Anonyma: Anonymous Invitation-Only Registration in Malicious Adversarial Model

Sanaz Taheri Boshrooyeh, Alptekin Küpçü, and Öznur Özkasap

Department of Computer Engineering, Koç University, İstanbul, Turkey
`{staheri14,akupcu,oozkasap}@ku.edu.tr`

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 to be authenticated, who verifies that the invitations are issued by legitimate members. This causes the administrator being aware of who is invited by whom. However, the inviter-invitee relationship is privacy-sensitive information whose exposure can lead to an inference attack 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 and unforgeability of invitations against a *malicious adversary.* Our design only incurs 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 credential for the newcomer to be able to act as an inviter. We additionally design *AnonymaX*, an anonymous cross-network invitation-based system where the invitations issued by the members of one system can be used for registering to another system.

Keywords: Invitation-Based System, Anonymity, Unforgeability, Integrity, Cross-Network Invitation, Malicious Adversary

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 users[1].

[1] https://pieregister.com/features/invitation-based-registrations/

In a nutshell, an invitation-only system is comprised of an *administrator/server*, a set of members, who act as *inviters*, and a new user, called the *invitee*, who wants to join the system. Figure 1 illustrates the parties and their interaction. The new user (i.e., invitee) can register to the system by being invited by the existing members. Successful registration relies on having a certain number (i.e., t) of invitations from distinct members. The invitee collects the invitations and hands them over to the administrator, who checks whether the invitations are issued by legitimate members. If so, he accepts the registration request and allows the invitee in. Additionally, the administrator may issue credentials for the invitee to be able to start inviting others.

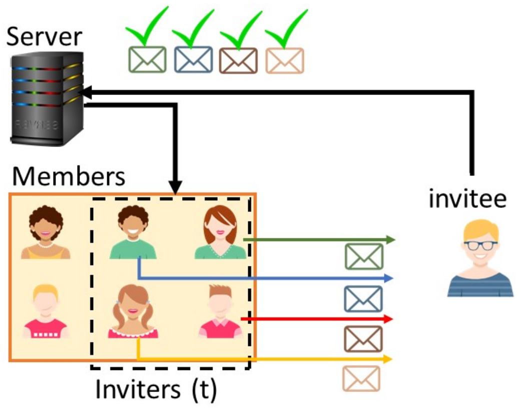

Fig. 1. The sample workflow of an invitation-only registration system. The server issues credentials to the members to generate invitations. The invitee collects t invitations and sends them to the server. Then, the server accepts or rejects the invitee's request for registration by verifying that each invitation is issued by a valid current member (resulting in knowing who is invited by whom).

Google initiated this invitation-only policy when deploying services such as Google Inbox, Orkut, and Google Wave[2]. 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 members[4]. Similarly, messengers such as *WhatsApp*[5] and *Telegram*[6] offer private groups running on the invite-only basis. The similar approach is sought in *PIE Register*[7] where it enables users to set up exclusive websites whose contents are only

[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] ttps://faq.whatsapp.com/en/android/26000123/?category=5245251
[6] https://telegram.org/tour/groups
[7] https://pieregister.com/features/invitation-based-registrations.

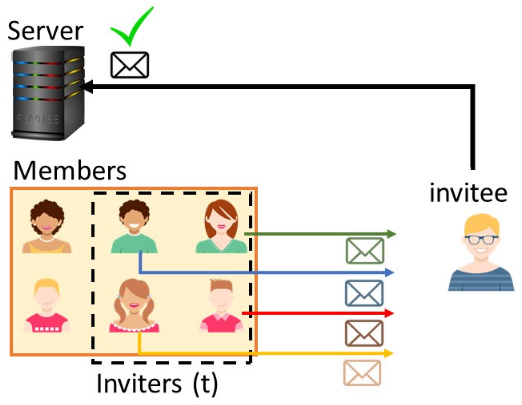

Fig. 2. *Anonyma* overview. The invitee receives the individual invitations and aggregates them into a unified letter to be sent to the server. The server authenticates the letter without knowing the identity of the inviters.

2

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 [14, 23, 5]. A backup authentication method is used when the user fails to pass the primary authentication, e.g., forgetting the password [14]. 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.

Apart from the benefits of invitation-only systems, they are prone to inference attacks as explained next. 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 [22, 7, 13]. This leakage is known as inference attack. Due to this issue, inviter-invitee relation counts as privacy-sensitive information.

Related Work: Hiding the inviter-invitee relationship has similarly been addressed by researchers in the context of electronic voting systems and threshold ring-based signature schemes. However, those proposals suit their unique settings and become inefficient when utilized for the invitation-only registration scenario. The issue in threshold ring-based signature schemes is that the computation complexity for invitation creation (at the user side) as well as invitation verification (at the server-side) is $O(N)$ where N is the total number of members in the system [6, 24, 21, 4]. A similar issue applies to the e-voting systems [1, 26] where all the existing members are required to get involved in every single registration. Added to this is the size of the invitation, which grows with the number of registered members (more precisely, each invitation contains N group elements, where N is the total number of members).

In our prior work, Inonymous [3], we address the problem of inference attack in invitation-based systems by proposing a protocol by which an invitee is able to authenticate herself to the group administrator without disclosing the identity of her inviters, thus protecting the inviter-invitee relationship (or inviter anonymity). Additionally, Inonymous ensures that an invitee with an insufficient number of inviters would not be able to convince the administrator and register to the system. This feature is called invitation unforgeability. Inonymous guarantees the above features under an *honest but curious* adversarial model, namely, all the parties shall follow the protocol descriptions honestly. However, a corrupted administrator, by disregarding the protocol description, can damage inviter anonymity and learn who is invited by whom. To cope with this attack, we propose *Anonyma*, which extends Inonymous to withstand the *malicious adversarial model* where parties may deviate from protocols' descriptions.

***Anonyma* design challenges:** Extension of Inonymous to *Anonyma* comes with challenges which we present next. In Inonymous, the general idea is that each group member receives some private credentials from the administrator, and then later uses these credentials to generate an invitation for the invitees. The invitee must aggregate the individual invitations to remove any identifiable information about the individual inviters. If all the parties follow the protocols correctly, the final aggregate of invitations that is given to the administrator would not leak any information regarding the individual inviters; hence inviter anonymity is preserved. Anonymity holds even if the administrator colludes with a subset of inviters. However, in a malicious adversarial model, the inviters may integrate garbage values (instead of valid credentials) in their invitations that would falsify the result of aggregate. Given the falsified aggregate, the administrator is able to identify the exact inviters. As such, the invitee must ensure that each inviter generates the invitation following the exact protocol and using the correct credentials. At the same time, the correctness of users' credentials is defined by the administrator who can also be corrupted (who may issue invalid credentials to violate inviter anonymity). Thus, two challenges must be addressed, firstly, a method to ensure that the administrator is issuing valid credentials and secondly, providing a way for the invitee to be convinced that the inviters are using the genuine credentials and follow the designated protocol correctly. Furthermore, regarding the unforgeability of invitations, *Anonyma* extends the adversarial model of Inonymous where the communication channels between all the inviters and invitees are considered to be insecure. As such, the challenge toward providing unforgeability is that an adversary (an invitee with an insufficient number of inviters), who eavesdrops the channels, must be unable to craft a valid invitation letter for herself and register to the system successfully.

***Anonyma*:** An overview of *Anonyma* is depicted in Figure 2. *Anonyma* consists of three entities: A server (administrator), existing members (inviters), and a newcomer (invitee). 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 example, 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 combines the invitations into a single invitation letter and submits it to the server. If the invitation is verified by the server, the invitee's registration request is accepted. The verification does not rely on any interaction between the inviters and the server. Upon successful registration, the server can decide to issue credentials for the new user to enable him to invite others. In particular, the ability to make invitations is up to the administrator who can either reserve this right to himself or share it with the members of his choice. In *Anonyma*, the confidentiality of the inviter-invitee relationship (i.e., inviter-anonymity) is protected against both the server and the other members including inviters of the same invitee. Together with this confidentiality, *Anonyma* guarantees invitation unforgeability where a malicious invitee cannot convince the server unless he has threshold many legitimate invitations. We provide formal definition for *inviter anonymity* and *invitation unforgeability* in Section 7. Our definitions are slightly different from Inonymous in the sense that we allow the adversary to spoof the communication channels between inviters and invitee, hence gaining more information. A formal game-based proof of security against an *active/malicious adversary* (who disregards the protocol instructions and acts arbitrarily) is also provided.

Anonyma preserves the inviter anonymity, i.e., hides the invitee-inviter relationship, only for the registration. 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.

Anonyma imposes only $O(t)$ overhead on the invitee, and constant number of group exponentiations on the inviters (for the invitation creation) and the system administrator (for the authentication of new registration). As such, *Anonyma* provides better efficiency compared to prior studies whose computation overheads for the invitee and the server are linear in the total number of existing members. Furthermore, in *Anonyma*, the server is able to efficiently, and without re-keying the existing members, generate credentials for a newcomer to empower him for inviting others. This is significantly better than the prior proposals where the server has to carry out $O(N)$ communication overhead to submit some information about the newly registered user to each one of the current members. Moreover, unlike the related work, the size of the invitation is $O(1)$ where each invitation embodies only two group elements regardless of the system size. Additionally, the complexity of communication among all the parties are also of constant overhead.

Additionally, we propose *AnonymaX*, an anonymous cross-network invitation-based system, which can be of independent interest. In the cross-network design, a user joins one system, e.g., Twitter, by obtaining invitations from members of another network, e.g., Facebook. The cross-network design is beneficial especially to bootstrap a system, for example in the case where a research group wants to hire qualified researchers from another group, where a qualified researcher is the one with enough recommendations (invitations) from his group. We also prove that *AnonymaX* preserves inviter anonymity and invitation unforgeability against a malicious adversary.

The summary of our contributions in the present work is as follows:

- *Anonyma* is the **first** anonymous invitation-based system that maintains *inviter anonymity* and *invitation unforgeability* under a malicious adversarial model. That is, all the involved entities may disregard the protocol descriptions and act as they desire, yet they would be unable to compromise the mentioned security objectives.
- We provide **formal security definitions and proofs** for inviter anonymity and invitation unforgeability. Our security definition for invitation unforgeability is the extension of the Inonymous definition [3] and indeed considers a more powerful adversary. Namely, we additionally consider an insecure network connection among inviters and invitees (while exchanging invitations), which would allow the adversary to eavesdrop the channels and collect more information.
- *Anonyma* is **efficiently scalable** in terms of the number of inviters. That is, the server can issue credentials to the new members (without re-keying other existing members) to be immediately able to invite others.
- *Anonyma* outperforms its counterparts concerning running time complexity. Prior studies' running time depends on the total number of system members, whereas *Anonyma* only incurs the overhead of $O(t)$ (t is the required number of inviters) to the invitee and $O(1)$ to the server and the inviters.
- We propose the **first cross-network** anonymous invitation-based system called *AnonymaX*, where the possibility of inter-network invitation is provided, with security against malicious adversaries.

2 Model

Anonyma is composed of three types of entities: a server, a set of existing members (inviters), and a new user (invitee) who is willing to join. The server sets up the system parameters, generates and distributes some secret values among

users, and administers user registrations. For successful registration, each newcomer needs to obtain a threshold many (denoted by t) invitations from the existing members. The inviters exchange the invitations with the invitee out of band, e.g., via a messaging application. After the collection of t invitations, the invitee removes the identifiable information from the individual invitations (through aggregation) and submits a final invitation letter to the server. The server authenticates the letter. Upon successful authentication, the server lets the new user register and can issue credentials to empower him to act like an inviter and make invitations. Note that the system shall start by having at least t initial members who begin inviting outsiders. These initial members are given credentials directly from the server. In the Google Inbox example, the initial members (account holders) can be the employees of Google. In our system, we assume all the entities communicate through a secure and authenticated channel.
Security Objectives: Our security objective is two-fold: *Inviter Anonymity* and *Invitation Unforgeability*, which are explained below. In *Anonyma*, we aim to satisfy both objectives in the *malicious adversarial model*, where the entities may deviate from protocol specifications.

1. **Inviter anonymity**: As we discussed in Section 1, due to the inference attack possibility, the inviter-invitee relationship must be treated as privacy-sensitive information. Thus, by inviter anonymity, we aim to hide who is invited by whom. This relation should be protected against both the server and other inviters of the same invitee (as they might be also curious to learn the identity of other inviters). Putting these together, we assume that the adversary against the inviter anonymity may get to control the server and $t-1$ inviters of an invitee and aims at determining the identity of the remaining non-colluding inviter. Please note that the invitee is concerned about his privacy, and hence has no incentive to expose the identity of his inviters to others. We formally propose a security definition for inviter anonymity in Section 7.1. Our definition also implies *between-inviter anonymity*, which refers to the fact that the anonymity of the invitee-inviter relationship holds even against the inviters of the same invitee.
2. **Invitation unforgeability**: The invitation unforgeability indicates that an invitee whose number of inviters ($t^{'}$) is less than the threshold t (i.e. $t^{'} < t$) should not be able to register to the system. Trivially, if the invitee already has t inviters, i.e. $t^{'} = t$, then he is an eligible person and can make a valid registration. We propose a security definition for invitation unforgeability in Section 7.2. That definition embodies the following security properties:
   - **Non-exchangability**: This means that invitations issued for a particular invitee are not reusable for another user. Otherwise, the current registered users can exchange their past invitations (by which they got invited to the system) with others and cause ineligible outsiders to join the system.
   - **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.

3 Notations, Definitions, and Preliminaries

Notation: We refer to a Probabilistic Polynomial Time entity as PPT. $x \in_R X$ 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. TTP stands for Trusted Third Party.
Negligible Function: Function f is negligible if for $\forall p(.)$ where $p(.)$ is polynomial, there exists integer N s.t. for every $n > N$, $f(n) < \frac{1}{p(n)}$.
Computational Indistinguishability: Let $X = \{(in, \lambda)\}_{in \in \{0,1\}^*, \lambda \in \mathbb{N}}$ and $Y = \{(a, \lambda)\}_{in \in \{0,1\}^*, \lambda \in \mathbb{N}}$ 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 $negl(.)$ s.t. $\forall in \in \{0,1\}^*$ and $\forall \lambda \in \mathbb{N}$ [16]:

$$|Pr[D(X(in, \lambda)) = 1] - Pr[D(Y(in, \lambda)) = 1]| \leq negl(\lambda) \tag{1}$$

Secure Multi-Party Computation [12]: Consider function $F(in_1, ..., in_N) = (f_1(in_1, ..., in_N), \cdots, f_N(in_1, ..., in_N))$ that receives inputs in_i from i^{th} party to whom delivers $f_i(in_1, ..., in_N)$. 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:

$$\{IDEAL_{F,Sim(aux),P_c}(in_1, ..., in_N, \lambda)\}\} \equiv_c \{REAL_{\gamma^F,A(aux),P_c}(in_1, ..., in_N, \lambda)\} \tag{2}$$

$IDEAL_{F,Sim(aux),P_c}(in_1, ..., in_N, \lambda)$ represents the output of parties in interaction with ideal functionality F while Sim is controlling parties specified in set P_c. Similarly, $REAL_{\gamma^F,A(aux),P_c}(in_1, ..., in_N, \lambda)$ 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 rusted third party running the ideal functionality F. This would be called F-hybrid model [16].
Sigma protocol: A Σ protocol is a three rounds proof system (P, V) for a relation R which satisfies the following properties [16]:

- **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 $e \in \{0, 1\}^t$

$$\{S(x, e)\} \equiv_c \{(P(x, w), V(x, e))\} \tag{3}$$

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).

Pseudo Random Generator [2]: A deterministic polynomial time function $P : \{0, 1\}^m \to \{0, 1\}^{l(m)}$ (where $l(.)$ 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 $negl(.)$ such that:

$$|Pr[x \leftarrow \{0, 1\}^m : D(P(x)) = 1] - Pr[y \leftarrow \{0, 1\}^{l(m)} : D(y) = 1]| = negl(m) \tag{4}$$

(t,n)-Shamir Secret Sharing Scheme: The (t,n)-Shamir secret sharing scheme [31] 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:

$$f(x) = \sum_{i=0}^{t-1} a_i \cdot x^i \tag{5}$$

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 jth 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.

$$S = f(0) = \sum_{i=1}^{t} s_i \cdot B_i \tag{6}$$

where B_is are Lagrange coefficients defined as

$$B_i = \sum_{i=1}^{t} s_i \prod_{j \neq i}^{j=1:t} \frac{j}{j - i} (\text{mod q}) \tag{7}$$

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 $KeyGen$, q 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 [11], $\otimes$ 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 [28] 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 $n4lg(.)$ s.t. the following holds [18]:

$$\begin{aligned} &Pr[(sk, vk) \leftarrow SGen(1^\lambda); (m, \sigma) \leftarrow A^{Sign_{sk}(.)}(vk) \\ &\quad \text{s.t. } m \notin Q \textit{ and } SVrfy_{vk}(m, \sigma) = accept] = negl(\lambda) \end{aligned} \tag{8}$$

$A^{Sign_{sk}(.)}$ 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 from Σ protocols: Following the method given in [16, 27], it is proven that one can efficiently construct a zero-knowledge proof of knowledge (ZKPOK) system from any sigma protocol. We refer to [16] 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^R_Π (defined in Equation 9) in the presence of malicious prover and verifier.

$$F^R_\Pi((x, w), x) = (\perp, R(x, w)) \tag{9}$$

where x refers to the statement whose correctness is to be proven and w indicates the witness. The ideal functionality F^R_Π 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.

Zero-knowledge Proof of Knowledge of Discrete Logarithm (ZKPODL): This proof system was initially introduced by Schnorr [18] 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 $x \in Z_q$ s.t. $x = DL_g(\omega)$ (DL stands for discrete logarithm). We apply the method given in [16] to the Schnorr protocol to convert it to a zero-knowledge proof system. We refer to the resultant protocol by $ZKPODL((G, q, g, \omega), r)$. Let F_{PODL}^{R} (given in Equation 10) demonstrate the security guarantees of the $ZKPODL$ protocol over the relation R that is given in Equation 11. F_{PODL}^{R} shall be run by a trusted third party. X refers to the statement whose correctness is to be proven i.e., $X = (G, q, g, \omega)$ and the witness $W = x$, which is only known to the prover. The ideal functionality F_{PODL}^{R}, which is run by a TTP, receives a common input X from the prover and the verifier as well as the private input W from the prover. F_{PODL}^{R} outputs to the verifier whether X and W fit into the relation R.

$$F_{POIC}^{R}((X, W), X) = (\perp, R(X, W)) \tag{10}$$

$$R = \{((G, q, g, \omega), x) |\ g^x = \omega\ (mod\ p)\} \tag{11}$$

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 [15] 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 $\alpha \in Z_q$ [8].

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 [32]

- 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: $e(g_1, g_1) \neq 1$.

Computational Diffie-Hellman Assumption [10]: 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:

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

4 Construction

Anonyma is comprised of three main entities, namely a *server*, a set of existing members who shall act as *inviters*, and newcomers/*invitees* wishing to become a member of the system. The general interaction between the parties is illustrated in Figure 3. *Anonyma* consists of six 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 4.1 followed by the full construction in Section 4.2. Throughout the paper, we assume that all the parties communicate via secure and authenticated channels.

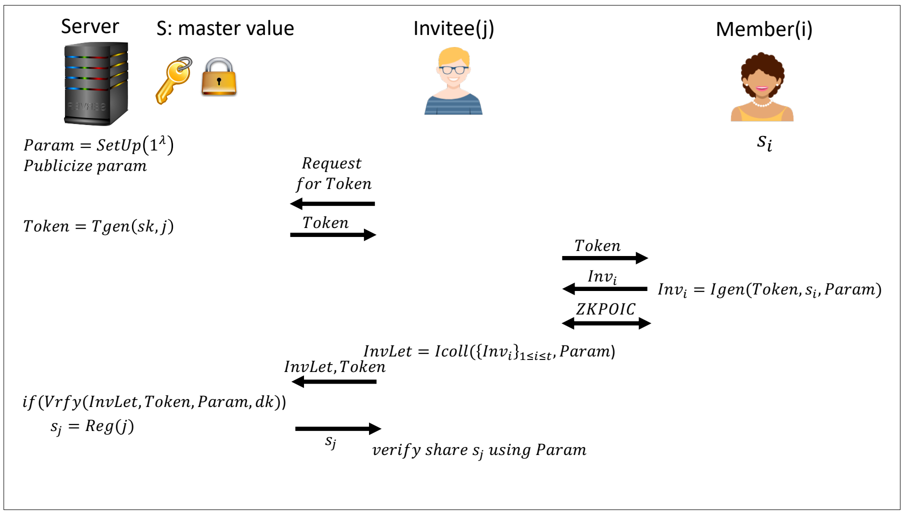
Server
S: master value
Invitee(j)
Member(i)
s_i
$Param = SetUp(1^\lambda)$
Publicize param
Request for Token
$Token = Tgen(sk, j)$
Token
Token
Inv_i
$Inv_i = Igen(Token, s_i, Param)$
ZKPOIC
$InvLet = Icoll(\{Inv_i\}_{1 \le i \le t}, Param)$
InvLet, Token
$if(Vrfy(InvLet, Token, Param, dk))$
$s_j = Reg(j)$
s_j
verify share s_j using Param

Fig. 3. Parties' interaction in *Anonyma*.

4.1 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)-Shamir secret sharing scheme. For the shares to be verifiable (the member can verify whether or not his piece is valid), the server publicizes the commitment to the selected polynomial function.

- $Tgen$: 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 issue an invitation for their intended invitee. Invitations issued for a particular token cannot be used for another token. This way, we guarantee the non-exchangeability of the invitations.
- *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: a masked version of the inviter's share s_i, 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 homomorphically gets 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.

4.2 Full Construction:

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 publicized.
- A signature scheme $Sig = (SGen, Sign, SVrfy)$. The signature and verification keys (sk, vk) are generated according to $SGen$. vk is publicized.
- A pseudo random generator PRG:$\{0,1\}^\lambda \rightarrow Z_q$
- A master value $S \leftarrow Z_q$
- A randomly chosen polynomial function $f(y) = a_{t-1}y^{t-1} + ... + a_1 y + a_0$ of degree $t-1$ whose coefficients $a_1, ..., a_{t-1}$ 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 ith user.
- The server publicizes $F_0 = g^{a_0}, F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$ as the commitment to the selected function f. Given $F_0, \cdots, F_{t-1}$, the computation of commitment on $f(i)$ for any i is immediate as given in Equation 12. We will use γ_i to indicate g^{s_i}.

$$\gamma_i = \prod_{j=0}^{t-1} F_j^{i^j} = g^{a_0} \cdot g^{a_1 \cdot i} \cdots g^{a_{t-1} \cdot i^{t-1}} = g^{a_0 + a_1 \cdot i + \cdots + a_{t-1} \cdot i^{t-1}} = g^{f(i)} = g^{s_i} \tag{12}$$

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

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 4.1. 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.

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 4.2. 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

Algorithm 4.1: Tgen [Server]

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

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, or for short *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 followings, 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 [16, 27].

Algorithm 4.2: Igen [Inviter]

```
Input: $Token = (\eta, j, \omega)$, $s_i$, $Param$
Output: $Inv_i / \perp$
1 if $Svrfy_{vk}(\eta, j||\omega)$=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 $F_0, ..., F_{t-1}$ as explained in Equation 12.
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 13.

$$\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r = \omega^{\delta_i + s_i} \cdot \omega^{-\delta_i} \cdot h^{-r} \cdot h^r = \omega^{s_i} \tag{13}$$

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 14. In our protocol, we incorporate the Shnorr protocol [18] for the proof of discrete logarithm knowledge (first anf third statement), proof of plaintext knowledge as proposed in [15] (for the second statement), and the proof of discrete logarithm equality [8] (for the fourth statement).

$$\begin{aligned}
&R = \{((\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega), (s_i, r, \delta_i))| \\
&DL_g(\gamma_i) = s_i \ \wedge \\
&e\delta_{i,1} = \omega^{\delta_i} \cdot h^r \ \wedge \\
&DL_g(e\delta_{i,2}) = r \ \wedge \\
&DL_\omega(\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r) = DL_g(\gamma_i) = s_i\}
\end{aligned} \tag{14}$$

$$\begin{array}{lcl}
\text{Prover } (s_i, r, \delta_i) & & \text{Verifier} \\
\hline
s', r', \delta' \leftarrow Z_q & & \\
A = g^{s'} & & \\
B_1 = \omega^{\delta'} \cdot h^{r'} & & \\
B_2 = g^{r'} & & \\
C = \omega^{s'+\delta'} & \xrightarrow{A, B=(B_1,B_2), C} & \\
 & \xleftarrow{e} & \\
Z_1 = s' + e \cdot s_i & & \\
Z_2 = \delta' + e \cdot \delta_i & & \\
Z_3 = r' + e \cdot r & & \\
 & \xrightarrow{Z_1, Z_2, Z_3} & \\
 & & \text{if } (A \cdot \gamma_i^e == g^{Z_1} \\
 & & \wedge B_1 \cdot e\delta_{i,1}^e == \omega^{Z_2} \cdot h^{Z_3} \\
 & & \wedge B_2 \cdot e\delta_{i,2}^e == g^{Z_3} \\
 & & \wedge C \cdot \tau_i^e \cdot B^{-1} \cdot e\delta_{i,1}^{-e} \cdot h^{Z_3} == \omega^{Z_1}) \\
 & & \text{Accept}
\end{array}$$

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).

Completeness: To prove that completeness holds, observe that if the prover P follows the protocol honestly, then due to the Equations 15, 16, 17, and 18, the verifier V accepts.

$$A \cdot \gamma_i^e = g^{s'} \cdot (g^{s_i})^e = g^{s'+e\cdot s_i} = g^{Z_1} \tag{15}$$

$$B_1 \cdot e\delta_{i,1}^e = (\omega^{\delta'} \cdot h^{r'}) \cdot (\omega^{\delta} \cdot h^r)^e = \omega^{\delta'+e\cdot\delta_i} \cdot h^{r'+e\cdot r} = \omega^{Z_2} \cdot h^{Z_3} \tag{16}$$

$$B_2 \cdot e\delta_{i,2}^e = (g^{r'}) \cdot (g^r)^e = g^{r'+e\cdot r} = g^{Z_3} \tag{17}$$

$$\begin{aligned} &C \cdot \tau_i^e \cdot B^{-1} \cdot e\delta_{i,1}^{-e} \cdot h^{Z_3} = \\ &(\omega^{s'+\delta'}) \cdot (\omega^{e\cdot s_i + e\cdot\delta_i}) \cdot (\omega^{-\delta'} \cdot h^{-r'}) \cdot (\omega^{-e\cdot\delta_i} \cdot h^{-e\cdot r}) \cdot h^{r'+e\cdot r} = \\ &\omega^{(s'+e\cdot s_i)} = \omega^{Z_1} \end{aligned} \tag{18}$$

We prove the special soundness and special honest verifier zero knowledge properties in Section A. We additionally present the security properties of a zero-knowledge proof system for POIC that is achieved using the method given in [16, 27].

Invitation Collection: Upon receipt of t invitations, the invitee runs the Invitation Collection ($Icoll$) procedure as indicated in Algorithm 4.3. 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 values are the Lagrange coefficients (as defined in Equation 7) 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 19 and 20, we expand the result of T and $e\Delta$, which leads to the following observations.

$$\begin{aligned} T = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \tau_i^{B_i} = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \omega^{B_i\cdot s_i + B_i\cdot\delta_i} &= \omega^{\delta^* + \sum_{i=1}^{t} B_i\cdot s_i + \sum_{i=1}^{t} B_i\cdot\delta_i} \\ &= \omega^{S+\delta^*+\sum_{i=1}^{t} B_i\cdot\delta_i} = \omega^{S+\Delta} \end{aligned} \tag{19}$$

$$\begin{aligned} e\Delta = Enc_{ek}(\omega^{\delta^*}). \prod_{i=1}^{t} e\delta_i^{B_i} &= Enc_{ek}(\omega^{\delta^*}). \prod_{i=1}^{t} Enc_{ek}(\omega^{B_i\cdot\delta_i}) \\ &= Enc_{ek}(\omega^{\delta^*+\sum_{i=1}^{t} B_i\cdot\delta_i}) = Enc_{ek}(\omega^{\Delta}) \end{aligned} \tag{20}$$

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 7 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 4.3: 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  $\delta^* = PRG(r)$
3  $T = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \tau_i^{B_i}$
4  $e\Delta = Enc_{ek}(\omega^{\delta^*}) \cdot \prod_{i=1}^{t} e\delta_i^{B_i}$
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 4.4. As the first step, the server authenticates the $Token$, i.e., whether it is signed under 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.

```
Algorithm 4.4: Ivrfy [Server]
   Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega)$, $Param$, $dk$
   Output: $reject/accept$
1  if $Svrfy_{vk}(\eta, j||\omega)$=$accept$ then
2      $\omega^\Delta = Dec_{dk}(e\Delta)$
3      if $\omega^S \cdot \omega^\Delta = T$ then
4          return accept
```

Registration: The server invokes the registration procedure (Algorithm 4.5) for users who pass the verification phase (Algorithm 4.4). The input to Algorithm 4.5 is the index j of the newcomer, and the output is a Shamir share s_j of the master value S, where s_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 $Token = (\eta, j, \omega)$. The server delivers s_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 12) against its own share, i.e., g^{s_j}. If they are equal, the user accepts and stores the share.

Algorithm 4.5: Reg [Server]

```
  Input: $j$
  Output: $s_j$
1 $s_j = f(j)$
```

5 *AnonymaX*: Anonymous Cross-Network Invitation-Based System

Consider the situation where one system, e.g. Twitter, offers a special service for the users of another system, e.g. Facebook. We name Twitter as the *registration* network, i.e. the network serving a special service, whereas Facebook is called the *inviter network* whose users will benefit from the services offered by the *registration* network. A user of the *inviter* network is served by the *registration* network upon convincing the *registration* server on being invited by an adequate number of inviters from the *inviter* network.

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 *inviter* user wants to join the *registration* network, the *inviter* 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 *inviter* server.

An alternative approach proposed by Inonymous [3] (our prior work), is that the *inviter* server would publicize 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 own service. As such, *registration*$_1$ can reuse the invitation letters of his own 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 *inviter* network with the master value S publicizes g^S as a part of its set of parameters $Param_{guest}$. Note that the description of group G is only generated by the *inviter* server and is used by other *registration* servers. On the other side, the *registration* networks denoted by S_j, $1 \leq j \leq N$, announce their $Param_{S_j}$ to be the pair of encryption keys ek_{S_j} and signature verification keys vk_{S_j}, i.e. $Param_{S_j} = (ek_{S_j}, vk_{S_j})$. The corresponding decryption key dk_{S_j} as well as the signature signing key sk_{S_j} remain private at the server side. Each invitee willing to join S_j shall obtain a token from S_j. During the token generation, the *registration* server follows Algorithm 4.1 and additionally must prove in zero-knowledge that it knows the discrete logarithm of the ω embodied in the token. As such, after the issuance of a token, the *registration* server runs an instance of *ZKPODL* protocol (given in section 3) with the invitee. The modified procedure is provided in Algorithm 5.1.

Algorithm 5.1: XTgen [*registration* Server S_j]

```
Input: $Param_{guest} = (G, q, g, ek_{guest}, vk_{guest}, (F_0, \cdots, F_{t-1})), sk_{S_j}, i$
Output: $Token$
1 $r \leftarrow Z_q$
2 $\omega = g^r$
3 $\eta = Sign_{sk_{S_j}}(i||\omega)$
4 $Token = (\eta, i, \omega)$
5 $Run \quad ZKPODL((G, q, g, \omega), r)$
```

Upon a successful proof, the invitee accepts the token. The invitee needs to collect invitations from the members of the *inviter* network to be used in the registration of a particular *registration* network. Inviters issue invitation as in the regular invitation procedure given in Algorithm 4.2. However, the inviters should verify the tokens against the *registration* server verification key who has issued it. Also, the inviters shall use the encryption key of the *registration* network to encrypt their masking values. Indeed, in Algorithms 4.2 and 4.3, the inviter uses ek_j and vk_{S_j}, i.e. $Param_{S_j}$ as input. Therefore, the invitation letters received by the S_j are of the form $InvLet = (T, e\Delta)$ where $e\Delta$ is an encrypted masking value under ek_j. The *registration* server runs a different verification routine, which is given in Algorithm 5.2. 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 5.2 and Algorithm 4.4 is at the second verification step i.e., line 3. The correctness holds by the bilinearity of the bilinear map e, as in Equation 21.

$$\begin{aligned} e(\omega, g^S) \cdot e(\omega^{\Delta}, g) &= e(\omega, g)^S \cdot e(\omega, g)^{\Delta} = e(w, g)^{S+\Delta} = e(w^{S+\Delta}, g) \\ &= e(T, g) \end{aligned} \tag{21}$$

Algorithm 5.2: XIVerify [*registration* Server S_j]

```
Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega), Param_{inviter}, Param_{S_j}, dk_{S_j}$
Output: $reject/accept$
1 if $Svrfy_{vk_{S_j}}(\eta, j||\omega)$=$accept$ then
2     $\omega^{\Delta} = Dec_{dk_{S_j}}(e\Delta)$
3     if $e(\omega, g^S) \cdot e(\omega^{\Delta}, g) = e(T, g)$ then
4         return accept
```

6 Performance

6.1 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 [20] 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.

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. As such, the invitee's running time for $Icoll$ is shown in Figure 5 for the threshold values of 1-9. In this diagram, we included the time for the verification of invitations' correctness proof as part of the $Icoll$ procedure as well. In particular, the running time of $Icoll$ is dominated by $t \cdot t_{POIC_{verify}} + (t-1) \cdot t_{agg}$, where t is the threshold, $t_{POIC_{verify}}$ the time to authenticate each invitation, and t_{agg} is the time required for homomorphic aggregation of two individual invitations.

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

	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)

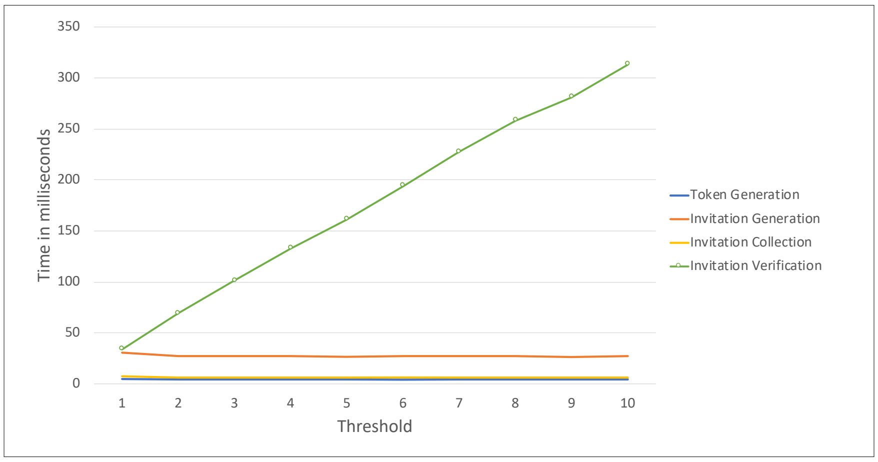

Fig. 5. The invitee's running time.

6.2 Communication Complexity

The asymptotic communication complexity among the parties is constant in the security parameter. However, we additionally measure the concrete communication complexity as the number of bits exchanged between parties. The communication complexity for the $Tgen$ protocol to transfer a Token with two group elements of size 2048 bits is 4096 bits (= 0.512 KB). For the invitation generation protocol, the inviter exchanges the invitation (consisting of 3 group elements) together with the ZKPOIC (with 8 group elements). Hence the total communication complexity of $Igen$ is 22528 bits (= 2.81 KB). The invitee submits the invitation letter (with 3 group elements) together with the token (of size 4096 bits) to the server for the sake of registration which results in 10240 bits (= 1.28 KB) communication complexity.

6.3 Storage

The storage requirement of each entity is measured based on the number of bits that the party needs to retain locally. The server holds a signature and encryption key pairs, 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.

7 Security

In this section, we provide security definitions for inviter anonymity and invitation unforgeability, and then prove the security of *Anonyma* (Sections 7.1 and 7.2). In Section 7.3, 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^{R}_{POIC} corresponding to the zero-knowledge version of POIC are provided in the Appendix, section A.

7.1 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*.

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

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

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^R_{POIC} 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: $S' = \sum_{i \in I_c} s_i.B_i + s_{u_b}.B_{u_b} = s_{u_b}.B_{u_b}$. 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 a formal proof.

Theorem 1. *Anonyma provides inviter anonymity in F^R_{POIC} hybrid model (as defined in Equation 36), 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 of A. Assume A's success probability is

$$Pr[InvAnonym_A(\lambda) \; = 1] = \frac{1}{2} + \epsilon(\lambda) \qquad (22)$$

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 $\overrightarrow{\delta} = (\delta^{'}, \delta^{''})$ and aims at specifying whether $\overrightarrow{\delta}$ 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 $\overrightarrow{\delta}$ is pseudo-random, otherwise random.

1. B is given the security parameter 1^{λ} and a vector of two values denoted by $\overrightarrow{\delta} = \{\delta^{'}, \delta^{''}\}$ s.t. $\delta', \delta'' \in Z_q$. Adversary A outputs $Param$ including ek, vk, and $(F_0, \cdots, F_{t-1})$.
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 $\prod_{j=0}^{t-1} F_j^{i^j}$.
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 $t-1$ incides $I_c \subset U_c$ to be the corrupted inviters. For every $i \in I_c$, and $Token$, A engages in $Igen$ with the callenger. A outputs $Inv_i = (\tau_i, e\delta_i)$ and contacts F^R_{POIC} with the input of $(\tau_i, e\delta_i, \gamma_i, \omega)(s_i, r_i, \delta_i)$. B acting as F^R_{POIC}, 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 14. 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 7. 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 $\delta^{''}$ (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 $\overrightarrow{\delta}$ 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 $\delta^{''}$ is a random value completely independent of inviters' indices. If $\overrightarrow{\delta}$ is a random vector then $\delta^{''}$ is also a random value from $\mathbb{Z}_q$. Therefore, in $\omega^{\delta^{''}+\Gamma}$, Γ is indeed masked with $\delta^{''}$ ($\delta^{''} + \Gamma \mod q$ 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.,

$$Pr[B(\overrightarrow{\delta} \leftarrow Z_q) = 1] = Pr[b = b^{'}] = \frac{1}{2} \tag{23}$$

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

$$Pr[r \leftarrow \{0,1\}^{\lambda} : B(\overrightarrow{\delta} = PRG(r)) = 1] = Pr[b = b^{'}] = \frac{1}{2} + \epsilon(\lambda) \tag{24}$$

where $\frac{1}{2}+\epsilon(\lambda)$ is the success probability of A (as assumed in our proof in Equation 22). By combining Equations 23 and 24 we have

$$|Pr[r \leftarrow \{0,1\}^{\lambda} : B(\overrightarrow{\delta} = PRG(r)) = 1] - Pr[B(\overrightarrow{\delta} \leftarrow Z_q) = 1]| = \epsilon(\lambda) \tag{25}$$

Equation 25 corresponds to the security definition of PRG (see Equation 4). 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*. ■

7.2 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. Next, we present a formal definition for invitation unforgeability together with a formal security proof of *Anonyma*.

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 the communication of other invitees and inviters and wishes to forge an invitation over its token.

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 2. *An invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function $negl(.)$ 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 ${\omega^*}^S$ 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 2. *Anonyma satisfies invitation unforgeability as defined in Definition 2, 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 $F_1, \cdots, F_{t-1}$ (where $F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [30] (where $f = S + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$).
   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 publicize the commitments of the polynomial and not the exact coefficients.
   B outputs $param = (G, q, g, ek, vk, (F_0, ..., F_{t-1}))$, 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., $InvLet = (T, e\varDelta)$ 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 expriment.
   - 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 $e\varDelta$ as $\omega_j^{\varDelta} = Dec_{dk}(e\varDelta)$. Next, B retrieves the record of (j, ω_j, r_j) corresponding to ω_j from Q^{Token} and checks whether $F_0^{r_j} \cdot \omega_j^{\varDelta} \overset{?}{=} T$ and responds to A accordingly. The right side of this equality check is the same as line 3 of $Ivrfy$ Algorithm (Algorithm 4.4) since

$$F_0^{r_j} \cdot \omega_j^{\varDelta} = g^{S \cdot r_j} \cdot \omega_j^{\varDelta} = g^{r_j \cdot S} \cdot \omega_j^{\varDelta} = \omega_j^S \cdot \omega_j^{\varDelta} \tag{26}$$

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 $\gamma_i = \prod_{j=0}^{t} F_j^{i^j}$ 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

$$\tau_{i,j} = \gamma_i^{r_j} \cdot g^{\delta_i \cdot r_j} = (g^{s_i})^{r_j} \cdot (g^{\delta_i})^{\cdot r_j} = (g^{r_j})^{s_i} \cdot (g^{r_j})^{\delta_i} = \omega_j^{s_i} \cdot \omega_j^{\delta_i} = \omega_j^{s_i + \delta_i} \tag{27}$$

   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^R_{POIC} 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

$$T \cdot Dec_{dk}(e\Delta)^{-1} = (\omega^*)^{S+\Delta} \cdot (\omega^*)^{\Delta^{-1}} = (\omega^*)^S = (g^x)^y = g^{xy} \quad (28)$$

$g^{x \cdot y}$ 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

$$\begin{aligned} Pr[\text{B wins}] &= Pr[B(G, q, g, X = g^x, Y = g^y) = g^{x \cdot y}] \quad (29)\\ &= Pr[\text{A wins} \wedge (Token^{'} = Token^*)]\\ &= Pr[\text{A wins } | Token^{'} = Token^*] \cdot Pr[Token^{'} = Token^*]\\ &\geq \epsilon \cdot \frac{1}{Poly(\lambda)} \end{aligned}$$

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 with the hardness of the CDH problem. Hence A's success probability in $InvUnforge_A(\lambda)$ must be negligible. This concludes the proof. ■

7.3 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 $S_{inviter}$ 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 *inviter* server and one *registration* server.

Theorem 3. *AnonymaX provides inviter anonymity in F^R_{POIC} hybrid model (as defined in Equation 36), 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 1. The only difference is in the *SetUp* phase where the challenger outputs two pair of encryption keys $(ek_{reg}, ek_{inviter})$ 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 *inviter* 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 $S_{inviter}$ acts against S_{reg}, i.e. $S_{inviter}$ wants to generate valid invitations for the invitee of its choice to join the *registration* service. This is trivial since $S_{inviter}$ is able to register arbitrary many users into its own system (i.e., *inviter* system). Then, every subset of t registered users of *inviter* 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 *inviter* 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 *inviter* service together with $S_{inviter}$ and some of the uncorrupted *registration* servers. At the end of the game, the mission of 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 *inviter* 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 *inviter* service is denoted by $U_{inviter}$. I_c represents the set of $t-1$ corrupted members in the *inviter service* whereas I_h contains the indices of the honest members. We have $U_{inviter} = I_h \cup I_c$. 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 $I_c \subset U_{inviter}$ 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 *inviter* 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 *inviter* 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 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 $XIvrfy(InvLet, Token^{'}, Param_{inviter}, Param_{S_{j^*}}, dk_{S_{j^*}})$ is accepted, then the game's output is 1 indicating the adversary's success, 0 otherwise.

Definition 3. *An cross-netwrok invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function negl(.) such that:*

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

Theorem 4. *AnonymaX satisfies invitation unforgeability as defined in Definition 3, 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 is able to 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_{S_j}).
   Similarly, B sets up an encryption and signature key pairs for $S_{inviter}$ as $(ek_{inviter}, dk_{inviter})$ and
   $(sk_{inviter}, vk_{inviter})$, 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 $F_1, \cdots, F_{t-1}$ (where $F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [30] (where $f = S + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$). 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 *inviter* server since it only needs to publicize the commitments of the polynomial and not the exact coefficients.
B outputs $param = (G, q, g, ek_{inviter}, vk_{inviter}, (F_0, ..., F_{t-1}))$, 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., $S_{inviter}.Tgen$ 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^R_{PODL}, 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^R_{PODL}, 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^R_{PODL} 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 $S_j.XIvrfy(InvLet, Token, Param_{inviter}, dk_{S_j})$ 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.
$$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = X^{S+\Delta} \cdot X^{-\Delta} = X^S = g^{xS} = g^{xy} \tag{30}$$
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 12) 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
$$\tau_i = \gamma_i^r \cdot g^{\delta_i \cdot r} = (g^{s_i})^r \cdot (g^{\delta_i})^{\cdot r} = (g^r)^{s_i} \cdot (g^r)^{\delta_i} = \omega^{s_i} \cdot \omega^{\delta_i} = \omega^{s_i+\delta_i} \tag{31}$$
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
$$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = (X)^{S+\Delta} \cdot (X)^{\Delta^{-1}} = (X)^S = (g^x)^y = g^{xy} \tag{32}$$

   $g^{x.y}$ 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 for 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

$$\begin{aligned} Pr[\text{B wins}] &= Pr[B(G,q,g,X=g^x,Y=g^y)=g^{x\cdot y}] \qquad (33)\\ &= Pr[\text{A wins} \wedge (Token^{'} = Token^* \text{ AND } j'=j^*)]\\ &= Pr[\text{A wins } |Token^{'} = Token^* \text{ AND } j'=j^*]\cdot Pr[Token^{'} = Token^* \text{ AND } j'=j^*]\\ &\geq \epsilon \cdot \frac{1}{Poly(\lambda)}\cdot\frac{1}{N} \end{aligned}$$

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].

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

[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 $Token = (\eta, l, \omega) \in Q_{j^*}^{Tgen}$. 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.

8 Related Works

In this section, we investigate related studies under two main categories: Electronic Voting (e-voting) systems and Ring-based signature schemes. These two topics show the most similarity to the invitation-only registration systems and address the confidentiality of the inviter-invitee relationship. However, the research done in both categories suit their unique settings and suffer from the efficiency issues when deployed for the invitation-only registration scenario. More details are provided below.

8.1 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 [19, 29], or secret sharing schemes where multiple authorities obtain one share of the ballot [25]. Before tallying the votes, the identifiable information shall be removed from the individual votes either by shuffling them through mix-net [9] or by homomorphically aggregating them [25]. 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 to the non-inviters (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 voting twice will have to risk the disclosure of their real identities (i.e., SSN) [1, 26]. 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).

8.2 (t,N) Threshold Ring Signature

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 [17]. 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 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 [6, 24, 21, 4]. In some other cases, the dependency is exponential [17]. 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 $O(1)$.

9 Conclusion

In *Anonyma*, we proposed an anonymous invitation-only system satisfying *inviter anonymity* and *invitation unforgeability* simultaneously. The inviter anonymity guarantees that the knowledge of who is invited by whom remains confidential against the system administrator as well as the inviters of the same invitee. By the invitation unforgeability, the system administrator is guaranteed that invitees without a sufficient number of inviters would not be able to successfully authenticate themselves to the system. Both security objectives are formally defined and proved in a *malicious adversarial model.* The anonymity of inviter relies on the security of the employed pseudo-random generator and the invitation unforgeability relies on the hardness of the computational Diffie-Hellman assumption. Unlike the prior studies whose running time depends on the total number of system members, in *Anonyma*, the running time complexity of invitation generation was $o(t)$ (t is the required number of inviters) and the verification of invitation required constant many operations at the server. Additionally, we devised an anonymous cross-network invitation-based system, *AnonymaX* which is a slightly modified variant of *Anonyma*. *AnonymaX* empowered users of one network to act as inviters for another network. *AnonymaX* achieves provable inviter anonymity the same way as *Anonyma* does. The proof of invitation unforgeability of *AnonymaX* is provided basing on the hardness of the computational Diffie-Hellman assumption.

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. Benaloh, J., and Tuinstra, D. Receipt-free secret-ballot elections. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (1994), ACM, pp. 544–553.
2. Blum, M., and Micali, S. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing 13*, 4 (1984), 850–864.
3. Boshrooyeh, S. T., and Küpçü, A. Inonymous: anonymous invitation-based system. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Springer, 2017, pp. 219–235.

4. Boyen, X. Mesh signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2007), Springer, pp. 210–227.
5. Brainard, J., Juels, A., Rivest, R. L., Szydlo, M., and Yung, M. Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM conference on Computer and communications security* (2006), ACM, pp. 168–178.
6. Bresson, E., Stern, J., and Szydlo, M. Threshold ring signatures and applications to ad-hoc groups. In *Annual International Cryptology Conference* (2002), Springer, pp. 465–480.
7. Chaabane, A., Acs, G., Kaafar, M. A., et al. You are what you like! information leakage through users' interests. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)* (2012).
8. Chaum, D., and Pedersen, T. P. Wallet databases with observers. In *Annual International Cryptology Conference* (1992), Springer, pp. 89–105.
9. Chaum, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM* (1981), vol. 24, ACM, pp. 84–90.
10. Diffie, W., and Hellman, M. New directions in cryptography. *IEEE transactions on Information Theory 22*, 6 (1976), 644–654.
11. ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory 31*, 4 (1985), 469–472.
12. Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), ACM, pp. 218–229.
13. Gong, N. Z., and Liu, B. Attribute inference attacks in online social networks. In *ACM Transactions on Privacy and Security (TOPS)* (2018), vol. 21, ACM, p. 3.
14. Gong, N. Z., and Wang, D. On the security of trustee-based social authentications. In *IEEE transactions on information forensics and security* (2014), vol. 9, IEEE, pp. 1251–1263.
15. Groth, J. Non-interactive zero-knowledge arguments for voting. In *International Conference on Applied Cryptography and Network Security* (2005), Springer, pp. 467–482.
16. Hazay, C., and Lindell, Y. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.
17. Isshiki, T., and Tanaka, K. An (n–t)-out-of-n threshold ring signature scheme. In *Australasian Conference on Information Security and Privacy* (2005), Springer, pp. 406–416.
18. Katz, J., and Lindell, Y. *Introduction to modern cryptography*. CRC press, 2014.
19. Kiayias, A., and Yung, M. The vector-ballot e-voting approach. In *International Conference on Financial Cryptography* (2004), Springer, pp. 72–89.
20. Kravitz, D. W. Digital signature algorithm, July 27 1993. US Patent 5,231,668.
21. Liu, J. K., Wei, V. K., and Wong, D. S. A separable threshold ring signature scheme. In *International Conference on Information Security and Cryptology* (2003), Springer, pp. 12–26.
22. Mahmood, S. Online social networks: Privacy threats and defenses. In *Security and Privacy Preserving in Social Networks*. Springer, 2013, pp. 47–71.
23. Malar, G. P., and Shyni, C. E. Facebookfs trustee based social authentication. In *Int. J. Emerg. Technol. Comput. Sci. Electron* (2015), vol. 12, pp. 224–230.
24. Melchor, C. A., Cayrel, P.-L., Gaborit, P., and Laguillaumie, F. A new efficient threshold ring signature scheme based on coding theory. In *IEEE Transactions on Information Theory* (2011), vol. 57, IEEE, pp. 4833–4842.

25. Nair, D. G., Binu, V., and Kumar, G. S. An improved e-voting scheme using secret sharing based secure multi-party computation. In *arXiv preprint arXiv:1502.07469* (2015).
26. Radwin, M. J., and Klein, P. An untraceable, universally verifiable voting scheme. In *Seminar in Cryptology* (1995), pp. 829–834.
27. Rosen, A. A note on constant-round zero-knowledge proofs for np. In *Theory of Cryptography Conference* (2004), Springer, pp. 191–202.
28. Roy, A., and Karforma, S. A survey on digital signatures and its applications. In *Journal of Computer and Information Technology* (2012), vol. 3, pp. 45–69.
29. Schneider, A., Meter, C., and Hagemeister, P. Survey on remote electronic voting. In *arXiv preprint arXiv:1702.02798* (2017).
30. Schoenmakers, B. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference* (1999), Springer, pp. 148–164.
31. Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.
32. Yu, J., Kong, F., Cheng, X., Hao, R., and Li, G. One forward-secure signature scheme using bilinear maps and its applications. In *Information Sciences* (2014), vol. 279, Elsevier, pp. 60–76.

A Proof of Invitation Correctness

A.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

$$g^{Z_1 - Z_1^*} = \gamma_i^{e-e^*} = g^{s_i \cdot (e-e^*)} (\text{mod } p) \tag{34}$$

Thus, $Z_1 - Z_1^* \equiv s_i \cdot (e - e^*)\ mod\ q$. 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

$$\begin{aligned} e\delta_{i,1}^{e-e^*} &= \omega^{Z_2 - Z_2^*} \cdot h^{Z_3 - Z_3^*} (\text{mod } p) \\ \omega^{\delta_i (e-e^*)} \cdot h^{r(e-e^*)} &= \omega^{Z_2 - Z_2^*} \cdot h^{Z_3 - Z_3^*} (\text{mod } p) \end{aligned} \tag{35}$$

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

A.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 $A = \frac{g^{Z_1}}{\gamma_i^e} \text{ mod } p$ and $B_1 = \frac{\omega^{Z_2} \cdot h^{Z_3}}{e\delta_{i,1}^e} \text{ mod } p$ and $B_2 = \frac{g^{Z_3}}{e\delta_{i,2}^e} \text{ mod } p$ and $C = \tau_i^{-e} \cdot B \cdot e\delta_{i,1}^e \cdot h^{-Z_3} \cdot \omega^{Z_1} \text{ mod } p$. 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.

A.3 Zero-knowledge POIC (ZKPOIC):

We apply the method given in [16] to our Σ protocol to convert it to a zero-knowledge proof system. Let F^{R}_{POIC} (given in Equation 36) demonstrate the security guarantees of the resultant ZKPOIC over the relation R that we defined in Equation 14.

$$F^{R}_{POIC}((X, W), X) = (\perp, R(X, W)) \tag{36}$$

F^{R}_{POIC} 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^{R}_{POIC} receives a common input X from the prover and the verifier as well as the private input W from the prover. F^{R}_{POIC} outputs to the verifier whether X and W fit into the relation R.