of $O(N)$. The running time complexity of the inviter for aggregation of the t invitations is $O(t)$ whereas the group administrator verifies the final consolidated invitation in $O(1)$.
- **Scalability:** In *Anonyma*, the set of group members who are eligible to issue invitations (being inviters) is dynamic and the server can, efficiently and without re-keying the existing members, generate credentials for a new user to empower her to invite others. This is

significantly better than the related studies [4, 36] where the same action imposes $O(N)$ communication overhead.

Cross-network invitation-based system: Additionally, we propose *AnonymaX*, an anonymous cross-network invitation-based system, which empowers third-party authentication paradigm [2] (we note that *AnonymaX* extends InonymuX [7] to stand a malicious adversarial model). In this paradigm, mutually trusted domains can rely on each-others authentication mechanism. That is, if a user is authenticated by one domain, its authentication is accepted by all other domains that trust the authenticating domain. This transitive trust relationship is similarly sought by Microsoft Active Directory [1]. Likewise, in *AnonymaX*, users of different trusting administrative domains can be inviters of each other. In specific, a user can join one domain, e.g., Instagram, by obtaining invitations from members of another network, e.g., Twitter.

2 NOTATIONS AND PRELIMINARIES

Notation: We refer to a Probabilistic Polynomial-Time entity as PPT. TTP stands for Trusted Third Party. $x \in_R X$ and $x \leftarrow X$ both mean x is randomly selected from set X . \perp indicates an empty string. \equiv_c stands for computational indistinguishability. We use $DL_g(y)$ to indicate the discrete logarithm of y in base g .

Negligible Function: Function f is negligible if for $\forall p(\cdot)$ where $p(\cdot)$ is polynomial, there exists integer N s.t. for every $n > N$, $f(n) < \frac{1}{p(n)}$.

Pseudo-Random Generator [5]: A deterministic polynomial time function $P : \{0, 1\}^m \rightarrow \{0, 1\}^{l(m)}$ (where $l(\cdot)$ is a polynomial) is called Pseudo Random Generator (PRG) if $m < l(m)$ and for any probabilistic polynomial-time distinguisher D there exists a negligible function $negl(\cdot)$ such that:

$$|Pr[x \leftarrow \{0, 1\}^m : D(P(x)) = 1] - Pr[y \leftarrow \{0, 1\}^{l(m)} : D(y) = 1]| = negl(m) \quad (1)$$

(t,n)-Shamir Secret Sharing Scheme (SSS): The (t,n)-Shamir secret sharing scheme [42] is a tool by which one can split a secret value into n pieces such that any subset of t shares can reconstruct the secret. The scheme works based on polynomial evaluations. Let F_q be a finite field of order q . The secret holder/dealer picks a random polynomial f of degree $t - 1$ with coefficients from Z_q :

$$f(x) = \sum_{i=0}^{t-1} a_i \cdot x^i \quad (2)$$

The dealer sets the secret data S as the evaluation of that function at point 0 i.e., $f(0) = a_0 = S$. The share of each participant shall be one point on f e.g., $f(j)$ is the share of j^{th} shareholder. As such, a dealer can generate arbitrary many shares from its secret (i.e., by evaluating function f on a new point). Since each polynomial of degree $t - 1$ can be uniquely reconstructed by having t distinct points of that function, t Shamir shareholders are able to reconstruct the secret. Given any t shares $\{(i, s_i)\}_{i=1}^t$, the secret reconstruction algorithm works as below.

$$S = f(0) = \sum_{i=1}^t s_i \cdot B_i \quad (3)$$

where B_i s are Lagrange coefficients defined as

$$B_i = \sum_{j=1}^t s_j \prod_{j \neq i} \frac{j}{j-i} \pmod{q} \quad (4)$$

Shamir secret sharing scheme satisfies the following properties: 1) Given t or more than t shares, it can reconstruct the secret S easily; and 2) with knowledge of fewer than t shares, it cannot reconstruct the secret S . Shamir's scheme is *information theoretically secure* relying on no computational assumption.

Shamir shares are homomorphic under addition operation i.e., let $[s_1]$ and $[s_2]$ be shares of S_1 and S_2 (using (t, n) -Shamir secret sharing scheme), then $[s_1] + [s_2]$ constitutes a share of $S_1 + S_2$.

Multiplicative Homomorphic Encryption: A public key encryption scheme π consists of three algorithms key generation, encryption, and decryption, denoted by $\pi = (KeyGen, Enc, Dec)$. Using $KeyGen$, a pair of keys is generated called encryption key ek and decryption key dk . π is called multiplicative homomorphic encryption if for every a and b , $Enc_{ek}(a) \otimes Enc_{ek}(b) = Enc_{ek}(a \cdot b)$ where a and b belong to the encryption message space and \otimes is an operation over ciphertexts. As an example, in El Gamal encryption [18], \otimes corresponds to group multiplication. Additionally, we have $Enc_{ek}(a)^c = Enc_{ek}(a^c)$ where a is a plain message and c is any integer. Throughout the paper, we consider El Gamal scheme as our underlying encryption scheme.

Signature Scheme: A signature scheme [38] Sig consists of three algorithms key generation, sign and verify denoted by $Sig = (SGen, Sign, SVrfy)$. A pair of keys (sk, vk) is generated via $SGen$ where sk is the signature key and vk is the verification key. The signer signs a message m using sk by computing $\eta = Sign_{sk}(m)$. Given the verification key vk , a receiver of signature runs $SVrfy_{vk}(\eta, m)$ to verify.

A signature scheme $Sig = (SGen, Sign, SVrfy)$ is said to be existentially unforgeable under adaptive chosen message attack if \forall probabilistic polynomial time adversary A , there exists a negligible function $negl(\cdot)$ s.t. the following holds [26]:

$$\Pr[(sk, vk) \leftarrow SGen(1^\lambda); (m, \sigma) \leftarrow A^{Sign_{sk}(\cdot)}(vk) \text{ s.t. } m \notin Q \text{ and } SVrfy_{vk}(m, \sigma) = \text{accept}] = \text{negl}(\lambda) \quad (5)$$

$A^{Sign_{sk}(\cdot)}$ indicates that adversary has oracle access to the signature algorithm. Q indicates the set of adversary's queries to the signature oracle.

Zero-knowledge Proof of Knowledge of Discrete Logarithm (ZKPODL): This proof system was initially introduced by Schnorr [40] for proving the knowledge of a discrete logarithm in the group G of prime order q with generator g . That is, for a given $\omega, g \in G$, one can prove the knowledge of $x \in \mathbb{Z}_q$ s.t. $x = DL_g(\omega)$ (DL stands for discrete logarithm). Using the method featured by [24], we convert the Σ protocol of Schnorr to a zero-knowledge proof system. More information about Σ protocols and their conversion to a zero-knowledge proof system is provided in Section A.

Zero-Knowledge Proof of Plaintext Knowledge: This proof system is used to prove the plaintext knowledge of a given ciphertext. That is, given ciphertext C that is encrypted under public key pk , a prover proves the knowledge of x and r s.t. $C = Enc_{pk}(x, r)$. r is the randomness used while encryption. We instantiate such a proof system using the proposal of [23] for the El Gamal encryption scheme.

Zero-Knowledge Proof of Discrete Logarithm Equality: For a group G of prime order q and generators $g_1, g_2, h_1, h_2 \in G$, the ZKP of discrete logarithm equality is a protocol to prove that $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$ where $\alpha \in \mathbb{Z}_q$ [14].

Bilinear Map: Consider G_1 and G_2 as multiplicative groups of prime order q . Let g_1 be the generator of G_1 . We employ an efficiently computable bilinear map $e : G_1 \times G_1 \rightarrow G_2$ with the following properties [44]

- Bilinearity: $\forall u, v \in G_1$ and $\forall a, b \in \mathbb{Z}_q : e(u^a, v^b) = e(u, v)^{a \cdot b}$.
- Non-degeneracy: $e(g_1, g_1) \neq 1$.

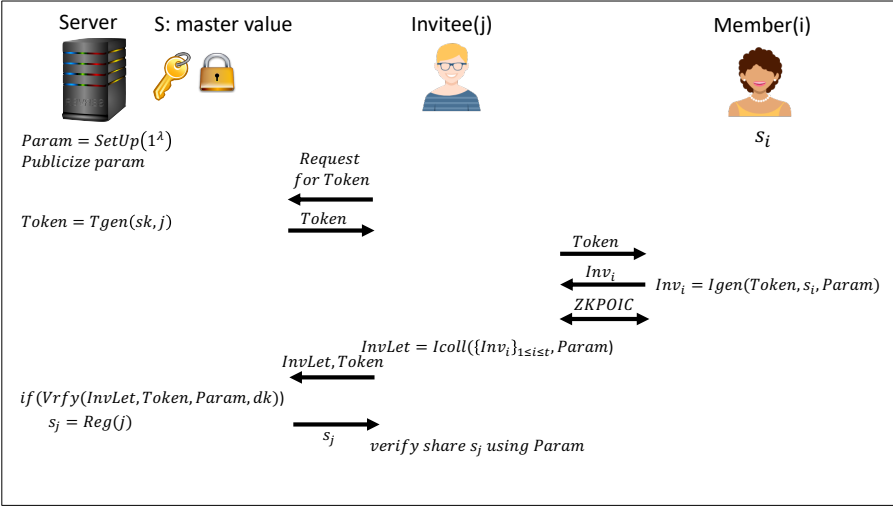


Fig. 3. Anonyma workflow.

We adopted Type 1 pairing (symmetric) in the favor of simple protocol description and security analysis, nevertheless, this can be translated to an asymmetric pairing type to empower more efficient implementation [13].

Computational Diffie-Hellman Assumption [17]: Given a cyclic group G of prime order q with a generator g , and two randomly selected group elements $h_1 = g^{r_1}, h_2 = g^{r_2}$, the Computational Diffie-Hellman (CDH) assumption is hard relative to G if for every PPT adversary A there exists a negligible function $negl(\lambda)$ where λ is the security parameter, such that:

$$\Pr[A(G, q, g, h_1, h_2) = g^{r_1 \cdot r_2}] = negl(\lambda)$$

3 CONSTRUCTION

Anonyma consists of the following algorithms: *Setup*, *Token generation (Tgen)*, *Invitation generation (Igen)*, *Invitation collection (Icoll)*, *Invitation Verification (Ivrfy)* and *Registration (Reg)*. The summary of each algorithm is explained in Section 3.1 followed by the full construction in Section 3.2. Throughout the paper, we assume that all the parties communicate via secure and authenticated channels. The general interaction between the parties is illustrated in Figure 3.

3.1 Construction Overview

- *Setup*: The server invokes the *Setup* algorithm with the input of the security parameter 1^λ to initialize the system parameters: a cyclic group G , a master value $S \in_R G$, as well as key pairs for a signature scheme (denoted by sk, vk) and ElGamal encryption (denoted by ek, dk). At the beginning of the system lifetime, the server needs to register at least t initial users so that they can start inviting others. These initial members are given credentials by the server to be able to make invitations. Each credential is indeed a share s_i of the server's master value S that is generated using (t, n) -SSS scheme. For the shares to be verifiable (the member can verify whether or not his piece is valid), the server publishes the commitment to the polynomial function generated as part of the SSS scheme. A newcomer can register to the system if she manages to compute a designated function of the master value S . Given that S can be reconstructed only with the combination of t valid shares i.e., the presence of t

shareholders, the newcomer cannot register unless with having t distinct inviters. As such, invitation unforgeability is guaranteed. Note that any group of t members can invite arbitrary many invitees to the system. However, this ability should not be mistaken for having control of the system compromised by its members. The invitation-only policy is a method to ensure the growth of the system based on members' trust but does not mean to prevent growth.

- *Tgen*: Each newcomer (i.e., invitee) contacts the server to get a token. The server runs the *Tgen* algorithm to generate a *Token* and hands it to the invitee. The *Token* is a server signed certificate that embodies the index of the newcomer (each user is associated with a unique index) as well as a random element from the group G . Tokens shall be used by the inviters to invite their intended invitee. Invitations issued for a particular token cannot be used for another token. This way, the non-exchangeability of the invitations is guaranteed.
- *Igen*: The Invitee contacts each of his t inviters (this communication cannot be observed by the server/administrator) and communicates his *Token* with them. Provided a valid *Token*, each inviter generates an invitation by executing *Igen*. The invitation consists of two parts: 1) a masked version of the inviter's share s_i , 2) and the masking value encrypted using the server's ek . The *Token* is integrated into both parts of the invitation. As a part of *Igen*, the inviter has to prove in zero-knowledge that his invitation is well structured. For this sake, we devise a zero-knowledge proof protocol (i.e., Zero-Knowledge Proof of Invitation Correctness (ZKPOIC)). This proof helps in protecting between-inviter anonymity, i.e., inviters who collude with the server do not learn the identity of other inviters. Next, the inviter hands his invitation Inv_i to the invitee.
- *Icoll*: Upon the receipt of t invitations $\{Inv_i\}_{i=1}^t$, the invitee invokes the *Icoll* algorithm through which he aggregates and blinds the invitations into a unified invitation letter *InvLet*. Aggregation and blinding remove any identifiable information about the identity of the inviters and helps in providing inviter anonymity (especially against a corrupted server). Additionally, through aggregation, the masking version of the master value S gets homomorphically reconstructed. We utilize the homomorphic property of both Shamir shares and the ElGamal encryption scheme to enable aggregation. At last, the invitee submits the final invitation letter *InvLet* together with his *Token* to the server.
- *Ivrfy*: The server authenticates the invitation letter by running *Ivrfy* and accepts or rejects accordingly. In a nutshell, the *InvLet* is valid if and only if it contains the master value S .
- *Reg*: If the verification passes successfully (i.e., *Ivrfy* outputs accept), the server runs the *Reg* algorithm to issue credentials for the newcomer to enable him to act as an inviter. This credential is a Shamir share of the server's master value S . The newcomer verifies the validity of his share using the parameters output by the server in the *SetUp* phase and then stores his share for inviting others.

3.2 Full Construction

3.2.1 *SetUp*: This algorithm is run by the server who inputs the security parameter 1^λ and generates system parameters *Param* as follows.

- Two primes p and q of length λ such that $q|p - 1$.
- g is a generator of a cyclic subgroup G of order q in Z_p^* .
- ElGamal encryption scheme $\pi = (EGen, Enc, Dec)$ with the key pair $(ek = h = g^a, dk = a)$ denoting encryption key and decryption key, respectively. dk remains at the server while ek is published.
- A signature scheme $Sig = (SGen, Sign, SVrfy)$. The signature and verification keys (sk, vk) are generated according to *SGen*. vk is published.

- A pseudo random generator $PRG:\{0, 1\}^\lambda \rightarrow Z_q$.
- A master value $S \leftarrow Z_q$.
- A randomly chosen polynomial function $f(y) = a_{t-1}y^{t-1} + \dots + a_1y + a_0$ of degree $t - 1$ whose coefficients a_1, \dots, a_{t-1} belong to Z_q and $a_0 = S$.
- The server initially registers t users into the system so that they can start inviting outsiders. Each user is associated with a unique index i and shall receive the evaluation of function f on that index, i.e. $s_i = f(i)$. We refer to s_i as the master share of the i^{th} user.
- The server publishes $F_0 = g^{a_0}, F_1 = g^{a_1}, \dots, F_{t-1} = g^{a_{t-1}}$ as the commitment to the selected function f . Given F_0, \dots, F_{t-1} , the computation of commitment on $f(i)$ for any i is immediate as given in Equation 6. We will use γ_i to indicate g^{s_i} .

$$\gamma_i = \prod_{j=0}^{t-1} F_j^{i^j} = g^{a_0} \cdot g^{a_1 \cdot i} \dots g^{a_{t-1} \cdot i^{t-1}} = g^{a_0 + a_1 \cdot i + \dots + a_{t-1} \cdot i^{t-1}} = g^{f(i)} = g^{s_i} \quad (6)$$

- The server publishes $Param = (G, p, q, g, ek, vk, (F_0, \dots, F_{t-1}))$.

3.2.2 Token Generation: Users wishing to register to the system first need to contact the server and obtain a token. The server generates a token through the token generation algorithm shown in Algorithm 3.1. In this procedure, the server initially assigns the user a unique index j . Indices can simply be assigned based on the arrival order of users, as long as no two users are assigned the same index. Hence, the j^{th} coming user receives the index value of j . Next, the server generates a random group element ω (lines 1-2) and certifies $j||\omega$ using his signing key sk (line 3). Let η be the signature outcome. The tuple (η, j, ω) constitutes the Token (line 4). We remark that the server is not required to record any information regarding the issued tokens. Thus, the generated tokens can simply be discarded and only the last value of j (the number of token requests) needs to be remembered. Therefore, we do *not* incur any storage load on the server per token.

Algorithm 3.1: Tgen [Server]

Input: sk, j

Output: *Token*

- 1 $r \leftarrow Z_q$
 - 2 $\omega = g^r$
 - 3 $\eta = Sign_{sk}(j||\omega)$
 - 4 $Token = (\eta, j, \omega)$
-

3.2.3 Invitation Generation: Invitation generation is run by the inviter to generate an invitation for a token given by the invitee. The procedure is shown in Algorithm 3.2. We assume that invitee and inviter communicate out of band (cannot be observed by the server/administrator), e.g., using a messaging application. Firstly, the inviter checks the authenticity of the token against the server verification key vk (line 1). Then, he samples a random value δ_i from Z_q by applying PRG on the random seed v (lines 2-3). Then, he blinds his master share using δ_i , i.e., $s_i + \delta_i$, and then ties this value to the provided token as $\tau_i = \omega^{s_i + \delta_i}$ (line 4). He also encrypts the masking value ω^{δ_i} as $e\delta_i$ using the server's encryption ek (line 5). To ensure that the inviter is acting honestly (i.e., generating the invitation as instructed in the algorithm), the inviter must prove the correctness of the invitation in zero-knowledge. To enable this, we propose a zero-knowledge proof system for the Proof Of Invitation Correctness *ZKPOIC*. The inviter and invitee engage in *ZKPOIC* (line 7) through which the inviter proves the correctness of his invitation $Inv_i = (\tau_i, e\delta_i)$ to the invitee

in zero-knowledge. In the following, we explain our proposed proof system. We first draw a Σ protocol for *POIC* and prove its security. Then, the zero-knowledge variant is immediate using the method proposed in [24, 37].

Algorithm 3.2: Igen [Inviter]

Input: $Token = (\eta, j, \omega), s_i, Param$

Output: Inv_i/\perp

```

1 if  $Svrf_{yvk}(\eta, j || \omega) = accept$  then
2    $v \leftarrow \{0, 1\}^\lambda$ 
3    $\delta_i = PRG(v)$ 
4    $\tau_i = \omega^{s_i + \delta_i}$ 
5    $e\delta_i = Enc_{ek}(\omega^{\delta_i})$ 
6    $Inv_i = (\tau_i, e\delta_i)$ 
7   return  $inv_i$  // Inviter authenticates  $Inv_i$  through ZKPOIC
8 return  $\perp$ 

```

Σ **Protocol for Proof Of Invitation Correctness (POIC):** The invitation is constructed correctly if the inviter proves the following statements:

- (1) The inviter possesses a valid share of the master value S . That is, the inviter holding index i must prove the knowledge of the discrete log of γ_i , i.e., s_i . Note that $\gamma_i = g^{s_i}$ can be computed from F_0, \dots, F_{t-1} as explained in Equation 6.
- (2) The inviter knows the plaintext of $e\delta_i$, i.e., the knowledge of ω^{δ_i} and r such that $e\delta_i = (e\delta_{i,1} = \omega^{\delta_i} \cdot h^r, e\delta_{i,2} = g^r)$.
- (3) The randomness δ_i used in the creation of τ_i is correctly encrypted in $e\delta_i$. This can be captured by proving that $\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r (= \omega^{s_i})$ and $\gamma_i = g^{s_i}$ have the same discrete logarithm s_i . The former is true due to Equation 7.

$$\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r = \omega^{\delta_i + s_i} \cdot \omega^{-\delta_i} \cdot h^{-r} \cdot h^r = \omega^{s_i} \quad (7)$$

To enable zero-knowledge proof of the aforementioned statements, we devise a Σ -protocol (P, V) as depicted in Figure 4. We refer to this proof system by **Proof of Invitation Correctness**, or *POIC*. *POIC* captures the relation R indicated in Equation 8 which embodies four different predicates.

$$R = \{ \overbrace{((\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega))}^{\text{Public inputs}}, \overbrace{(s_i, r, \delta_i)}^{\text{Private inputs}} \} \quad (8)$$

$$DL_g(\gamma_i) = s_i \quad \wedge \quad (9)$$

$$e\delta_{i,1} = \omega^{\delta_i} \cdot h^r \quad \wedge \quad (10)$$

$$DL_g(e\delta_{i,2}) = r \quad \wedge \quad (11)$$

$$DL_\omega(\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r) = DL_g(\gamma_i) = s_i \quad (12)$$

For the proof of relation R in Equation 8, we incorporate the Shnorr protocol [26] for the proof of discrete logarithm knowledge (Equations 9 and 11), proof of plaintext knowledge as proposed in [23] (Equation 10), and the proof of discrete logarithm equality [14] (Equation 12).

Completeness: To prove that completeness holds, observe that if the prover P follows the protocol honestly, then due to the Equations 13, 14, 15, and 16, the verifier V accepts.

$$A \cdot \gamma_i^e = g^{s'} \cdot (g^{s_i})^e = g^{s' + e \cdot s_i} = g^{Z_1} \quad (13)$$

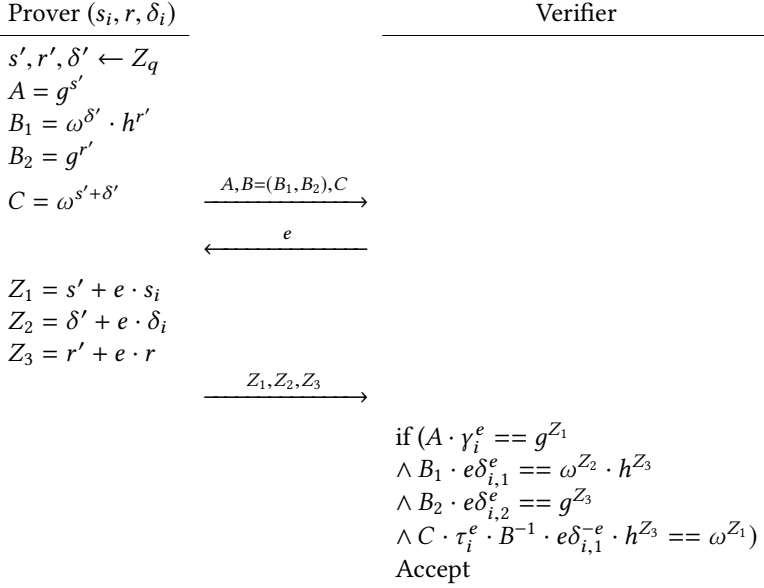


Fig. 4. Σ protocol of Proof of Invitation Correctness for the common input $\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega$. The prover has the private input (s_i, r, δ_i) .

$$B_1 \cdot e\delta_{i,1}^e = (\omega^{\delta'} \cdot h^{r'}) \cdot (\omega^\delta \cdot h^r)^e = \omega^{\delta'+e\cdot\delta_i} \cdot h^{r'+e\cdot r} = \omega^{Z_2} \cdot h^{Z_3} \quad (14)$$

$$B_2 \cdot e\delta_{i,2}^e = (g^{r'}) \cdot (g^r)^e = g^{r'+e\cdot r} = g^{Z_3} \quad (15)$$

$$\begin{aligned} C \cdot \tau_i^e \cdot B^{-1} \cdot e\delta_{i,1}^{-e} \cdot h^{Z_3} &= \\ (\omega^{s'+\delta'}) \cdot (\omega^{e\cdot s_i + e\cdot\delta_i}) \cdot (\omega^{-\delta'} \cdot h^{-r'}) \cdot (\omega^{-e\cdot\delta_i} \cdot h^{-e\cdot r}) \cdot h^{r'+e\cdot r} &= \\ \omega^{(s'+e\cdot s_i)} = \omega^{Z_1} & \end{aligned} \quad (16)$$

We prove the special soundness and special honest verifier zero knowledge properties in Section B. We additionally present the security properties of a zero-knowledge proof system for POIC that is achieved using the method given in [24, 37].

3.2.4 Invitation Collection: Upon receipt of t invitations, the invitee runs the Invitation Collection (*Icoll*) procedure as indicated in Algorithm 3.3. The invitee aggregates τ_i values as $\prod_{i=1}^t \tau_i^{B_i}$ (line 3). He operates similarly for $e\delta_i$ values as $\prod_{i=1}^t e\delta_i^{B_i}$ (line 4). B_i s are the Lagrange coefficients (as defined in Equation 4) used for the reconstruction of the master value S from the Shamir shares. Next, the invitee randomizes both aggregates T and $e\Delta$ by adding a random value of his own choice, i.e., δ^* . The randomization cancels out the effect of the Lagrange coefficients and makes the final aggregates, i.e., T and $e\Delta$, independent of the B_i values. Recall that the Lagrange coefficients are dependent on the inviters' indices and by hiding them we aim at protecting inviter anonymity. The final invitation letter *InvLet* shall be the pair $(T, e\Delta)$. The invitee submits the invitation letter and the token to the server.

In Equations 17 and 18, we expand the result of T and $e\Delta$, which leads to the following observations.

$$T = \omega^{\delta^*} \cdot \prod_{i=1}^t \tau_i^{B_i} = \omega^{\delta^*} \cdot \prod_{i=1}^t \omega^{B_i \cdot s_i + B_i \cdot \delta_i} = \omega^{\delta^* + \sum_{i=1}^t B_i \cdot s_i + \sum_{i=1}^t B_i \cdot \delta_i} \quad (17)$$

$$= \omega^{S + \delta^* + \sum_{i=1}^t B_i \cdot \delta_i} = \omega^{S + \Delta}$$

$$e\Delta = \text{Enc}_{ek}(\omega^{\delta^*}) \cdot \prod_{i=1}^t e\delta_i^{B_i} = \text{Enc}_{ek}(\omega^{\delta^*}) \cdot \prod_{i=1}^t \text{Enc}_{ek}(\omega^{B_i \cdot \delta_i}) \quad (18)$$

$$= \text{Enc}_{ek}(\omega^{\delta^* + \sum_{i=1}^t B_i \cdot \delta_i}) = \text{Enc}_{ek}(\omega^{\Delta})$$

The first observation is that T has the master value S embedded in its exponent. Intuitively, the presence of S in the exponent is a proof that the invitee has t distinct invitations. Since otherwise, the reconstruction of S would be impossible (we elaborate on this in Section 6 and formally prove the unforgability of invitations). Another observation is that the computation of both T and $e\Delta$ depends on the token ω . Hence, as desired, the resultant *InvLet* is now bound to the given token. This would help for the non-exchangeability of the invitations. At last, T contains a masked version of master value, i.e., $S + \Delta$, in the exponent whereas $e\Delta$ embodies the corresponding masking value Δ . The encryption $e\Delta$ of the masking value shall be used at the server for the verification purpose (see invitation verification below).

Algorithm 3.3: Icoll [Invitee]

Input: $\{Inv_i = (\tau_i, e\delta_i) | 1 \leq i \leq t\}$, *Param*

Output: *InvLet*

- 1 $r \leftarrow \{0, 1\}^\lambda$
 - 2 $\delta^* = \text{PRG}(r)$
 - 3 $T = \omega^{\delta^*} \cdot \prod_{i=1}^t \tau_i^{B_i}$
 - 4 $e\Delta = \text{Enc}_{ek}(\omega^{\delta^*}) \cdot \prod_{i=1}^t e\delta_i^{B_i}$
 - 5 $InvLet = (T, e\Delta)$
-

Invitation Verification: Once the invitee hands his invitation letter *InvLet* together with the corresponding token *Token* to the server, the server executes the invitation verification procedure shown in Algorithm 3.4. As the first step, the server authenticates the *Token*, i.e., whether it is signed under the server's signature key sk (line 1). Next, the validity of the invitation letter *InvLet* must be checked. For that, the server decrypts $e\Delta$ using its decryption key dk and obtains ω^Δ (line 2). Recall that Δ was used to mask the master value S in $T = \omega^{S + \Delta}$. Thus, if T and $e\Delta$ are constructed correctly, we expect that $\omega^S \cdot \omega^\Delta = T$ (line 3). If all the verification steps are passed successfully, then the server accepts the user's membership request.

Registration: The server invokes the registration procedure (Algorithm 3.5) for users who pass the verification phase (Algorithm 3.4). The input to Algorithm 3.5 is the index j of the newcomer, and the output is a Shamir share Φ_j of the master value S , where Φ_j is the evaluation of polynomial f at point j (line 1). Note that the index j is the index included in the user's *Token* = (η, j, ω) . The server delivers Φ_j to the user who can then start making invitations as an inviter. The user authenticates his share by comparing the commitment γ_j (as given in Equation 6) against its own share, i.e., g^{Φ_j} . If they are equal, the user accepts and stores the share.

Algorithm 3.4: Ivrfy [Server]

Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega), Param, dk$ **Output:** *reject/accept*

```

1 if  $Svrf_{yvk}(\eta, j || \omega) = \text{accept}$  then
2    $\omega^\Delta = Dec_{dk}(e\Delta)$ 
3   if  $\omega^S \cdot \omega^\Delta = T$  then
4     return accept

```

Algorithm 3.5: Reg [Server]

Input: j **Output:** Φ_j

```

1  $\Phi_j = f(j)$ 

```

4 ANONYMAX: ANONYMOUS CROSS-NETWORK INVITATION-BASED SYSTEM

AnonymaX is an anonymous cross-network invitation-based system, which enables third-party authentication [2]. That is, mutually trusted domains can rely on each-others authentication mechanism. That is, if a user is authenticated by one domain, its authentication is accepted by all other domains that trust the authenticating domain. Users of different trusting administrative domains can be inviters of each others. In specific, a user can join one domain, e.g., Twitter, by obtaining invitations from members of another network, e.g., Facebook. Facebook and Twitter in the preceding examples are called *authenticating* domain Υ and *registration* domain Φ , respectively.

Failed Approaches: One simple but cumbersome solution to empower a cross-network invitation-based system is to follow the regular invitation-based system, i.e. each time a user wants to join the *registration* domain, the *authenticating* server authenticates that particular user and communicates the authentication result to the *registration* server. However, this solution requires the two servers to keep in contact with each other and imposes unnecessary overhead on the *authenticating* server.

An alternative approach proposed by Inonymous [7] (our prior work), is that the *authenticating* server would publish the commitment over the master value S as g^S to the *registration* servers. Subsequently, *registration* servers would follow a different verification method (relying on bilinear maps) to authenticate invitations on their own. While this solution works for the honest but curious adversarial model, it fails in providing invitation unforgeability against a malicious adversary, which is explained next. Consider *registration*₁ and *registration*₂ as two *registration* servers. The corrupted *registration*₁ wants to join *registration*₂ as an invitee without enough inviters. *registration*₁ receives a token with the random value ω from *registration*₂ and then issues the same token to the users who want to join its service. As such, *registration*₁ can reuse the invitation letters of his users to craft a valid invitation letter to join *registration*₂. We address this issue in *AnonymaX* by making the *registration* servers prove in zero-knowledge (using an interactive proof) that they know the discrete logarithm $DL(w)$ of their issued tokens during the *Tgen* protocol. As such, no *registration* server can issue tokens that are not generated by itself.

***AnonymaX* Overview:** The server of *authenticating* network, denoted by Υ , publishes g^S as a part of its set of parameters $Param_{authenticating}$ (S is the master value S). Note that the description of group G is only generated by the *authenticating* server Υ and is used by other *registration* servers. On the other side, the *registration* networks denoted by Φ_j , $1 \leq j \leq N$, announce their $Param_{\Phi_j}$ to

be the pair of encryption keys ek_{Φ_j} and signature verification keys vk_{Φ_j} , i.e. $Param_{\Phi_j} = (ek_{\Phi_j}, vk_{\Phi_j})$. The corresponding decryption key dk_{Φ_j} as well as the signature signing key sk_{Φ_j} remain private at the server-side. Each invitee willing to join Φ_j shall obtain a token from Φ_j . During the token generation, the *registration* server follows Algorithm 3.1 and additionally must prove in zero-knowledge that it knows the discrete logarithm of the ω embodied in the token. This is to protect inviter anonymity against the corrupted *registration* server. As such, after the issuance of a token, the *registration* server runs an instance of ZKPODL protocol (given in section 2) with the invitee. The modified procedure is provided in Algorithm 4.1.

Algorithm 4.1: XTgen [*registration* Server Φ_j]

Input: $Param_{quest} = (G, q, g, ek_{quest}, vk_{quest}, (F_0, \dots, F_{t-1})), sk_{\Phi_j}, i$
Output: *Token*

```

1  $r \leftarrow Z_q$ 
2  $\omega = g^r$ 
3  $\eta = Sign_{sk_{\Phi_j}}(i || \omega)$ 
4  $Token = (\eta, i, \omega)$ 
5 Run ZKPODL( $(G, q, g, \omega), r$ )

```

Upon successful proof, the invitee accepts the token. The invitee needs to collect invitations from the members of the *authenticating* network Υ to be used in the registration of a particular *registration* network Φ_j . Inviters issue invitation as in the regular invitation procedure given in Algorithm 3.2. However, the inviters should verify the tokens against the verification key of the issuing server i.e., Φ_j . Also, the inviters shall use the encryption key of the *registration* network ek_{Φ_j} to encrypt their masking values. As such, in Algorithms 3.2 and 3.3, the inviter uses ek_{Φ_j} and vk_{Φ_j} , i.e. $Param_{\Phi_j}$ as input. Therefore, the invitation letters received by the Φ_j are of the form $InvLet = (T, e\Delta)$ where $e\Delta$ is an encrypted masking value under ek_{Φ_j} . The *registration* server Φ_j runs a different verification routine, which is given in Algorithm 4.2. We assume the existence of a bilinear map $e: G \times G \rightarrow G_2$ where G and G_2 are multiplicative groups of prime order q . The only difference between Algorithm 4.2 and Algorithm 3.4 is at the second verification step i.e., line 3. The correctness holds by the bilinearity of the bilinear map e , as in Equation 19.

$$\begin{aligned}
 e(\omega, g^S) \cdot e(\omega^\Delta, g) &= e(\omega, g)^S \cdot e(\omega, g)^\Delta = e(\omega, g)^{S+\Delta} = e(\omega^{S+\Delta}, g) \\
 &= e(T, g)
 \end{aligned} \tag{19}$$

Algorithm 4.2: XIVerify [*registration* Server Φ_j]

Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega), Param_{authenticating}, Param_{\Phi_j}, dk_{\Phi_j}$
Output: *reject/accept*

```

1 if  $Srvfy_{vk_{\Phi_j}}(\eta, j || \omega) = \text{accept}$  then
2    $\omega^\Delta = Dec_{dk_{\Phi_j}}(e\Delta)$ 
3   if  $e(\omega, g^S) \cdot e(\omega^\Delta, g) = e(T, g)$  then
4     return accept

```

	SetUp	Tgen	Igen	Ivrfy	Reg
Server	1.6 s	4.24 ms	-	6.5 ms	0.08 ms
Inviter	-	-	27.5 ms	-	-

Table 1. Running time of the server and the inviter. (s: seconds, ms: milliseconds)

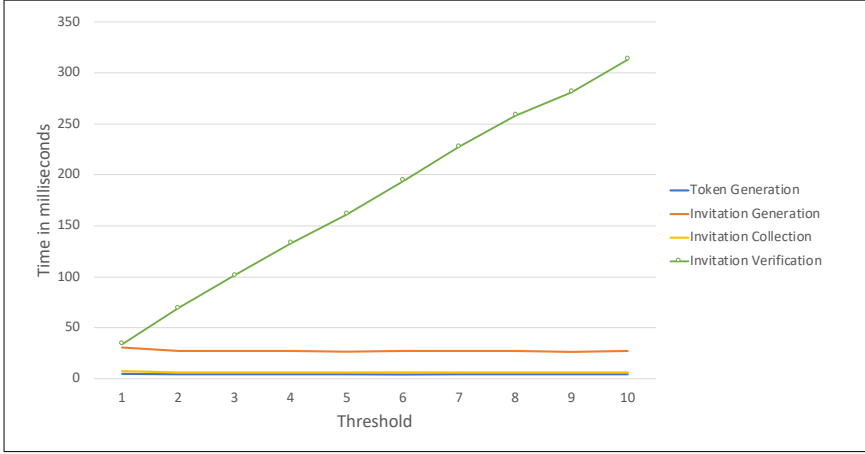


Fig. 5. The invitee's running time.

5 PERFORMANCE

5.1 Running Time

In this section, we analyze the running time of each algorithm of *Anonyma*.

Simulation setting: The running time is measured on a standard laptop with 8 GB 1600 MHz DDR3 memory and 1.6 GHz Intel Core i5 CPU. The simulation setup consists of 100 registered members and 100 invitees. Each invitee collects threshold many invitations from randomly chosen inviters, i.e., the 100 initially registered members. The running time of all the algorithms is recorded over threshold values 1-10. The DSA signature scheme [28] is instantiated with the key length of 1024 bits.

Under the aforementioned setting, the running time of the parties are as follows, and the results are summarized in Table 1 and Figure 5.

Server: The server spends 1.6 seconds in order to run the *SetUp* phase. This phase should be executed only once for the entire system lifetime. The *Token Generation* algorithm requires 4.24 milliseconds. The *Invitation Verification* procedure incurs 6.5 milliseconds. The *Registration* of each newcomer requires 0.08 milliseconds.

Invitee: The invitee performs *Invitation Collection (Icoll)* procedure, whose running time is linearly dependent on the number of required invitations, i.e., t . In particular, the running time of *Icoll* can be formulated as $t \cdot t_{auth} + (t - 1) \cdot t_{agg}$, where t is the threshold, t_{auth} the time to authenticate each invitation, and t_{agg} is the time required for homomorphic aggregation of two individual invitations. The experimental results as illustrated in Figure 5 also attest the same relation.

Inviter: The inviter is only involved in the execution of the *Invitation Generation* algorithm for which he spends 27.5 milliseconds.

5.2 Communication Complexity

The communication complexity signifies the amount of data exchanged among the parties. *Anonyma* features an asymptotic communication complexity that is constant in the security parameter. The concrete values are also measured as the number of exchanged bits and are as follows. In *Tgen* protocol, the transmission of *Token* (two group elements of size 2048 bits) incurs 4096 bits= 0.512 KB. For the invitation generation protocol *Igen*, the inviter exchanges the invitation (consisting of 3 group elements) together with the ZKPOIC (with 8 group elements) which on aggregate yields 22528 bits (= 2.81 KB) data transfer. The registration protocol involves the invitee submitting the invitation letter (with 3 group elements) together with the token (of size 4096 bits) to the server which results in 10240 bits (= 1.28 KB) data exchange.

5.3 Storage

The storage requirement of each entity is measured based on the number of bits that the party needs to persistently retain locally. The server holds two pairs of keys for signature and encryption, hence requires 20896 bits (2.612 KB) of storage. Moreover, the server saves the description of the polynomial of degree $t - 1$ with t coefficients and their corresponding commitments which approximately results in $2t \cdot 2048$ bits of storage requirement at the server. The invitee only needs to keep its share of the master value which is of size 2046 bits (0.25 KB). For the invitee, no local storage is required.

6 SECURITY

In this section, we provide security definitions for inviter anonymity and invitation unforgeability, and then prove the security of *Anonyma* (Sections 6.1 and 6.2). In Section 6.3, we prove the security of *AnonymaX* for which we supply a new security definition capturing invitation unforgeability in the cross-network invitation based systems. The formal proof of special soundness and honest verifier zero-knowledge property of our proposed Σ protocol for proof of invitation correctness (POIC), as well as the ideal functionality F_{POIC}^R corresponding to the zero-knowledge version of POIC are provided in the Appendix, section B.

6.1 Inviter Anonymity

An invitation-based system protects inviter anonymity if an invitee with t inviters can authenticate himself to the server without disclosing the identity of his inviters to the server. In the extreme situation where a corrupted server also manages to control $t - 1$ inviters of an invitee, the inviter anonymity should guarantee that the identity of the remaining non-colluding inviter remains protected against the server and other inviters. The coalition of the server and $t - 1$ inviters is the most powerful adversary against inviter anonymity. In the following, we present the formal definition of inviter anonymity as well as a formal security proof of inviter anonymity of *Anonyma*.

6.1.1 Security Definition: We model inviter anonymity as a game denoted by $InvAnonym_A(\lambda)$ played between a challenger and an adversary. The challenger acts as the invitee as well as the uncorrupted members of the system. On the other hand, the adversary plays as the server as well as arbitrary many corrupted members. The challenger is to register the invitee into the system while $t - 1$ inviters of the invitee are controlled by the adversary and the remaining inviter is under the control of the challenger. As such, the adversary issues $t - 1$ invitations on behalf of the corrupted inviters. Then, the adversary selects two indexes u_0, u_1 corresponding to two uncorrupted members. The challenger selects one of them randomly as u_b , where $b \in \{0, 1\}$, to be the other inviter. The challenger issues an invitation from u_b for the invitee and combines it with the $t - 1$ invitations issued by the adversary. The final invitation letter is submitted to the adversary (who also plays

the role of the server). The challenge of the adversary is to guess a bit b indicating the index of the uncorrupted inviter. If the adversary cannot guess that index with more than a negligible advantage, then the system provides inviter anonymity. The formal definition follows.

Inviter Anonymity Experiment $InvAnonym_A(\lambda)$

- (1) The adversary is given the security parameter 1^λ . It acts as the server and hands over $Param$ to the challenger.
- (2) The adversary registers arbitrary many users to the system. The adversary instructs the challenger to register honest users through the Reg protocol. U_h and U_c contain the indices of the honest and corrupted members, respectively.
- (3)(a) The adversary outputs the index of two honest inviters $u_0, u_1 \in U_h$.
 - (b) The adversary, acting as the server, generates a token $Token$ for the invitee with index $j^* \in U_h$.
 - (c) The adversary specifies a set of $t - 1$ indices $I_c \subset U_c$ to be the corrupted inviters. For every $i \in I_c$, the adversary engages with the challenger in the execution of the $Igen$ protocol using $Token$ as the input. As the result, the invitee (i.e., the challenger) obtains a set of $t - 1$ invitations denoted by $\{Inv_i\}_{i \in I_c}$.
- (4)(a) The challenger selects a bit value $b \leftarrow \{0, 1\}$. The challenger runs the $Igen$ protocol over $Token$ to issue an invitation from u_b for the invitee. Let Inv_b be the result.
 - (b) The challenger runs $Icoll$ using $\{Inv_i\}_{i \in I_c} \cup Inv_b$ and $Param$ and generates an invitation letter $InvLet$. The challenger attempts to register to the system by sending $InvLet$ to the adversary.
- (5) The adversary guesses a bit b' .
- (6) The output of the game is 1 if $b == b'$, 0 otherwise.

Definition 6.1. An invitation-based system has inviter anonymity if for every probabilistic polynomial time adversary A there exists a negligible function $negl(\cdot)$ such that:

$$Pr[InvAnonym_A(\lambda) = 1] = \frac{1}{2} + negl(\lambda)$$

At a high level, in *Anonyma*, the anonymity of the inviter holds due to the soundness of the proposed ZKPOIC (zero-knowledge proof of invitation correctness) and the security of the pseudo-random number generator (i.e., PRG). Below, to give an insight into how ZKPOIC can protect inviter anonymity, we draw a situation where the lack of ZKPOIC would immediately break inviter anonymity. Then, by relying on the F_{POIC}^R hybrid model for our proof, we relate the inviter anonymity of *Anonyma* to the security of the deployed PRG.

Recall that, as defined in the game, the adversary controls the server and $t - 1$ inviters of the invitee. Due to the employed ZKPOIC, the invitee is assured that the inviters are not able to deviate from the protocol descriptions and hence would have to use their real master shares for the invitation generation. This implies that the master value S shall be reconstructed correctly as the output of $Icoll$. Therefore, as the result of the registration of the invitee (step 4.b from $InvAnonym_A(\lambda)$ experiment), the server obtains $InvLet = (T = \omega^{S+\Delta}, e\Delta)$ out of which the adversary can learn S and Δ . According to the Shamir secret sharing scheme, although the adversary knows $t - 1$ shares that are used for the reconstruction of S , the remaining contributing shareholder can be any of the existing members, and hence the inviter anonymity is guaranteed. Now, consider that the inviters are not required to prove the correctness of their invitations. The $t - 1$ corrupted inviters use zeros instead of their real master shares for invitation generation, i.e., $s_i = 0$ for $i \in I_c$. Then, the server

obtains $w^{S'}$ with the following value: $S' = \sum_{i \in I_c} s_i \cdot B_i + s_{u_b} \cdot B_{u_b} = s_{u_b} \cdot B_{u_b}$. The adversary can simply try the combinations of master shares s_{u_0} and s_{u_1} with B_0 and B_1 , respectively, and figure out the remaining inviter's index (in practice, the possible number of values is linear in the number of non-colluding inviters, which is the number of registered users). This trivially breaks inviter anonymity, which follows from the lack of ZKPOIC.

As we discussed before, due to ZKPOIC all the invitations issued for the invitee are guaranteed to be well-structured (and their correctness are proven during *Igen*). Thus, the execution of *Icoll* by the invitee would lead to a valid invitation letter of the form $InvLet = (\omega^{S+\Delta}, e\Delta)$ where S is the server's master value, $e\Delta$ is the encryption of ω^Δ and $\Delta = \delta^* + \sum_{i \in I_c} B_i \cdot \delta_i + B_{u_b} \cdot \delta'$ (δ^* is the masking value added by the invitee, δ' is the non-colluding inviter's masking value resulted from a PRG and B_{u_b} is the Lagrange coefficient computed based on the index of the non-colluding inviter). The adversary may get some idea about the identity of the non-colluding inviter by extracting the Lagrange coefficients from the Δ value (Lagrange coefficients are the function of inviters' indices). Two cases may occur. If the random values δ' and δ^* are selected truly at random, then we know that Δ is also a random value and conveys nothing about the Lagrange coefficient B_{u_b} . Though, if δ' and δ^* are the output of a PRG then the adversary may have advantages to extract the Lagrange coefficients. We denote the adversary's advantage by ϵ . If ϵ is non-negligible, it implies that we can distinguish between a PRG and a random number generator hence we break the security of the PRG. In the following, we provide formal proof.

THEOREM 6.2. *Anonyma provides inviter anonymity in F_{POIC}^R hybrid model (as defined in Equation 38), assuming that PRG is a secure pseudo-random number generator.*

Proof: We reduce the inviter anonymity of *Anonyma* to the security of the employed PRG. If there exists a PPT adversary A who breaks the inviter anonymity of *Anonyma* with non-negligible advantage, then we can construct a PPT adversary B who distinguishes between a random number generator and a pseudo-random number generator with the same advantage as A. Assume A's success probability is

$$Pr[InvAnonym_A(\lambda) = 1] = \frac{1}{2} + \epsilon(\lambda) \quad (20)$$

B runs A as its subroutine to distinguish the pseudo-random number generator from the truly random number generator. B is given a vector of values in Z_q denoted by $\vec{\delta} = (\delta', \delta'')$ and aims at specifying whether $\vec{\delta}$ is selected truly at random or is the output of a PRG. B invokes A as its subroutine and emulates the game of inviter anonymity for A as follows. If A succeeds then B realizes that $\vec{\delta}$ is pseudo-random, otherwise random.

- (1) B is given the security parameter 1^λ and a vector of two values denoted by $\vec{\delta} = \{\delta', \delta''\}$ s.t. $\delta', \delta'' \in Z_q$. Adversary A outputs *Param* including ek, vk , and (F_0, \dots, F_{t-1}) .
- (2) A registers its own users. U_c contains the indices of corrupted members. Also, A instructs B to register users into the system. Let U_h indicate the set of indices registered by B. For each user $i \in U_h$, B obtains a share s_i and verifies its correctness by checking whether g^{s_i} is equal to $\prod_{j=0}^{t-1} F_j^{i'}$.
- (3) (a) A outputs two indices $u_0, u_1 \in U_h$.
 - (b) A outputs a token $Token = (\eta, j^*, \omega)$. j^* is the index of the invitee in U_h .
 - (c) A specifies a set of $t - 1$ indices $I_c \subset U_c$ to be the corrupted inviters. For every $i \in I_c$, and $Token$, A engages in *Igen* with the challenger. A outputs $Inv_i = (\tau_i, e\delta_i)$ and contacts F_{POIC}^R with the input of $(\tau_i, e\delta_i, \gamma_i, \omega)(s_i, r_i, \delta_i)$. B acting as F_{POIC}^R , accepts or rejects A's proof by verifying whether $(\tau_i, e\delta_i, \gamma_i, \omega)$ and (s_i, r_i, δ_i) fit into the relation R as defined in Equation

8. Note that at this step B can learn all the $t - 1$ master shares of corrupted inviters, i.e., $\{s_i\}$ for $i \in I_c$.
- (4)(a) B selects a random bit b . B uses the *Token* and runs *Igen* to create an invitation letter from u_b as $Inv_{u_b} = (\tau_{u_b}, e\delta_{u_b}) = (\omega^{s_{u_b} + \delta'}, Enc_{ek}(\omega^{\delta'}))$ (δ' is given from the distinguish-ability game of PRG).
- (b) B runs *Icoll* over $\{Inv_i\}_{i \in I_c} \cup Inv_b$, sets $\delta^* = \delta''$ and computes
- $$T = \omega^{\delta^*} \cdot \tau_{u_b}^{B_{u_b}} \cdot \prod_{i \in I_c} \tau_i^{B_i}$$
- and
- $$e\Delta = Enc_{ek}(\omega^{\delta^*}) \cdot Enc_{ek}(\omega^{\delta'})^{B_{u_b}} \cdot \prod_{i \in I_c} e\delta_i^{B_i}.$$
- The value of $e\Delta$ will be equal to $Enc_{ek}(\omega^{\delta^* + \delta' \cdot B_{u_b} + \sum_{i \in I_c} \delta_i \cdot B_i})$. B_i and B_{u_b} denote the Lagrange coefficients as defined in Equation 4. B submits $InvLet = (T, e\Delta)$ to the adversary A .
- (5) A outputs a bit b' .
- (6) If $b = b'$ then B outputs 0, otherwise 1.

Note that B follows all the steps as indicated in the $InvAnonym_A(\lambda)$ game and hence is indistinguishable from a real challenger. This means that B also runs in polynomial time (as there is no rewind). B ties the $InvAnonym_A(\lambda)$ game to the security of PRG by embedding δ' and δ'' (the challenge of PRG game) as the randomness δ_{u_b} (used by the non-colluding inviter for the invitation generation), and the value of δ^* (used by the invitee in *Icoll* execution), respectively. Below is the success probability analysis of the reduction.

Let $\vec{\delta}$ be a truly random vector. Once the adversary decrypts $e\Delta$ he obtains

$$\omega^{\delta'' + \Gamma}$$

where

$$\Gamma = \delta' \cdot B_{u_b} + \sum_{i=1}^{t-1} \delta_i \cdot B_i$$

Γ is a function of inviters indices due to the presence of Lagrange coefficients whereas δ'' is a random value completely independent of inviters' indices. If $\vec{\delta}$ is a random vector then δ'' is also a random value from \mathbb{Z}_q . Therefore, in $\omega^{\delta'' + \Gamma}$, Γ is indeed masked with δ'' ($\delta'' + \Gamma \pmod q$ is a completely random element of \mathbb{Z}_q). By this masking, Δ (i.e., $\delta'' + \Gamma$) becomes completely independent of the Lagrange coefficients and A has no advantage to infer the identity of the uncorrupted inviter. Thus, A 's advantage is exactly $\frac{1}{2}$ i.e.,

$$Pr[B(\vec{\delta} \leftarrow Z_q) = 1] = Pr[b = b'] = \frac{1}{2} \quad (21)$$

but if $\vec{\delta}$ is the output of a PRG then

$$Pr[r \leftarrow \{0, 1\}^\lambda : B(\vec{\delta} = PRG(r)) = 1] = Pr[b = b'] = \frac{1}{2} + \epsilon(\lambda) \quad (22)$$

where $\frac{1}{2} + \epsilon(\lambda)$ is the success probability of A (as assumed in our proof in Equation 20). By combining Equations 21 and 22 we have

$$|Pr[r \leftarrow \{0, 1\}^\lambda : B(\vec{\delta} = PRG(r)) = 1] - Pr[B(\vec{\delta} \leftarrow Z_q) = 1]| = \epsilon(\lambda) \quad (23)$$

Equation 23 corresponds to the security definition of PRG (see Equation 1). Thus, if $\epsilon(\lambda)$ is non-negligible, then the distinguisher B can distinguish a PRG from a random generator. This contradicts with the security definition of PRG. Therefore, $\epsilon(\lambda)$ must be negligible according to the

PRG definition. This concludes the security proof of inviter anonymity of *Anonyma*. ■

6.2 Invitation Unforgeability

In an invitation-based system, the invitation unforgeability indicates that people who do not have enough inviters (less than t) should not be able to join the system. Hence, no adversary can forge invitations on his own. A system that prevents unforgeability needs to adhere to the following traits as well, namely, the *invitation non-exchangability* and *prevention of double-invitations*. The former means that invitations are bound to their invitee and are not exchangeable from one to another while the latter indicates that an inviter is only able to make one single (and not more) valid invitation for each invitee. Next, we present a formal definition for invitation unforgeability together with a formal security proof of *Anonyma*.

6.2.1 Security Definition: We define the following game denoted by $InvUnforge_A(\lambda)$ running between a challenger and an adversary. The adversary controls a set of $t - 1$ members. The rest of the users denoted by I_h are controlled by the challenger. Also, the adversary has oracle access to the token generation $Tgen(sk, j)$, invitation generation $Igen(., s_i, Param)$ for $i \in I_h$, and invitation verification $Ivrfy(., ., Param, dk)$ algorithms. Finally, the adversary wins the game if it manages to register to the system successfully, using a token that was not queried from the invitation generation oracle. The success of the adversary asserts that the invitations are forgeable. Otherwise, the system provides invitation unforgeability. We remark that by giving the adversary oracle access to the invitation generation algorithm we aim to capture the *non-exchangability* of invitations. This oracle access is equivalent to having an adversary who eavesdrops on the communication of other invitees and inviters and wishes to forge an invitation over its token. The *prevention of double-invitations* is modeled in the fact that the adversary has the control of $t - 1$ inviters and can internally execute their invitation generation function any number of times to possibly make a double-invitation.

Invitation Unforgeability experiment $InvUnforge_A(\lambda)$:

- (1) The adversary specifies a set I_c consisting of the index of $t - 1$ users to be controlled by the adversary.
- (2) The challenger runs the setup algorithm and outputs $Param$ to the adversary. The next steps (3-6) are the learning phase of the adversary and can be run in an arbitrary order.
- (3)(a) The adversary registers a corrupted user $i \in I_c$ to the system. The adversary can repeat this part for every user $i \in I_c$.
- (b) The adversary instructs the challenger to register an honest user to the system. I_h shall contain the index of honest members.
- (4) The adversary asks the challenger to issue a token. The challenger generates a token for the next available index j . This step may be repeated polynomially many times upon the adversary's request. Q^{Token} holds the set of tokens queried by the adversary.
- (5) The adversary queries invitation verification function on the invitations of his own choice. The challenger runs the $Ivrfy$ algorithm and responds accordingly.
- (6) The adversary has oracle access to the $Igen$ algorithm. That is, the adversary asks the challenger to use a particular token and generate an invitation from an honest member. As such, the adversary specifies the index $i \in I_h$ of an honest member together with a valid $Token_j \in Q^{Token}$. Then, the challenger issues an individual invitation by running $Inv_{i,j} = Igen(Token_j, s_i, Param)$ and gives the output to the adversary. Let $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$ be the set of tokens together with the individual invitations queried by the adversary.
- (7) The adversary outputs an invitation letter $InvLet$ for a valid token $Token' \in Q^{Token}$ for which no query exists in Q^{Inv} .
- (8) If the output of $Ivrfy(InvLet, Token', Param, dk)$ is accepted then the game's output is 1 indicating the adversary's success, 0 otherwise.

Definition 6.3. An invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function $negl(\cdot)$ such that:

$$Pr[InvUnforge_A(\lambda) = 1] = negl(\lambda)$$

At a high level, in order for the adversary to be able to win the game, it has to compute ω^{*S} for some token $Token' = (\tau, j, \omega^*)$ where $Token'$ does not belong to Q^{Inv} . Through the oracle accesses, the adversary learns a set of individual invitations $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$ where $Inv_{i,j} = (\tau_{i,j} = \omega_j^{s_i + \delta_{i,j}}, e_{\delta_{i,j}} = Enc_{ek}(\omega_j^{\delta_{i,j}}))$. The $Inv_{i,j}$ carries no useful information regarding the master value S to the adversary as the master share s_i is masked through a random value $\delta_{i,j}$. There is no way for the adversary to get to learn $\delta_{i,j}$ unless with decryption of $e_{\delta_{i,j}}$ which is not possible as the adversary lacks the decryption key dk . Alternatively, the adversary may attempt to combine invitations issued under different tokens to obtain a valid invitation under a new token $Token'$. This is impossible due to the CDH problem. That is, given $\tau_{i,j} (= \omega_j^{s_i + \delta_{i,j}})$ and $\omega^* (= g^x)$, the adversary must compute $\tau_{i,j}^x$ which corresponds to a solution to the CDH problem. Similarly, the knowledge of ω^* and $F_0 = g^S$ (from $Param$) does not help in making a valid invitation letter since computing ω^{*S} is equivalent to solving the CDH problem. That is, given $\omega^* = g^x$ and $F_0 = g^S$, the adversary shall compute $g^{x \cdot S} = \omega^S$. In the theorem and proof given below, we reduce $InvUnforge_A(\lambda)$ game to the CDH problem.

THEOREM 6.4. *Anonyma satisfies invitation unforgeability as defined in Definition 6.3, in F_{POIC}^R hybrid model, given that the signature scheme Sig is existentially unforgeable under chosen message attack, and Computational Diffie-Hellman problem is hard relative to group G .*

Proof: If there exists a PPT adversary A who breaks the invitation unforgeability with the non-negligible advantage then we can construct a PPT adversary B who solves the CDH problem with non-negligible advantage.

Let ϵ denote the probability of success of A . B interacts with the CDH challenger and also runs A as its subroutine. B is given $G, q, g, X = g^x, Y = g^y \in G$ for which B is supposed to compute $Z = g^{x \cdot y}$.

- (1) A outputs a set of $t - 1$ indices as I_c to be the index of members under its control.
- (2) B runs the setup algorithm and generates the encryption and signature key pairs (ek, dk) (sk, vk) as normal. To set up Shamir secret sharing scheme, B performs as follows. B sets $F_0 = Y$ (recall that F_0 is the commitment to master value S thus $F_0 = g^{f(0)} = g^S$; this implies that B does not know the master value S since it is the discrete logarithm of Y (i.e., y), which is selected by the CDH challenger). B selects $t - 1$ random values $s_{i \in I_c} \leftarrow Z_q$ to be the master shares of the corrupted members. Also, B computes $\gamma_i = g^{s_i}$ for $i \in I_c$. Recall that the share of the master value for the i^{th} user is $f(i)$, thus by setting the master shares of corrupted parties, B fixes $t - 1$ points of polynomial f as $f(i) = s_i$ for $i \in I_c$. These $t - 1$ points together with F_0 , which is indeed $g^{f(0)}$, will fix polynomial f since the degree of f is $t - 1$. Next, B interpolates Y i.e., $(g^{f(0)})$ and $\{(i, \gamma_i)\}_{i \in I_c}$, and computes the commitments F_1, \dots, F_{t-1} (where $F_1 = g^{a_1}, \dots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [41] (where $f = S + a_1 \cdot x + \dots + a_{t-1} \cdot x^{t-1}$).

Note that B does not obtain the exact coefficients of the polynomial f (i.e., a_i values) but only computes the commitments $F_i = g^{a_i}$. This is sufficient for B to simulate the role of the server since it only needs to publish the commitments of the polynomial and not the exact coefficients.

B outputs $param = (G, q, g, ek, vk, (F_0, \dots, F_{t-1}))$, as well as the security parameter 1^λ . Note that B also records the master shares of corrupted members i.e., $\{(i, f(i))\}_{i \in I_c}$ to use in the registration phase.

- (3)(a) A registers a corrupted user to the system (a user with the index $i \in I_c$). As such, B sends $f(i)$ (which was computed during the setup protocol) to A .
- (b) A instructs B to register an honest user to the system. Note that B cannot generate the master shares of honest users since it does not know the coefficients of the function f . However, since it is a local calculation for B , this shortage remains unnoticed to A . B records the index of honest user inside I_h .
- (4) A has oracle access to the token generation $Tgen$. Initially, B draws a random value $j^* \in [1, P(\lambda)]$ where $P(\lambda)$ is the upper-bound on the number of adversary's queries to $Tgen$. B answers the queries of A for $Tgen$ as follows. For the j^{*th} query, B sets $Token_{j^*} = (Sign_{sk}(j^*||X), j^*, X)$ (X was given to B from the CDH game) and inserts (j, X, \perp) into Q^{Token} . Otherwise, B selects a random $r_j \in_R Z_q$, sets $\omega_j = g^{r_j}$ and outputs $Token_j = (Sign_{sk}(j||\omega_j), j, \omega_j)$. B records (j, ω_j, r_j) inside Q^{Token} .
- (5) The adversary queries the invitation verification function on the invitation letters and tokens of his own choice i.e., $InvLet = (T, e\Delta)$ and $Token = (\eta, j, \omega_j)$. B first authenticates the token against the signature verification key. If not verified, B outputs *reject* to A . Also, if $\omega_j = X$, then B aborts. Otherwise,
 - If $Token == Token^*$, then B aborts the experiment.

- If $Token \neq Token^*$, B proceeds as follows. Due to the lack of master value S , B has to run different than the normal *Ivrfy* algorithm. B decrypts $e\Delta$ as $\omega_j^\Delta = Dec_{dk}(e\Delta)$. Next, B retrieves the record of (j, ω_j, r_j) corresponding to ω_j from Q^{Token} and checks whether $F_0^{r_j} \cdot \omega_j^\Delta \stackrel{?}{=} T$ and responds to A accordingly. The right side of this equality check is the same as line 3 of *Ivrfy* Algorithm (Algorithm 3.4) since

$$F_0^{r_j} \cdot \omega_j^\Delta = g^{S \cdot r_j} \cdot \omega_j^\Delta = g^{r_j \cdot S} \cdot \omega_j^\Delta = \omega_j^S \cdot \omega_j^\Delta \quad (24)$$

- (6) A has oracle access to *Igen* algorithm. A outputs an index i of an honest member together with a $Token_j = (\eta, j, \omega_j)$. B first authenticates the token against the signature verification key. If successful, then, it attempts issuing an invitation. Notice that B cannot generate the invitation by following *Igen* since it does not have the master share of honest users i.e., s_i for $i \in I_h$. B performs differently to compute a valid invitation as explained next. B retrieves the record (j, ω_j, r_j) from Q^{Token} (if the token is valid and has a correct signature from the server then it must be already queried by the adversary and hence should exist in Q^{Token} , otherwise, the signature forgery happens which is not possible due to the security of the underlying signature scheme). B computes $\gamma_i = \prod_{j=0}^t F_j^{r_j}$ as well as selects a random value $\delta_i \in_R Z_q$. Then, B constructs $\tau_{i,j} = \gamma_i^{r_j} \cdot g^{\delta_i \cdot r_j}$ where r_j is the discrete logarithm of ω_j in base g . It is immediate that $\tau_{i,j}$ is well-structured since

$$\tau_{i,j} = \gamma_i^{r_j} \cdot g^{\delta_i \cdot r_j} = (g^{s_i})^{r_j} \cdot (g^{\delta_i})^{r_j} = (g^{r_j})^{s_i} \cdot (g^{r_j})^{\delta_i} = \omega_j^{s_i} \cdot \omega_j^{\delta_i} = \omega_j^{s_i + \delta_i} \quad (25)$$

B constructs $e\delta_{i,j}$ as $Enc_{ek}(\omega_j^{\delta_i})$ and outputs $Inv_{i,j} = (\tau_{i,j}, e\delta_{i,j})$ to A .

Finally, B acts as F_{POIC}^R and waits for A 's message asking verification of $(\tau_{i,j}, e\delta_{i,j}, \gamma_i, \omega_j)$ for which B responds *accept* to A . B keeps the set of individual invitations and their tokens queried by A in $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$.

- (7) The adversary outputs an invitation letter $InvLet = (T, e\Delta)$ for a token $Token'$ for which no query has been made from *Igen* i.e., $Token' \notin Q^{Inv}$.
- (8) B verifies whether the $Token'$ is correctly signed under sk . If not, B outputs \perp to CDH challenger. Otherwise:
- If $Token' \neq Token^*$, B outputs \perp to the CDH game.
 - If $Token' = Token^*$, B outputs $T \cdot Dec_{dk}(e\Delta)^{-1}$ to the CDH challenger. In fact, if A constructs $InvLet$ correctly, we expect that $T = \omega^{*S+\Delta}$ and $e\Delta = Enc(\omega^{*\Delta})$. Given that $X = g^x = \omega^*$ and $Y = g^y = g^S$, we have

$$T \cdot Dec_{dk}(e\Delta)^{-1} = (\omega^*)^{S+\Delta} \cdot (\omega^*)^{\Delta^{-1}} = (\omega^*)^S = (g^x)^y = g^{xy} \quad (26)$$

$g^{x \cdot y}$ is the solution to the given CDH problem.

This is immediate that B runs in polynomial time. The index j^* chosen by B at step 4 represents a guess as to which *Tgen* oracle query of A will correspond to the token of eventual invitation letter forgery output by A . If this guess is correct, then A 's view while running with B is identical to $InvUnforge_A(\lambda)$ game.

When B guesses correctly and A outputs a forgery, then B can solve the given instance of CDH. Assume that A 's advantage in $InvUnforge_A(\lambda)$ game is ϵ . The probability that B wins is

$$\begin{aligned}
 Pr[B \text{ wins}] &= Pr[B(G, q, g, X = g^x, Y = g^y) = g^{x \cdot y}] \\
 &= Pr[A \text{ wins} \wedge (Token' = Token^*)] \\
 &= Pr[A \text{ wins} | Token' = Token^*] \cdot Pr[Token' = Token^*] \\
 &\geq \epsilon \cdot \frac{1}{Poly(\lambda)}
 \end{aligned} \tag{27}$$

The last equality holds since the number of queries made by A is at most $P(\lambda)$ (P is polynomial in 1^λ), hence, the probability $Token^* = Token'$ is $\frac{1}{Poly(\lambda)}$. Note that due to the signature unforgeability, A cannot create a valid token outside of the set of queried tokens i.e., $\notin Q^{Token}$.

Assuming that ϵ is non-negligible, B also wins with non-negligible probability. This contradicts the hardness of the CDH problem. Hence A 's success probability in $InvUnforge_A(\lambda)$ must be negligible. This concludes the proof. ■

6.3 Security of Anonymax

Inviter Anonymity: The inviter anonymity of *Anonymax* can be defined identically to the experiment of $InvAnonym_A(\lambda)$. The challenger shall control the honest members, i.e. U_h and invitee whereas the adversary will have the control of $S_{authenticating}$ and all the *registration* servers S_j together with the corrupted members I_c which shall constitute $t - 1$ inviters of the invitee. *Anonymax* meets inviter anonymity due to the similar proof supplied for *Anonyma*. Without loss of generality and for the sake of simplicity, we consider only one *registration* server to exist, though the extension of proof for multiple *registration* servers is straightforward. In particular, the following theorem holds for *Anonymax* with one *authenticating* server and one *registration* server.

THEOREM 6.5. *Anonymax provides inviter anonymity in F_{POIC}^R hybrid model (as defined in Equation 38), assuming that PRG is a secure pseudo-random number generator.*

Proof Sketch: Given that a PPT adversary A' can break the inviter anonymity game for *Anonymax* with non-negligible advantage, we can construct an adversary B' to distinguish between a PRG and a truly random number generator. The internal code of adversary B' shall be identical to the simulator B in proof of Theorem 6.2. The only difference is in the *SetUp* phase where the challenger outputs two pair of encryption keys $(ek_{reg}, ek_{authenticating})$ among which only ek_{reg} will be used throughout the experiment.

Invitation Unforgeability: Recall that invitation unforgeability guarantees that a corrupted invitee with an insufficient number of inviters would not be able to join the system. In a cross-network invitation-based system, the invitation unforgeability should additionally hold for the *registration* service. That is, if Alice does not have enough inviters from the *authenticating* system, she should not be able to successfully register to the *registration* service.

Note that in a cross-network invitation-based system, invitation unforgeability cannot be defined for the case that $S_{authenticating}$ acts against S_{reg} , i.e. $S_{authenticating}$ wants to generate valid invitations for the invitee of its choice to join the *registration* service. This is trivial since $S_{authenticating}$ can register arbitrary many users into its system (i.e., *authenticating* system). Then, every subset of t registered users of *authenticating* service will consequently be able to issue invitations and register arbitrary many users into the *registration* service. Note that this is not a limitation imposed by our design and rather is implicit in any cross-network invitation-based system.

In the invitation unforgeability game as defined in $XInvUnforge_A(\lambda)$, we consider N *registration* servers which all accept invitations from the members of one *authenticating* server. The adversary

plays on behalf of $t - 1$ corrupted users of the *guest* service and a subset of *registration* servers. The challenger controls the honest users, i.e. I_h of the *authenticating* service together with $S_{\text{authenticating}}$ and some of the uncorrupted *registration* servers. At the end of the game, the mission of the adversary as a corrupted invitee with an insufficient number of inviters is to successfully register to one of the honest *registration* servers S_{j^*} controlled by the challenger.

In $XInvUnforge_A(\lambda)$, we index *registration* servers as S_j , where $1 \leq j \leq N$, and the *authenticating* server as S_0 . The set of servers controlled by the adversary are denoted as set C . We assume H indicates the set of un-corrupted *registration* servers. The set of members of *authenticating* service is denoted by $U_{\text{authenticating}}$. I_c represents the set of $t - 1$ corrupted members in the *authenticating service* whereas I_h contains the indices of the honest members. We have $U_{\text{authenticating}} = I_h \cup I_c$. We prefix the algorithms with its executing entity, e.g. we write $S_j.Tgen$ to show the invocation of the token generation algorithm at the server j .

Invitation Unforgeability experiment $XInvUnforge_A(\lambda)$ for cross-network invitation based system:

- (1) The adversary specifies a set $I_c \subset U_{\text{authenticating}}$ consisting of the index of $t - 1$ users to be under his control.
- (2) The challenger runs *Setup* for all $S_j \in H$ and outputs $Param_j$ to the adversary. The adversary outputs $Param_j$ for $j \in C$. The next steps (3-6) are the learning phase of the adversary and can be run in an arbitrary order.
- (3)(a) The adversary registers a corrupted user $i \in I_c$ to the *authenticating* system. The adversary repeats this part for every user $i \in I_c$.
 - (b) The adversary instructs the challenger to register an honest user i to the *authenticating* system where $i \in I_h$.
- (4) The adversary has Oracle access to the $S_j.Tgen$ for $j \in H$. Also, the adversary generates a *Token* for a user $i \in I_h$ from S_j where $j \in C$ and hands it over to the challenger.
- (5) The adversary has oracle access to $S_j.XIvrfy$ for $j \in H$.
- (6) The adversary has oracle access to the *Igen* algorithm. That is, the adversary specifies the index l of an honest member i.e., $l \in I_h$ and a server index $j \in H \cup C$ together with a *Token* issued by S_j .

The challenger generates an individual invitation by running $Inv_l = Igen(Token, s_l, Param_j)$ and gives the output Inv_l to the adversary. Let $Q_j^{Inv} = \{(Token, Inv_l)\}$ be the set of tokens together with the individual invitations queried by the adversary to be generated by the l^{th} user for the j^{th} service.

- (7) The adversary outputs an invitation letter *InvLet* together with token $Token'$ for the registration in the j^{*th} server where $j^* \in H$. There should not be any issued invitation using $Token'$ in $Q_{j^*}^{Inv}$.
- (8) If the output of $XIvrfy(InvLet, Token', Param_{\text{authenticating}}, Param_{S_{j^*}}, dk_{S_{j^*}})$ is accepted, then the game's output is 1 indicating the adversary's success, 0 otherwise.

Definition 6.6. An cross-network invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function $negl(\cdot)$ such that:

$$Pr[XInvUnforge_A(\lambda) = 1] = negl(\lambda)$$

THEOREM 6.7. *AnonymaX satisfies invitation unforgeability as defined in Definition 6.6, in F_{POIC}^R and F_{PODL}^R hybrid model, given that the signature scheme Sig is existentially unforgeable under chosen message attack, and Computational Diffie-Hellman problem is hard relative to group G .*

At a high level, The reduction idea between CDH problem and $XInvUnforge_A(\lambda)$ of *AnonymaX* is similar to $InvUnforge_A(\lambda)$. However, in *AnonymaX*, the simulator B additionally can extract the CDH solution during the *Tgen* and *XIvrfy* which we explain below. B is given $X = g^x$ and $Y = g^y$ from the CDH challenger and sets Y as the commitment to the master value S . B also guesses at which query of *Tgen* A will succeed to forge a valid *InvLet*. B sets the value ω of that token to X . B can solve the CDH challenge if

- A creates a token with the value of X for which A must prove the knowledge of the discrete logarithm x . Then B outputs Y^x as the CDH solution.
- The adversary A queries *XIvrfy* with a valid invitation letter *InvLet* over the token with $\omega = X$, then B extracts the CDH solution. The *InvLet* is of the form $\omega^S = g^{xy}$ which is the solution to the CDH problem.
- A submits a valid invitation letter using the token X . That is of the form $\omega^S = g^{xy}$ which is the solution to the CDH problem.

A may also win by forging a token (i.e., a signature) on behalf of an honest *registration* server $\in H$. However, since the signature scheme is secure, the probability of signature forgery is negligible.

Proof: If there exists a PPT adversary A who breaks the invitation unforgeability of *AnonymaX* with non-negligible advantage, then we can construct a PPT adversary B who solves the CDH problem with non-negligible advantage.

Let ϵ denote the probability of success of A . B interacts with the CDH challenger and also runs A as its subroutine. B is given the security parameter 1^λ , $G, q, g, X = g^x, Y = g^y \in G$ for which B is supposed to compute Z s.t. $Z = g^{x \cdot y}$.

- (1) A outputs a set of $t - 1$ indices as I_c to be the index of members under its control.
- (2) For every $S_j, j \in H$, B runs the setup algorithm and generates the encryption and signature key pairs $(ek_{S_j}, dk_{S_j}) (sk_{S_j}, vk_{S_j})$ as normal. $Param_{S_j}$ will be (ek_{sk_j}, vk_{sk_j}) .

Similarly, B sets up an encryption and signature key pairs for $S_{authenticating}$ as $(ek_{authenticating}, dk_{authenticating})$ and

$(sk_{authenticating}, vk_{authenticating})$, respectively. As for the initialization of Shamir secret sharing scheme, B performs as follows. B sets $F_0 = Y$ (recall that F_0 is the commitment to master value S thus $F_0 = g^{f(0)} = g^S$; this implies that B does not know the master value S since it is the discrete logarithm of Y (i.e., y), which is selected by the CDH challenger). B selects $t - 1$ random values $s_{i \in I_c} \leftarrow Z_q$ to be the master shares of the corrupted members. Also, B computes $\gamma_i = g^{s_i}$ for $i \in I_c$. Recall that the share of the master value for the i^{th} user is $f(i)$, thus by setting the master shares of corrupted parties, B fixes $t - 1$ points of polynomial f as $f(i) = s_i$ for $i \in I_c$. These $t - 1$ points together with F_0 , which is indeed $g^{f(0)}$, will fix polynomial f since the degree of f is $t - 1$. Next, B interpolates Y i.e., $(g^{f(0)})$ and $\{(i, \gamma_i)\}_{i \in I_c}$, and computes the commitments F_1, \dots, F_{t-1} (where $F_1 = g^{a_1}, \dots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [41] (where $f = S + a_1 \cdot x + \dots + a_{t-1} \cdot x^{t-1}$). Note that B does not obtain the exact coefficients of the polynomial f (i.e., a_i values) but only computes the commitments $F_i = g^{a_i}$. This is sufficient for B to simulate the role of the *authenticating* server since it only needs to publish the commitments of the polynomial and not the exact coefficients.

B outputs $param = (G, q, g, ek_{\text{authenticating}}, vk_{\text{authenticating}}, (F_0, \dots, F_{t-1}))$, as well as the security parameter 1^λ to the adversary. Note that B also records the master shares of corrupted members i.e., $\{(i, f(i))\}_{i \in I_c}$ to use in the registration phase.

- (3)(a) A registers a corrupted user to the system (a user with the index $i \in I_c$). As such, B sends $f(i)$ (which was computed during the *SetUp* protocol) to A .
- (b) A instructs B to register an honest user to the system. Note that B cannot generate the master shares of honest users since it does not know the coefficients of the function f . However, since it is a local calculation for B , this shortage remains unnoticed to A . B records the index of the honest user inside I_h .

- (4) A has oracle access to token generation i.e., $S_{\text{authenticating}}.Tgen$ and $S_j.Tgen$ for all $j \in H$. B keeps the set of tokens queried by A for each server S_j inside Q_j^{Token} . Initially, B draws two random values $j^* \in [1, N]$ (to be the guess over the index of the honest registration server for which the adversary comes up with the invitation letter forgery) and $l^* \in [1, P(\lambda)]$ where $P(\lambda)$ is the upper-bound on the number of adversary's queries to $Tgen$ for each of the servers.

- If j is equal to j^* , and if this is the l^* query to $S_{j^*}.Tgen$ then B returns

$$Token^* = (Sign_{sk_{S_{j^*}}}(l^* || X), l^*, X)$$

X was given to B from the CDH game. B plays the role of F_{PODL}^R , receives the verification request of (G, q, g, X) from the adversary and outputs *accept* to the adversary. B inserts (X, \perp) into $Q_{j^*}^{Token}$.

- If $j \neq j^*$, and assuming this is the l^{th} query of adversary to $S_j.Tgen$, B selects a random $r \in_R Z_q$, sets $\omega = g^r$ and outputs $Token = (Sign_{sk_{S_j}}(l || \omega), l, \omega)$. B plays the role of F_{PODL}^R , receives the verification request of (G, q, g, ω) from the adversary and outputs *accept* to the adversary. B inserts (ω, r) to Q_j^{Token} .

The adversary may generate a token $Token = (\eta, l, \omega)$ for a user $l \in I_h$ from S_j where $j \in C$ and hands over to the challenger. The adversary contacts F_{PODL}^R i.e., the challenger B and hands over $((G, q, g, \omega), r)$. B checks whether $g^r = \omega$ and accepts or rejects the token accordingly. Also, B verifies the signature η against the verification key of S_j and accepts or rejects the token accordingly. If the verification passed successfully, B stores (ω, r) in Q_j^{Token} . If $\omega = X$ (the CDH challenge), and the token is accepted, then B outputs Y^r to the CDH challenger.

- (5) The adversary queries $S_j.XIvrfy(InvLet, Token, Param_{\text{authenticating}}, dk_{S_j})$ for $j \in H$ on the invitation letters and tokens of his own choice i.e., $InvLet = (T, e\Delta)$ and $Token = (\eta, l, \omega)$. B runs $XIvrfy$ algorithm and responds accordingly. If the output of $XIvrfy$ is not reject and if $j = j^*$ and $Token = Token^*$ (i.e., $\omega = X$), then B outputs $T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1}$ to the CDH game.

$$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = X^{S+\Delta} \cdot X^{-\Delta} = X^S = g^{xS} = g^{xy} \quad (28)$$

- (6) A has oracle access to $Igen$ algorithm. A asks the challenger to generate an invitation from the honest member i for the registration server $j \in H \cup C$ using a $Token = (\eta, l, \omega)$. B first authenticates the token against the signature verification key of S_j . If not verified, B outputs *reject* to A . Also, if $\omega = X$, then B aborts. Otherwise, B attempts issuing an invitation. Notice that B cannot generate the invitation by following $Igen$ since it does not have the master share of honest users i.e., s_i for $i \in I_h$. B performs differently to compute a valid invitation as explained next. B computes $\gamma_i = \prod_{v=0}^t F_v^{i^v} = g^{s_i}$ (the second equality holds due to Equation 6) as well as selects a random value $\delta_i \in_R Z_q$. Then, B constructs $\tau_i = \gamma_i^r \cdot g^{\delta_i \cdot r}$ where r is the discrete logarithm of ω in base g . It is immediate that τ_i (to be the first component of the invitation letter) is well-structured since

$$\tau_i = \gamma_i^r \cdot g^{\delta_i \cdot r} = (g^{s_i})^r \cdot (g^{\delta_i})^r = (g^r)^{s_i} \cdot (g^r)^{\delta_i} = \omega^{s_i} \cdot \omega^{\delta_i} = \omega^{s_i + \delta_i} \quad (29)$$

B constructs $e\delta_i$ as $Enc_{ek}(\omega^{\delta_i})$ and outputs $Inv_i = (\tau_i, e\delta_i)$ to A .

Finally, B acts as F_{POIC}^R and waits for A 's message asking verification of $(\tau_i, e\delta_i, \gamma_i, \omega)$ for which B responds *accept* to A . B keeps the set of individual invitations and their tokens queried by A for each server S_j in $Q_j^{Inv} = \{(Inv_i, Token)\}$.

- (7) The adversary outputs an invitation letter $InvLet = (T, e\Delta)$ for a valid token $Token'$ issued by $S_{j' \in H}$ i.e., $Token' \in Q_{j'}^{Token}$ for which no query has been made from $S_{j'}$. Igen i.e., $Token' \notin Q_{j'}^{Inv}$.
- (8) B verifies whether the $Token'$ is correctly signed under $sk_{j'}$. If not, B outputs \perp to CDH challenger. Otherwise:
- If $j' \neq j^*$ or $Token' \neq Token^*$ B outputs \perp to the CDH game.
 - If $j' = j^*$ and $Token' = Token^*$, B outputs $T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1}$ to the CDH challenger. In fact, if A constructs $InvLet$ correctly, we expect that $T = X^{S+\Delta}$ and $e\Delta = Enc_{ek_{S_{j^*}}}(X^\Delta)$. Given that $X = g^x$ and $Y = g^y = g^S$, we have

$$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = (X)^{S+\Delta} \cdot (X)^\Delta^{-1} = (X)^S = (g^x)^y = g^{xy} \quad (30)$$

$g^{x \cdot y}$ is the solution to the given CDH problem.

This is immediate that B runs in polynomial time. The index j^* and l^* chosen by B at step 4 represents a guess as to which server S_{j^*} and to which S_{j^*} .Tgen oracle query of A will correspond to the token of eventual invitation letter forgery output by A . If this guess is correct, then A 's view while running with B is identical to $XInvUnforge_A(\lambda)$ game.

When B guesses correctly and A outputs a forgery, then B can solve the given instance of CDH. Assume that A 's advantage in $XInvUnforge_A(\lambda)$ game is ϵ . The probability that B wins is

$$\begin{aligned} Pr[B \text{ wins}] &= Pr[B(G, q, g, X = g^x, Y = g^y) = g^{x \cdot y}] \quad (31) \\ &= Pr[A \text{ wins} \wedge (Token' = Token^* \text{ AND } j' = j^*)] \\ &= Pr[A \text{ wins} | Token' = Token^* \text{ AND } j' = j^*] \cdot Pr[Token' = Token^* \text{ AND } j' = j^*] \\ &\geq \epsilon \cdot \frac{1}{Poly(\lambda)} \cdot \frac{1}{N} \end{aligned}$$

The last equality holds since the number of queries made by A is at most $Poly(\lambda)$ (i.e., polynomial in λ), and there are N registration servers (honest and corrupted), hence, the probability $Token^* = Token'$ and $j' = j^*$ is at least $\frac{1}{Poly(\lambda)} \cdot \frac{1}{N}$.

A may attempt to forge a token on behalf of $S_{j'}$ for which it has obtained an individual invitation from an honest user for the registration in one of the corrupted registration servers. However, due to the signature unforgeability, A cannot create a valid token outside of the set of queried tokens i.e., $\notin Q_j^{Token}$ for all $j \in H$. Also, all the queries to $Igen(Token = (\eta, i, \omega), s_l, Param_{S_j})$ where $l \in I_h$ and $j \in C$ are answered if the given $Token$ is generated by the corrupted server S_j correctly i.e., $Token \in Q_j^{Tgen}$ which means that the adversary has passed ZKPODL successfully (knows the DL of the ω). The presence of zero-knowledge proof will prevent the adversary from using a token of an honest server since the adversary does not know the DL of ω due to the hardness of discrete logarithm assumption. Without ZKPODL, the adversary can win the $XInvUnforge_A(\lambda)$ ⁸.

⁸ A takes ω from one of the tokens $Token = (\eta, l, \omega)$ in $Q_{j^*}^{Tgen}$ and then generates a valid token $Token'' = (Sign_{sk_j}(\omega), i, \omega)$ at step 4 from a corrupted server $S_{j \in C}$. Next, A queries $Igen(Token'', s_l, Param_{S_j})$ for $l \in I_h$ and obtains a valid invitation $Inv_l = (\tau_l, e\delta_l)$. Given Inv_l and $dk_{sk_{j^*}}$, the adversary would be able to open $e\delta_l$ to ω^{δ_l} hence can construct its t^{th} valid individual invitation as $Inv_t = (\tau_l, Enc_{ek_{S_{j^*}}}(\omega^{\delta_l}))$ for a $Token = (\eta, l, \omega) \in Q_{j^*}^{Tgen}$.

Assuming that ϵ is non-negligible, B also wins with non-negligible probability. This contradicts the hardness of the CDH problem. Hence A 's success probability in $InvUnforge_A(\lambda)$ must be negligible. This concludes the proof. ■

7 RELATED WORKS

In this section, we investigate related studies that show the most similarity to the invitation-only registration systems and address the confidentiality of the inviter-invitee relationship, namely, group signatures, selective-disclosure credentials, electronic voting (e-voting) systems, and ring-based signature schemes. However, the research done in these categories suit their unique settings and show inefficiency issues when deployed in the invitation-only registration scenario. More details are provided below.

7.1 Group Signatures and Linkable Group Signatures

In group signature schemes [15, 29, 30], a group of users is granted signature power by a group authority in such a way that any member of the group can anonymously sign on behalf of the group. One can consider the adoption of group signatures for the invitation-based system in which the signers are the inviters and the signature corresponds to an invitation. However, such adoption does not preserve invitation unforgeability [15]. In specific, there is no limit on the number of signatures (invitations) that one (inviter) can generate for an invitee hence no invitation unforgeability is guaranteed. The unforgeability can be addressed by adopting the linkable versions of group signatures [6, 11, 19, 45], where the identity of the signer gets disclosed in case she attempts signing the same message twice (i.e., making multiple invitations for the same invitee). However, these schemes do not protect the inviter anonymity against the group administrator. In specific, the group administrator is the one who can identify the signer from a given signature. This does not adhere to the inviter anonymity requirement of invitation-based systems.

7.2 Selective-Disclosure Credentials with Threshold Issuance

Selective-disclosure credentials with threshold issuance [43] consist of a set of distributed and mutually un-trusted authorities who can jointly issue credentials on users private/public attributes. Users later can use their credentials to anonymously reveal the possession of certain attributes to verifiers. Credentials enjoy unforgeability in the face of a small subset of corrupted issuing authorities. Moreover, users can randomize their credentials and protect their attributes against the conspiracy of the verifier and all the authorities. One can adopt such a scheme to enable an invitation-based system where the issuing authorities correspond to the inviters, the user is the invitee, the issued credentials constitute invitations, and the verifier has the role of the group administrator. While such integration achieves invitation-unforgeability, it falls short in preserving the anonymity of the inviter-invitee relationship. Notice that the notion of anonymity in selective-disclosure credential [43] concerns the attribute anonymity (i.e., being able to anonymously prove the possession of an attribute without explicitly revealing the credentials) whereas the concept of anonymity in an invitation-based system regards inviter-invitee relationship, i.e., to provably ensure that no verifier (i.e., the group administrator in the invitation-based scenario) can collude with a subset of issuing authorities (i.e., colluding inviters) and identify other non-colluding authorities (i.e., verifiers) involved in the credential issuance. The latter definition has no clear and provable measurement in the selective-disclosure credential schemes (as it is an orthogonal and irrelevant concern in this context).

The adversary combines Inv_t with $t - 1$ invitations issued by the $t - 1$ corrupted inviters under its control and hands over an intact $InvLet$ to B .

7.3 Electronic Voting (e-voting)

Electronic voting systems consist of a set of voters, some candidates to be voted, and one/multiple authorities which handle tallying. An e-voting system must ensure that only the authorized users participate in the voting, and each voter casts only one vote. More importantly, the content of the individual votes must be kept private, i.e., no vote can be traced back to its voter. In the literature, this property is known as vote privacy, anonymity, and untraceability. E-voting techniques are similar to the anonymous invitation-only systems in many aspects. The role of voters is analogous to the inviters. Each round of the election with the Yes/No votes for a candidate can be treated as inviters casting their invitations for the registration of a newcomer. A Yes vote indicates inviting the candidate/newcomer and a No vote implies not inviting. Preserving the privacy of the vote is equivalent to the inviter anonymity. Likewise, the prevention of double-voting resembles the invitation unforgeability.

Despite the aforementioned similarities, e-voting proposals fall short in satisfying inviter anonymity, invitation unforgeability, and scalability simultaneously. To illustrate this incompatibility, we first classify the e-voting techniques into two main categories: 1- explicit vote casting, 2-anonymous vote casting. Under each category, we identify the subtleties to transplant the e-voting solution into the invitation-only systems.

- (1) Explicit vote casting: The voter authenticates himself to the authorities explicitly and immediately casts his private ballot. The ballot is shielded using either a threshold encryption scheme whose decryption key is divided between multiple authorities [27, 39], or secret sharing schemes where multiple authorities obtain one share of the ballot [35]. Before tallying the votes, the identifiable information shall be removed from the individual votes either by shuffling them through mix-net [16] or by homomorphically aggregating them [35]. In the context of the invitation-only system, this type of proposal has performance problems. That is, to preserve the inviter's anonymity (namely, hiding the identity of voters with the Yes vote), all the members should participate in the voting (including those who will cast a No vote). Otherwise, the real inviters will be revealed to the voting authorities. This imposes an unnecessary load on the non-inviter (voters with the No votes). In contrast, in *Anonyma*, the entire invitation procedure is carried out only by the invitee and his inviters.
- (2) Anonymous vote casting: This technique relies on one-time pseudonyms together with an anonymous communication channel. A voter hands over its credential (e.g., social security number – SSN) to the voting authority. Then, through a blind signature scheme, the voting authority issues a signature on the voter's pseudonym (that is also bond to the voter's SSN). The pseudonym is a one-time value and untraceable to the real identity, i.e., SSN. Later on, a voter casts a vote under his pseudonym and via an anonymous communication channel to the voting authority. Voters attempting to vote twice will have to risk the disclosure of their real identities (i.e., SSN) [4, 36]. When we integrate this solution to the invitation-only system, the main problem is that the pseudonyms are one-time hence a user cannot use the same pseudonym for multiple elections (i.e., to invite different users). Otherwise, his identity and will be disclosed. To cope with this issue, upon the arrival of each newcomer, the authority has to issue new pseudonyms for all the existing members to enable them to act as the inviter for the newcomer (regardless of being the inviters of the newcomer or not). This is certainly not an efficient solution as the load of the authority scales linearly with the number of joining members. Moreover, all the existing members also have to work linearly in the number of joining users. Alternatively, the authority should issue multiple pseudonyms (instead of one) for each member. However, this is not clear how to ensure that an inviter will only be able to use one pseudonym for each newcomer. In other words, the inviter should

not be able to use all of his pseudonyms to make t valid invitations for just a single invitee since otherwise it would violate the invitation unforgeability (in which the invitations must be issued by t *distinct* inviters).

7.4 (t,N) Threshold Ring Signature

A ring signature is essentially a variant of group signature without any manager [3]. A ring signature specifies a group of N signers together with a proof that shall convince any verifier that a message is signed by t members of the group [25]. A ring signature scheme must satisfy three properties: 1-correctness which denotes that every group of t signers must be able to create a valid signature and proof, 2-unforgeability that means making a valid signature is not feasible for non-signers, and 3-anonymity which indicates that given a signature and its proof, the identity of the signers should not be predictable with probability negligibly better than $\frac{1}{N}$.

An invitation-based system can be instantiated from a threshold ring signature, assuming that the signers are the inviters and a valid signature constitutes a valid invitation for a newcomer. The important shortcoming of such schemes is that their running time complexity for the generation of an invitation is at least linearly dependent on the size of the system, i.e., the total number of existing members [8, 10, 31, 34]. In some other cases, the dependency is exponential [25]. The same issue applies to the length of the signature (invitation letter) as it is comprised of $O(N)$ group elements where N is the total number of existing members. This considerably degrades the system's performance. In contrast, the performance of *Anonyma* is only influenced by the threshold of t and is independent of the size of the system. That is, the invitation generation complexity is $O(t)$, and the invitation verification is done in $O(1)$. Also, the invitation length is $O(1)$.

8 CONCLUSION

Anonyma was an anonymous invitation-based system fulfilling *inviter anonymity* and *invitation unforgeability* properties in a *malicious adversarial model*. Both properties were associated with formal security definition and proof. The inviter anonymity rooted from the security of the employed pseudo-random generator and the invitation unforgeability relied on the hardness of the computational Diffie-Hellman assumption.

Anonyma featured light and efficient computation, i.e., the constant overhead for the inviters and the system administrator and $O(t)$ overhead for the invitee. Moreover, invitations were short and of constant size containing only two group elements. The non-intrusiveness of the system was achieved where the proposed protocols namely, invitation generation, aggregation, and verification which were stand-alone and executed locally by their respective callers needless of other entities' presence. *Anonyma* enjoys scalability in terms of the number of authorized inviters that can be dynamically added to the system. The process of inviter authorization was of constant computation and communication complexity and in essence, did not require re-keying of the existing authorized inviters. We also proposed the cross-network version of *Anonyma* i.e., *AnonymaX* empowering third-party authentication. *AnonymaX* enjoys the same level of scalability and efficiency as *Anonyma*. We separately proved the invitation unforgeability and inviter anonymity for the cross-network version.

ACKNOWLEDGEMENTS

We acknowledge the support of the Turkish Academy of Sciences, the Royal Society of UK Newton Advanced Fellowship NA140464, and European Union COST Action IC1306.

REFERENCES

- [1] [n. d.]. [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc977993\(v=technet.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc977993(v=technet.10)?redirectedfrom=MSDN).
- [2] [n. d.]. <https://www.synopsys.com/blogs/software-security/5-reasons-third-party-authentication/>.
- [3] Amit K Awasthi and Sunder Lal. 2005. ID-based ring signature and proxy ring signature schemes from bilinear pairings. *arXiv preprint cs/0504097* (2005).
- [4] Josh Benaloh and Dwight Tuinstra. 1994. Receipt-free secret-ballot elections. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, 544–553.
- [5] Manuel Blum and Silvio Micali. 1984. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing* 13, 4 (1984), 850–864.
- [6] Dan Boneh and Hovav Shacham. 2004. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM conference on Computer and communications security*. 168–177.
- [7] Sanaz Taheri Boshrooyeh and Alptekin Küpçü. 2017. Anonymous invitation-based system. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 219–235.
- [8] Xavier Boyen. 2007. Mesh signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 210–227.
- [9] John Brainard, Ari Juels, Ronald L Rivest, Michael Szydlo, and Moti Yung. 2006. Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 168–178.
- [10] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. 2002. Threshold ring signatures and applications to ad-hoc groups. In *Annual International Cryptology Conference*. Springer, 465–480.
- [11] Jan Camenisch and Markus Michels. 1998. A group signature scheme with improved efficiency. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 160–174.
- [12] Abdelberri Chaabane, Gergely Acs, Mohamed Ali Kaafar, et al. 2012. You are what you like! information leakage through users’ interests. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*.
- [13] Sanjit Chatterjee, Darrel Hankerson, and Alfred Menezes. 2010. On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In *International Workshop on the Arithmetic of Finite Fields*. Springer, 114–134.
- [14] David Chaum and Torben Pryds Pedersen. 1992. Wallet databases with observers. In *Annual International Cryptology Conference*. Springer, 89–105.
- [15] David Chaum and Eugène Van Heyst. 1991. Group signatures. In *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 257–265.
- [16] David L Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, Vol. 24. ACM, 84–90.
- [17] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654.
- [18] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.
- [19] Matthew Franklin and Haibin Zhang. 2012. Unique group signatures. In *European Symposium on Research in Computer Security*. Springer, 643–660.
- [20] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 218–229.
- [21] Neil Zhenqiang Gong and Bin Liu. 2018. Attribute inference attacks in online social networks. In *ACM Transactions on Privacy and Security (TOPS)*, Vol. 21. ACM, 3.
- [22] Neil Zhenqiang Gong and Di Wang. 2014. On the security of trustee-based social authentications. In *IEEE transactions on information forensics and security*, Vol. 9. IEEE, 1251–1263.
- [23] Jens Groth. 2005. Non-interactive zero-knowledge arguments for voting. In *International Conference on Applied Cryptography and Network Security*. Springer, 467–482.
- [24] Carmit Hazay and Yehuda Lindell. 2010. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media.
- [25] Toshiyuki Ishiki and Keisuke Tanaka. 2005. An $(n-t)$ -out-of- n threshold ring signature scheme. In *Australasian Conference on Information Security and Privacy*. Springer, 406–416.
- [26] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography*. CRC press.
- [27] Aggelos Kiayias and Moti Yung. 2004. The vector-ballot e-voting approach. In *International Conference on Financial Cryptography*. Springer, 72–89.
- [28] David W Kravitz. 1993. Digital signature algorithm. US Patent 5,231,668.
- [29] Benoît Libert, Thomas Peters, and Moti Yung. 2012. Group signatures with almost-for-free revocation. In *Annual Cryptology Conference*. Springer, 571–589.

- [30] Benoît Libert, Thomas Peters, and Moti Yung. 2012. Scalable group signatures with revocation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 609–627.
- [31] Joseph K Liu, Victor K Wei, and Duncan S Wong. 2003. A separable threshold ring signature scheme. In *International Conference on Information Security and Cryptology*. Springer, 12–26.
- [32] Shah Mahmood. 2013. Online social networks: Privacy threats and defenses. In *Security and Privacy Preserving in Social Networks*. Springer, 47–71.
- [33] G Pon Malar and C Emilin Shyni. 2015. Facebookfs trustee based social authentication. In *Int. J. Emerg. Technol. Comput. Sci. Electron*, Vol. 12. 224–230.
- [34] Carlos Aguilar Melchor, Pierre-Louis Cayrel, Philippe Gaborit, and Fabien Laguillaumie. 2011. A new efficient threshold ring signature scheme based on coding theory. In *IEEE Transactions on Information Theory*, Vol. 57. IEEE, 4833–4842.
- [35] Divya G Nair, VP Binu, and G Santhosh Kumar. 2015. An improved e-voting scheme using secret sharing based secure multi-party computation. In *arXiv preprint arXiv:1502.07469*.
- [36] Michael J Radwin and Phil Klein. 1995. An untraceable, universally verifiable voting scheme. In *Seminar in Cryptology*. 829–834.
- [37] Alon Rosen. 2004. A note on constant-round zero-knowledge proofs for NP. In *Theory of Cryptography Conference*. Springer, 191–202.
- [38] Abhishek Roy and Sunil Karforma. 2012. A Survey on digital signatures and its applications. In *Journal of Computer and Information Technology*, Vol. 3. 45–69.
- [39] Alexander Schneider, Christian Meter, and Philipp Hagemester. 2017. Survey on Remote Electronic Voting. In *arXiv preprint arXiv:1702.02798*.
- [40] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*. Springer, 239–252.
- [41] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*. Springer, 148–164.
- [42] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
- [43] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. 2018. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *arXiv preprint arXiv:1802.07344* (2018).
- [44] Jia Yu, Fanyu Kong, Xiangguo Cheng, Rong Hao, and Guowen Li. 2014. One forward-secure signature scheme using bilinear maps and its applications. In *Information Sciences*, Vol. 279. Elsevier, 60–76.
- [45] Lingyue Zhang, Huilin Li, Yannan Li, Yong Yu, Man Ho Au, and Baocang Wang. 2019. An efficient linkable group signature for payer tracing in anonymous cryptocurrencies. *Future Generation Computer Systems* 101 (2019), 29–38.

A DEFINITIONS

Computational Indistinguishability: Let $X = \{(in, \lambda)\}_{in \in \{0,1\}^*, \lambda \in \mathbb{N}}$ and $Y = \{(a, \lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ be two series of random variables which are indexed with in and λ where in is the input and λ is the security parameter. The two distributions are computationally indistinguishable i.e., $X \equiv_c Y$ if the following holds: $\forall D$ (a non-uniform polynomial-time distinguisher), \exists a negligible function $negl(\cdot)$ s.t. $\forall in \in \{0, 1\}^*$ and $\forall \lambda \in \mathbb{N}$ [24]:

$$|Pr[D(X(in, \lambda)) = 1] - Pr[D(Y(in, \lambda)) = 1]| \leq negl(\lambda) \quad (32)$$

Secure Multi-Party Computation [20]: Consider function $F(in_1, \dots, in_N)$ $= (f_1(in_1, \dots, in_N), \dots, f_N(in_1, \dots, in_N))$ that receives inputs in_i from i^{th} party to whom delivers $f_i(in_1, \dots, in_N)$. F shall be run by a trusted third party. We refer to such execution as the *IDEAL* world. Assume γ^F is a multi-party protocol that computes F . The execution of γ^F by the interaction of parties constitutes the *REAL* world. γ^F is said to securely realize F if the following holds. That is, for every PPT adversary A in protocol γ^F with auxiliary input $aux \in \{0, 1\}^*$ and controlling parties specified in P_c , there exists a PPT simulator Sim for the ideal functionality F , that \forall security parameter λ :

$$\{IDEAL_{F, Sim(aux), P_c}(in_1, \dots, in_N, \lambda)\} \equiv_c \{REAL_{\gamma^F, A(aux), P_c}(in_1, \dots, in_N, \lambda)\} \quad (33)$$

$IDEAL_{F, Sim(aux), P_c}(in_1, \dots, in_N, \lambda)$ represents the output of parties in interaction with ideal functionality F while Sim is controlling parties specified in set P_c . Similarly, $REAL_{\gamma^F, A(aux), P_c}(in_1, \dots, in_N, \lambda)$ asserts the output of the parties interacting in protocol

γ^F .

Hybrid Model: Assume θ is a multiparty protocol that makes use of a sub-protocol γ^F . γ^F in turn securely realizes the ideal functionality F . The hybrid model allows proving the security of θ by replacing γ^F with F . As such, for any execution of γ^F in the proof, parties contact a trusted third party running the ideal functionality F . This would be called F -hybrid model [24].

Sigma protocol: A Σ protocol is a three rounds proof system (P, V) for a relation R which satisfies the following properties [24]:

- **Completeness:** An honest prover P holding a private input w , where $(x, w) \in R$, can always convince an honest verifier V .
- **Special soundness:** There exists a polynomial time machine A that for every pair of accepting transcripts (a, e, z) and (a, e', z') (where $e \neq e'$) of an statement x , A extracts witness w s.t. $(x, w) \in R$
- **Special honest verifier zero knowledge:** There exists a PPT machine S which given statement x and e can generate an accepting transcript (a, e, z) whose distribution is the same as the transcript of the real interaction of P and V . More formally, $\forall (x, w) \in R$ and $e \in \{0, 1\}^t$

$$\{S(x, e)\} \equiv_c \{(P(x, w), V(x, e))\} \quad (34)$$

The output of simulator S is denoted by $\{S(x, e)\}$. $\{(P(x, w), V(x, e))\}$ indicates the output transcript of an execution between P (holding inputs x and w) and V (with inputs x and random tape e).

Zero-knowledge proof of knowledge from Σ protocols: Following the method given in [24, 37], it is proven that one can efficiently construct a zero-knowledge proof of knowledge (ZKPOK) system from any sigma protocol. We refer to [24] for more details of such construction. Applying this method on a Σ protocol Π (defined for the relation R) will result in construction that securely realizes the ideal functionality F_{Π}^R (defined in Equation 35) in the presence of malicious prover and verifier.

$$F_{\Pi}^R((x, w), x) = (\perp, R(x, w)) \quad (35)$$

where x refers to the statement whose correctness is to be proven and w indicates the witness. The ideal functionality F_{Π}^R that is run by a trusted party, receives a common input x from the prover and the verifier. Also, prover inputs F with the private input w from the prover. F_{Π}^R outputs to the verifier whether x and w fit into the relation R .

B PROOF OF INVITATION CORRECTNESS

B.1 Soundness:

Consider two valid transcripts $(A, B = (B_1, B_2), C, e, Z_1, Z_2, Z_3)$ and $(A, B = (B_1, B_2), C, e^*, Z_1^*, Z_2^*, Z_3^*)$, where $e \neq e^*$, $Z_1 \neq Z_1^*$, $Z_2 \neq Z_2^*$, and $Z_3 \neq Z_3^*$, then we extract δ_i , r and s_i as explained below. Since both transcripts are accepting we have $A \cdot \gamma_i^e = g^{Z_1}$ and $A \cdot \gamma_i^{e^*} = g^{Z_1^*}$. We divide both sides of equalities and obtain

$$g^{Z_1 - Z_1^*} = \gamma_i^{e - e^*} = g^{s_i \cdot (e - e^*)} \pmod{p} \quad (36)$$

Thus, $Z_1 - Z_1^* \equiv s_i \cdot (e - e^*) \pmod{q}$. It follows that $s_i = \frac{Z_1 - Z_1^*}{e - e^*}$. To extract δ_i and r we proceed as follows. We know that $B_1 \cdot e \delta_{i,1}^e = \omega^{Z_2} \cdot h^{Z_3}$ as well as $B_1 \cdot e \delta_{i,1}^{e^*} = \omega^{Z_2^*} \cdot h^{Z_3^*}$. Dividing both sides of

equalities results in

$$\begin{aligned} e\delta_{i,1}^{e-e^*} &= \omega^{Z_2-Z_2^*} \cdot h^{Z_3-Z_3^*} \pmod{p} \\ \omega^{\delta_i(e-e^*)} \cdot h^{r(e-e^*)} &= \omega^{Z_2-Z_2^*} \cdot h^{Z_3-Z_3^*} \pmod{p} \end{aligned} \quad (37)$$

As such, it follows that $\delta_i = \frac{Z_2-Z_2^*}{e-e^*} \pmod{q}$ and $r = \frac{Z_3-Z_3^*}{e-e^*} \pmod{q}$.

B.2 Special honest verifier zero knowledge:

We construct a PPT simulator Sim which is given $\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega$ and e and generates an accepting transcript. It selects Z_1, Z_2, Z_3 at random and constructs $A = \frac{g^{Z_1}}{\gamma_i^e} \pmod{p}$ and $B_1 = \frac{\omega^{Z_2} \cdot h^{Z_3}}{e\delta_{i,1}^e} \pmod{p}$ and $B_2 = \frac{g^{Z_3}}{e\delta_{i,2}^e} \pmod{p}$ and $C = \tau_i^{-e} \cdot B \cdot e\delta_{i,1}^e \cdot h^{-Z_3} \cdot \omega^{Z_1} \pmod{p}$. Sim outputs $(A, B = (B_1, B_2), C, e, Z_1, Z_2, Z_3)$. It is immediate that the probability distribution of $(A, B, C, e, Z_1, Z_2, Z_3)$ and a real conversation between honest prover and honest verifier are identical.

B.3 Zero-knowledge POIC (ZKPOIC):

We apply the method given in [24] to our Σ protocol to convert it to a zero-knowledge proof system. Let F_{POIC}^R (given in Equation 38) demonstrate the security guarantees of the resultant ZKPOIC over the relation R that we defined in Equation 8.

$$F_{POIC}^R((X, W), X) = (\perp, R(X, W)) \quad (38)$$

F_{POIC}^R shall be run by a trusted third party. X refers to the statement whose correctness is to be proven, i.e., $X = (\tau_i, e\delta_i, \gamma_i, \omega)$ contains the content of an individual invitation letter $(\tau_i, e\delta_i)$ as well as the commitment to the inviter's master share, i.e., γ_i , and the token ω . The witness W , which is only known to the prover, is (s_i, r, δ_i) . The ideal functionality F_{POIC}^R receives a common input X from the prover and the verifier as well as the private input W from the prover. F_{POIC}^R outputs to the verifier whether X and W fit into the relation R .