Deniable Anonymous Group Authentication

Ewa Syta Benjamin Peterson David Isaac Wolinsky
Michael Fischer Bryan Ford

YALEU/DCS/TR-1486
February 13, 2014

Deniable Anonymous Group Authentication[*]

Ewa Syta[†]Benjamin Peterson David Isaac Wolinsky Michael Fischer
Bryan Ford

Abstract

In some situations, users need to authenticate as *distinct* members of some well-defined group, without revealing their individual identities: to validate and corroborate a leak, for example, or to count participants in a closed anonymous forum. Current group authentication techniques offering this capability, however, may de-anonymize users if an attacker *later* compromises their private keys. Addressing this under-explored risk, we present *deniable anonymous group authentication* (DAGA), the first anonymous authentication protocol offering *proportionality*, *forward anonymity*, and *deniability* in combination. To offer these properties, DAGA leverages a federation of collectively (but not individually) trusted servers. These servers collectively generate *tags* during authentication, which ensure client distinctness and proportionality, while cryptographically scrubbing information that could later de-anonymize clients. After an authentication round, clients and (honest) servers securely erase their ephemeral secrets, protecting clients from later de-anonymization even if an attacker eventually compromises *all* long-term client and server keys. A proof-of-concept prototype validates DAGA's practicality, authenticating a client into a 32-member group in one second, or into a 2048-member group in two minutes.

1 Introduction

In privacy-sensitive communications, one user sometimes needs to prove to be a member of some explicit, well-defined group, without revealing his individual identity. Consider for example a whistleblower who wishes to leak evidence of corporate or government wrongdoing to a journalist, via an anonymous electronic "drop box" [23]. The journalist needs to validate the source's trustworthiness, but the whistleblower is reluctant to reveal his identity for fear their communications might be compromised [34], or that the journalist will be coerced into testifying against the source [52]. The whistleblower thus wishes to *authenticate anonymously* as a member of some authoritative circle who plausibly has knowledge of and access to the leaked information, such as a corporate board member or executive, or a government official of a given rank.

Even if the whistleblower convinces the journalist of his authority, the journalist may also require *corroboration*: e.g., confirmation by one or more *other* members of this authoritative circle that the leaked information is genuine. Other members of this authoritative circle may

[*]This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018.

[†]Department of Computer Science, Yale University, CT

be just as reluctant to communicate with the journalist, however. If a potential corroborator also demands anonymity, how can the journalist (or the public) know that the corroborator is indeed a *second* independent source, and not just the original source wearing a second guise? In general, if the journalist knows k pseudonymous group members, how can he know that these pseudonyms *proportionally* represent k real, distinct group members, and are not just k Sybil identities [26]?

Finally, the whistleblower is concerned that once the leak becomes public, he may be placed under suspicion – perhaps merely for being in the relevant authoritative circle – and any of his computing devices may be confiscated or compromised along with his private keys. Even if his keys are compromised, the whistleblower needs his anonymity *forward* protected, against both the journalist and any third-parties who might have observed their communications. Further, the whistleblower wishes to be able to *deny* having even participated in any sensitive communication, including the fact of having authenticated at all (even anonymously) to the journalist.

We present *deniable anonymous group authentication* (DAGA), the first protocol we are aware of satisfying the above requirements, which we term *anonymity*, *proportionality*, *forward anonymity*, and *deniability.* Like ring signatures [50], DAGA allows a user to authenticate as an anonymous member of an ad hoc group or ring, defined by an arbitrary list of public keys. The user can *conscript* other users into a group without their participation, consent, or even knowledge. Neither ring signatures nor deniable ring authentication [47] offer proportionality, however: a verifier cannot tell whether several authentications were by the same or distinct group members. Linkable ring signatures [44] include a *tag* enabling a verifier to check distinctness, but anyone who later compromises the user's private key can reproduce the linkage tags in all past signatures, violating forward anonymity and deniability. It appears likely that no purely offline anonymous signature scheme can offer both proportionality (corroboration capability), forward anonymity, and deniability in combination.

To resolve these apparently conflicting requirements, DAGA relies on a federation of independently operated servers that are *collectively* but not individually trusted. DAGA's security property properties are ensured as long as *at least one* server operates correctly and honestly during an authentication process, even if the client does not know which server is honest. The servers divide authentication activity into *epochs*, choosing a set of fresh server-side secrets for each epoch. These secrets collectively protect the relationship between a client's private key and the epoch-specific tags that DAGA produces to offer proportionality and corroboration capability. After each epoch, the honest server(s) securely erase their secrets, preventing anyone from compromising any client's anonymity in past authentication epochs – even if the attacker later compromises the long-term private keys of *all* clients and *all* servers. Finally, the authentication process offers deniability by employing only interactive zero-knowledge proofs, ensuring that any valid DAGA communication transcript could have been synthesized independently by anyone.

We have analyzed and verified DAGA's four key security properties of anonymity, proportionality, forward anonymity, and deniability. We have also built a working proof-of-concept implementation of DAGA to validate its performance and practical usability. Using 2048-bit DSA keys, our DAGA prototype can authenticate as a member of a 32-member group to 2 servers in about one second after consuming less than 1KB of total messaging bandwidth. Authenticating in a 2048-member group takes about two minutes and consumes about 100KB of bandwidth. Our initial prototype is currently unoptimized, and we expect its performance and efficiency can be improved in many ways. Nevertheless, our results suggest that DAGA is already practical for sensitive anonymous interactions requiring maximum security, and we believe DAGA's unique combination of proportionality (corroboration), forward anonymity, and deniability features can justify this cost in such scenarios.

This paper makes the following key contributions:

1) proposes a new authentication scheme that offers anonymity, deniability, and proportionality even in the case of a full compromise of private keys (Section 4),

2) proposes an authentication scheme that supports evolving groups *while* preserving proportionality,

3) separates the notions of deniability, anonymity and forward anonymity, and analyzes these security properties.

Section 2 offers an overview of DAGA's trust model, operation, and security properties. Section 3 outlines several applications for which DAGA might be suited. Section 4 presents the details of the DAGA protocol. Section 5 outlines potentially useful extensions to the basic protocol, and Section 6 outlines practical implementation and deployment considerations. Section 7 presents our prototype implementation and experimental results, and Section 8 summarizes DAGA's formal security properties and briefly sketches our proofs of these properties. Finally, Section 9 outlines related work, and Section 10 concludes.

2 Overview

2.1 Trust Model

We assume an *anytrust* [60, 61] model, where there is a large set of n clients and a smaller set of m reliable servers, which includes at least one honest server that runs the prescribed protocols and does not collude with dishonest entities. The clients do not need to assume that any particular server is trustworthy; they need only trust that *some* honest server exists. We further assume that there are always at least two honest clients; anonymity is trivially impossible if $n-1$ clients choose to collude against only one honest client.

We assume that each anytrust server is run by a respected, reliable, and independently managed organization, each responsible for ensuring that its server remains online and uncompromised. We envision these anytrust servers being deployed by a federation of organizations wishing to support responsible forms of anonymous participation: e.g., providers of online services such as Wikipedia or Twitter, anonymity system providers such as the Tor project, non-profit organizations whose aim is to further online privacy and anonymity, or even for-profit organization desiring strong guarantees and large anonymity sets for their clients.

In such a deployment scenario, we expect the servers to offer high reliability and to offer clients with a high level of confidence that at least one honest server exists. In practice, we hope and expect a majority of the servers to be honest, allowing for an efficient resolution of issues related to the servers' performance or availability, should they arise, but we leave such availability issues outside the scope of this paper.

2.2 Protocol Overview

The main idea underlying DAGA is to allow a client to authenticate anonymously, and at the same time enforce proportionality, by enabling the servers to link authentications of the same client. To achieve this goal, we use a combination of proofs of knowledge to prove membership to a particular group and per-client *linkage tags* that effectively become clients' anonymous IDs.

Each client i authenticates using a publicly available *authentication context* C, which consists of a group definition G and other per-round authentication information. A client i prepares and sends his authentication message to an arbitrarily chosen server who starts the collective process of producing the client's final linkage tag by all servers, and upon its completions responds to the client with an authentication decision as shown in Figure 1.

To produce an authentication message, a client i generates an *initial linkage tag* $T_0^i = h_i^{\prod_{k=1}^{m} s_k}$, where h_i is the client's per-round generator assigned by the servers and s_k is a shared secret for every server k that a client generates in a way that each server is able to independently reconstruct it. In addition to creating the tag, the client proves in zero-knowledge that he correctly computed T_0^i *and* that he is a member of the group G and therefore he knows a private key x_i that corresponds to one of the public keys included in the group definition. A client i executes the following interactive "OR" proof [14,21]:

$$\text{PK} = \{\vee_{i=1}^{n}(\text{I know private key } x_i \wedge T_0 \text{ is correctly based on } h_i)\}$$

After completing these steps, client i securely erases his private ephemeral state and sends to some server j his tag, proof, and all other information needed by the servers to process his authentication request. The server who receives i's authentication request, verifies the attached proof, and processes the initial tag by scrubbing from T_0 the secret s_j it shares with i and adding his own per-round secret r_j. Finally, server j proves in zero-knowledge that he correctly performed these steps and generates the following proof using a standard proof of knowledge about discrete logarithms [19,29,53]:

$$\text{PK} = \{(\text{Tag } T_j \text{ is correct} \wedge \text{I know my secret } r_j)\}$$

The remaining servers repeat this process, however, also verifying that the proof coming from the previous server is valid. Provided that the client i and the servers correctly follow the protocol, it yields a final linkage tag $T_f^i = h_i^{\prod_{k=1}^{m} r_k}$. Each final linkage tag T_f is unique to a client and remains the same for each authentication within the same context C as the tag depends on a client's generator and a product of all servers' secrets which remain the same.

2.3 Security Properties

DAGA provides for a deniable and anonymous authentication scheme that maintains its properties even if a client's private key is compromised. A client should be able to convince the servers that he is a *unique* member of a *particular* group, without disclosing his non-anonymous identity and without leaving any evidence that can be used later on to link him to his well known identity. Anonymity and deniability should persist even in the case of a compromise of a client's private key after the round completion. More specifically, DAGA

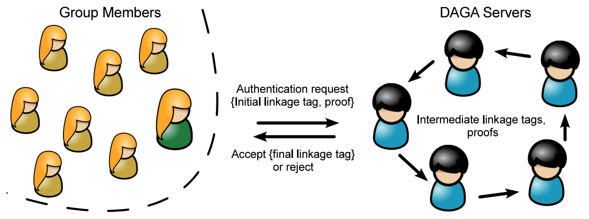

Group Members
DAGA Servers
Authentication request {Initial linkage tag, proof}
Accept {final linkage tag} or reject
Intermediate linkage tags, proofs

Figure 1: Conceptual model of DAGA

maintains anonymity and deniability as long as the private keys of at least two honest members and a private key of at least one honest server are not compromised. To maintain forward security, the basic DAGA protocol assumes that at least one honest server's private key remains secure. Section 5.4 proposes an extension that relaxes this requirement, however, preserving forward secrecy even if *all* servers are eventually compromised.

In addition to *completeness* and *soundness*, DAGA offers four security properties: *anonymity*, *deniability*, *forward anonymity*, and *proportionality*.

Soundness: Under the Discrete Logarithm assumption, servers only accept authentication requests coming from a client who is a member of a group G specified in his authentication context.

Anonymity: Informally speaking, we want to ensure that after a complete protocol run, an adversary cannot guess which group member has been authenticated with a probability greater than random guessing. DAGA provides anonymity under the DDH assumption in the random oracle model.

Forward Anonymity: We extend the anonymity property to situations in which an adversary obtains a client's private key but only after a protocol run has completed, and ensure that the knowledge of even all but the honest server's key does allow an adversary to break any client's anonymity.

Deniability: We want to ensure that the protocol does not leave a "paper trail" that an adversary could use to link a client to his authentication requests based on intercepted authentication transcripts. This guarantee persists even in the case of a compromise of private keys yielding an interesting notion of *forward deniability*.

Proportionality: We enforce that a client can authenticate as a unique member only once given a particular authentication context and each subsequent authentication request within the same context is recognized as coming from that client. At the same time, we ensure that client's authentications made within two different authentication contexts are unlinkable. Additionally, proportionality persists even when new clients are added to a group because proportionality is independent of the group membership.

3 Applications

DAGA may be useful in many applications desiring anonymous authentication, such as online surveys, electronic coupons, trial subscriptions, etc. DAGA is most suitable for authentication into well-defined, closed groups of manageable size, and when guarantees of deniability and forward security are needed.

3.1 Distributing Keys for Group Anonymity Systems

Most anonymity systems fall into two categories: mix networks [15] (mix-nets) mask the identity of the sender by forwarding messages through multiple relays, and Dining Cryptographers networks [17] (DC-nets) leverage secrets exchanged within a well-defined group of members to anonymize messages. While mix-nets based systems (e.g., Tor [24]) are efficient, they do not provide unconditional anonymity and traffic analysis resistance as DC-nets based systems (e.g., Dissent [?, 61], Herbivore [54]) do.

To provide *accountability* – the ability to identify and expel members that attempt to disrupt group communication – Dissent requires each member to have a long-term signing key. This key, if well-known, links the intermediate output of the protocol to the key's owner, and links the protocol's entire output to a particular group of keys. If a client's identity is defined by a long-term non-anonymous key pair, a compromise of the client's private key could retroactively compromise the user's anonymity in all past exchanges.

Therefore, an anonymity system such as Dissent can leverage DAGA to set up ephemeral pseudonyms (signing keys) for participating group members, breaking the link between the client's long-term key (and identity) and the anonymous exchanges, while ensuring fairness via DAGA's proportionality property. Additionally, we can achieve a larger anonymity set and plausible deniability for the anonymous communication if we draw ephemeral pseudonyms from a much larger group of members than those who actively participate: e.g., many "members" may be conscripted into the group without their participation or knowledge.

3.2 Anonymous Voting with Deniability

DAGA may lend itself to certain forms of anonymous voting. Anonymity (to preserve voter's privacy) and proportionality (to enforce one-voter one-vote rule) is generally required in any anonymous voting scheme. Many anonymous e-voting schemes [2, 38, 39, 42] provide additional properties such as coercion-resistant and receipt-freeness, which offer a weaker notion of deniability. DAGA's deniability property ensures that once an election has ended, the resulting communication transcript leaves no verifiable proof that the vote even occurred. DAGA may thus be attractive for voting in a "dissident forum" under repression from an authoritarian regime, for example.

3.3 Secure Access to Sensitive Resources

We can envision using DAGA to distribute access tokens to resources, in particular to sensitive resources, in a way that gives access to a certain group of clients while providing deniability of ever requesting those sensitive resources.

Additionally, because DAGA provides proportionality, the servers can keep track of requests made by a particular anonymous user based on his final linkage tag. This gives the servers the ability to limit access to resources as desired (to one or k times) without exposing a client who inadvertently makes $k+1$ requests as done in many k-show credential systems [41]

3.4 Server-provided Signatures

After a client has been successfully authenticated as a unique group member, he might request that the servers collectively perform a specific action on his behalf, for example to sign a message anonymously and deniably.

This might be accomplished in several ways. Servers might sequentially sign a message as they process the client's tag provided that the client's proof is valid. At the end of a successful authentication, the servers might endorse a collective ephemeral signing key and produce a signature on the provided message. Alternatively, a client-defined subset of servers might issue a threshold DSS signature [30, 31].

3.5 Supporting Anonymous Federated Login

Crypto-Book [46] provides for a privacy-preserving and accountable digital identities. It leverages the existing digital identity providers, such as Facebook or Twitter, and the use of public-key encryption and linkable ring signatures. Linkable ring signatures [44, 45] allow a group member to anonymously sign a message in a way that hides his identity but allows others to verify that the signature was produced by a group member and to link all future signatures as coming from the same, anonymous member.

DAGA can be used in place of any linkable ring signature scheme as it provides the same functionality (anonymity and linkability) while adding deniability and forward security.

4 Protocol Description

4.1 Notation

We denote the client i's proof of correctness as PK^{client_i}, the server j's proof of correctness as $\mathrm{PK}_1^{server_j}$, and the server j's proof of a client i's misbehavior as $\mathrm{PK}_2^{server_{j(i)}}$. We denote the client i's initial linkage tag as T_0^i, the intermediate linkage tag created by a server j as T_j, and the client i's final tag as T_f^i. We will omit the client's ID from T_0^i (T_f^i) and write T_0 (T_f) when it is clear from the context which client the tag belongs to. We denote a client i's authentication message as M_0^i and a server's j message as M_j.

To simplify notation, we will omit "mod p" when performing computation on elements of $\mathbb{Z}_p$ and "mod q" when performing computation on exponents. We will denote choosing a random element x from $\mathbb{Z}_q^*$ as $x \in_R \mathbb{Z}_q^*$

4.2 Building Blocks

Σ-Protocols

Zero-knowledge proofs of knowledge [32] are proofs that yield nothing beyond of the validity of the assertion a prover P wants to convince a verifier V about. However, such proofs normally require a large number of interactions between the prover and verifier. A Σ-protocol [22] is a special type of an interactive zero-knowledge proof of knowledge that requires only one interaction and always consists of exactly three moves: given common input I (1) P sends a commitment t to V, (2) V responds with a random ℓ-bit challenge c, and (3) P sends back a response r. V makes a decision based on (I, t, c, r). By definition [22], a Σ-protocol has the properties of completeness, special soundness and special honest-verifier zero-knowledge. The client's and servers' proofs instantiated in DAGA have these properties.

"OR" Proofs

An "OR" proof of knowledge is an example of a Σ-protocol and allows a prover to convince a verifier that he knows a secret x that corresponds to one out of two assertions without the verifier learning which one. An "OR" proof can be easily generalized to proving the knowledge of a witness to one of many assertions ("1-out-of-n") or even multiple witnesses ("k-out-of-n").

DAGA makes use of interactive and non-interactive proofs. The client's proof is an interactive protocol instantiated using the techniques of Camenisch and Stadler [14] which are an extension of the previous works on proof of knowledge [21,29,53]. The server's proofs uses non-interactive protocols based on Schnorr's proof of knowledge of discrete logarithms [53], proof of equality of discrete logarithms [19], While Σ-protocols are interactive by nature, a heuristic proposed in [29] allows to replace the interaction with a verifier with a hash function modeled as a random oracle. For simplicity, we write "proof" or "proof of knowledge" for "three-move honest-verifier computationally zero-knowledge proof of knowledge".

4.3 Assumptions

We assume that communication channels exist between all parties and a client has an authenticated channel with every server. We assume an adversary that is polynomial-time limited, can control a colluding subset of up to $n-2$ clients and up to $m-1$ servers, and can observe and record all network messages.

We assume that each client has a long-lived non-anonymous identity associated with a public-private key pair. We define a client's identity as his associated key pair; therefore, a client i represents a client who owns a public key X_i. Specifically, each client i has a long-term Diffie-Hellman (DH) key pair consisting of a private key x_i and public key $X_i = g^{x_i}$ and each server j has a corresponding private/public key pair $(y_j, Y_j = g^{y_j})$. We assume that there is a readily available group definition $G = (\vec{X}, \vec{Y})$ listing clients and servers and their long-term public keys, X_i and Y_j respectively. The author of a group definition may conscript arbitrary clients knowing only their public keys. Some of the clients listed in the group definition need not ever participate in the protocol or even be aware that they are included. The group definition is a part of an authentication context which defines all constants for each authentication round.

4.4 Authentication Context

In DAGA, a client i anonymously authenticates as a member of a particular group G with the help of a set of *anytrust* servers using a publicly available *authentication context* C. An authentication context $\mathrm{C} = (G, \vec{R}, \vec{H}, p, g)$ consists of a group definition G, a set $\vec{R}$ of each server's commitment to a per-round secret, a set $\vec{H}$ of each client's per-round generators, a safe prime $p = 2q + 1$ where q is a sufficiently large prime, and a generator g of the order q subgroup $\mathcal{G}$ of $\mathbb{Z}_p^*$. We define a *group* G as a tuple $(\vec{X}, \vec{Y})$ where $\vec{X}$ is a set of the n clients' public keys and $\vec{Y}$ is a set of the m servers' public keys. To generate $\vec{R} = (R_1, \dots, R_m)$, each server chooses a secret $r_j \in_R \mathbb{Z}_q^*$ and publishes a commitment $R_j = g^{r_j}$. $\vec{H} = (h_1, \dots, h_n)$ consists of n unique per-round generators of $\mathcal{G}$, one for each client i, such that no one knows the logarithmic relationship between any h_i and g or between h_i and $h_{i'}$ for any pair of clients $i \neq i'$. Section 6.5 describes how to find these generators and Section 6.3 further discusses issues related to creating and using an authentication context.

4.5 Client's Protocol

A client i wishing to authenticate, obtains an authentication context C, uses it to produce an authentication message M_i^0, and sends it to one arbitrarily chosen server listed in $\vec{Y}$. Upon receiving the client's message, all servers collectively process M_i^0 and either accept or reject i's authentication request. If i's request is accepted, then it results in a final linkage tag T_f^i. It it is rejected, however, then the client's proof PK^{client_i} is invalid, at least one server produces a proof $\mathrm{PK}_2^{server_{j(i)}}$ of the client's misbehavior, or some server produces an invalid proof $\mathrm{PK}_1^{server_j}$.

We define an *authentication round* with respect to a particular authentication context C. Each authentication request, regardless of the identity of the originating client, belongs to the same round if it is made with respect to C. All requests within the same round are linkable, that is, each time a client i authenticates, the servers will be able to link these requests as coming from *some* client from G.

A client i performs the following steps to create M_i^0.

Step 1: Client i first picks an ephemeral private DH key $z_i \in_R \mathbb{Z}_q^*$ and computes a public key $Z_i = g^{z_i}$. Client i keeps z_i secret.

Step 2: For each server j, i computes a shared secret exponent $s_j = \mathcal{H}_1(Y_j^{z_i}) = \mathcal{H}_1(g^{y_j z_i})$, where $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{Z}_q^*$ is a hash function and Y_j is the sever j's public key as listed in $\vec{Y}$.

Step 3: Client i computes his initial linkage tag $T_0^i = h_i^{\prod_{k=1}^m s_k}$ using his per-round generator h_i as listed in $\vec{H}$ and the secret exponents shared with all servers. Then, for each server $1 \leqslant j \leqslant m$, i computes $\vec{S} = (S_0, \dots, S_m)$, a set of commitments to a secret s_j he shares with each server j: $S_j = g^{\prod_{k=1}^j s_k}$ such that $S_0 = g$, $S_1 = g^{s_1}$, $\dots$, $S_m = g^{\prod_{k=1}^m s_k}$. Finally, client i sets $S = (Z_i, \vec{S})$.

Step 4: Now, client i proves that *(i)* he correctly followed the protocol, that is his initial linkage tag T_0^i is correctly constructed using h_i and $s = \prod_{k=1}^m s_k$, and *(ii)* he belongs to the group G because he knows *some* private key x that corresponds to *some* public key $X \in \vec{X}$. To do so, i runs an interactive proof of knowledge as described in Section 4.7. The client's proof PK^{client_i} looks as follows.

$$PK\{(x_i, s) : \{\vee_{k=1}^n (X_k = g^{x_k} \wedge S_m = g^s \wedge T_0 = h_k^s)\}$$

Step 5: Client i securely erases each secret s_j and z_i. Finally, client i creates and sends to an arbitrarily chosen server j his message $\mathrm{M}_i^0 = (C, S, T_0, P_0)$, where $\mathrm{C} = (G, \vec{R}, \vec{H}, p, g)$ is the authentication context i used, $\mathrm{S} = (Z_i, S_0, \dots, S_m)$ is the client's ephemeral public key and the set of client's commitments, T_0^i is the initial linkage tag, P_0 is the client's proof.

4.6 Servers' Protocol

All servers collectively process a client i's authentication message M_i^0 and at the end of this process either reject i's authentication request or accepts it as output i's final linkage tag T_f^i. A client i arbitrarily chooses some server j to whom he sends M_i^0. We denote that server j as server 1, since j is the first server to process i's request, and denote server 2 as $j+1$ and finally the last server m as $j-1$. This defines a unique order based on the list of server in $\vec{Y}$ in which each server processes i's message.

The first server to process the client's authentication requests receives the message $\mathrm{M}_i^0 = (C, S, T_0, P_0)$. Then, each server j creates a message $\mathrm{M}_j = (M_{j-1}, T_j, P_j)$ to pass to the next server in sequence, where M_{j-1} is the authentication message M_i^0 (if $j = 1$) and the message received from the previous server (if $j > 1$), T_j is the linkage tag produced by j and P_j is the proof produced by j. Each M_j for $j > 1$ consists of all previous messages such that each server j can verify all messages produced thus far (including the client's original message M_i^0).

Each server j performs the followings steps to create M_j.

Step 1: Server j checks the incoming message M_{j-1} and rejects it the message is valid. Then, j checks the proof of correctness of the previous servers' computations (unless $j = 1$) as well as the client's proof P_0. Server j proceeds only if all proofs are valid, and aborts otherwise.

Step 2: First, server j reconstructs the secret s_j he shares with the client as $s_j = \mathcal{H}_1(Z^{y_j}) = \mathcal{H}_1(g^{y_j z})$. Then, j verifies the client's commitments S_{j-1} and S_j against s_j. That is, the server checks that $S_j = S_{j-1}^{s_j}$. If yes, then j proceeds to Step 3 and if not, then j reveals s_j together with a proof that he computed it correctly based on the client's commitment Z and his public key Y_j as described in Section 4.9. In such a case, server j produces the following proof PK_2^{server}.

$$PK\{(y_j) : (Z_{s_j} = Z^{y_j} \wedge Y_j = g^{y_j})\}.$$

Server j creates and sends to the next server his message $\mathrm{M}_j = M_{j-1}, T_j = 0, P_j = \mathrm{PK}_2^{server}$.

Step 3: Server j computes his intermediate linkage tag $T_j = (T_{j-1})^{(r_j)(s_j^{-1})}$ using his per-round secret r_j and a multiplicative inverse of the shared secret s_j which results in a tag $T_j = h_i^{\prod_{k=1}^{j} r_k \prod_{k=j+1}^{m} s_k}$. Now, j produces a non-interactive proof P_j of correctness as described in Section 4.8. The server proves that he correctly computed the new tag T_j with respect to the server's per-round commitment R_j and the shared secret s_j. Server j produces PK_1^{server} as follows.

$$PK\{(r_j, s_j) : T_{j-1}^{r_j} = T_j^{s_j} \wedge R_j = g^{r_j} \wedge S_j = S_{j-1}^{s_j}\}$$

Step 4: Finally, server j securely erases s_j, forms his outgoing message $\mathrm{M}_j = (\mathrm{M}_{j-1}T_j, P_j = \mathrm{PK}_1^{server})$, and sends M_j to server $j+1$ if $j < m$, or to all servers if $j = m$.

Step 5: Server j securely erases his per-round secrets r_j upon a completion of a round, that is when the authentication context C expires.

After a successful completion of the protocol, all servers learn a final linkage tag $T_f = h_i^{\prod_{k=1}^{m} r_k}$. The tag only depends on the client's per round generator h_i and a product of all servers' per-round secrets r_j, regardless of the initial linkage tag T_0. Thus, a client can obtain only one linkage rage per round.

4.7 Client's Proof PK^{client_i}

Each clients $\hat{\imath}$'s authentication message $\mathrm{M}_{\hat{\imath}}^0$ includes the following proof of knowledge P_0:

$$PK\{(x_{\hat{\imath}}, s) : \{\vee_{k=1}^{n}(X_k = g^{x_k} \wedge S_m = g^s \wedge T_0 = h_k^s)\}$$

In this proof, the client $\hat{\imath}$ proves that he either knows a private key x_1 and his tag T_0 is correct, or that he knows x_2 and T_0 is correct, including an "OR" statement for each private key included in $\vec{X}$. Because $\hat{\imath}$ knows only one private key, namely $x_{\hat{\imath}}$, he simulates the "OR" statements for all other private keys in a way that will convince the servers that the authenticating client knows one private key and the tag is properly formed. More specifically, client $\hat{\imath}$ proves that that *(i)* client $\hat{\imath}$'s linkage tag T_0 is created with respect to his per-round generator $h_{\hat{\imath}}$, *(ii)* S_m is a proper commitment to $s = \prod_{k=1}^{m} s_k$, the product of all secrets that $\hat{\imath}$ shares with the servers, and *(iii)* client $\hat{\imath}$'s private key $x_{\hat{\imath}}$ corresponds to one of the public keys included in the group definition G.

Prover's Steps The prover, a client $\hat{\imath}$ holding private key $x_{\hat{\imath}}$ and s performs the following step to calculate $P_0 = P$:

1. Choose $w_1, \ldots, w_n$ such that $w_{\hat{\imath}} = 0$ and $w_i \in_R \mathbb{Z}_q^*$ for $i \neq \hat{\imath}$, and choose $v_{1.0}, v_{1.1}, \ldots, v_{n.0}, v_{n.1} \in_R \mathbb{Z}_q^*$. For each client $i \in G$, compute commitments $t_{i.0} = X_i^{w_i} g^{v_{i.0}}$, $t_{i.10} = S_m^{w_i} g^{v_{i.1}}$, and $t_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$.
   Set $t = (t_{1.0}, t_{1.10}, t_{1.11}, \ldots, t_{n.0}, t_{n.10}, t_{n.11})$ and send it to an arbitrarily chosen server.
2. Upon receiving the client's commitments, the severs collectively generate a random challenge c_s (as described in Section 6.4) and send c_s back to the client.
3. Compute $c = (c_1, \ldots, c_n)$ as:

$$c_i = \begin{cases} c_s - \sum_{k=1}^{n} w_k & \text{for } i = \hat{\imath} \\ w_i & otherwise \end{cases}$$

   Compute responses $r = (r_{1.0}, r_{1.1}, \ldots, r_{i.0}, r_{i.1})$ as follows. Let $x_{i.0} = x_{i.1} = 0$ for all $i \neq \hat{\imath}$, let $x_{\hat{\imath}.0} = x_{\hat{\imath}}$, and let $x_{\hat{\imath}.1} = s$. Compute $r_{i.k} = v_{i.k} - c_i x_{i.k}$ for all $1 \leqslant i \leqslant n$ and $k \in \{0, 1\}$. Set $P = (c_s, t, c, r)$.

Verifier's Steps The verifier, one of the servers, performs the following steps to verify the proof.

1. Check the commitments $t_{i.0} \stackrel{?}{=} X_i^{c_i} g^{r_{i.0}}$, $t_{i.10} \stackrel{?}{=} S_m^{c_i} g^{r_{i.1}}$, and $t_{i.11} \stackrel{?}{=} T_0^{c_i} h_i^{r_{i.1}}$, for all $1 \leqslant i \leqslant n$.
2. Check the challenge $c_s \stackrel{?}{=} \sum_{i=1}^{n} c_i$.

4.8 Server's Proof: Proving Correctness of its Work

After processing an incoming tag T_{j-1}, each server j must prove the correctness of its computations. That is, a server produces a proof of knowledge $\mathrm{PK}_1^{server_j}$ that he created the tag T_j according to the protocol specification. That is, j proves that he *(i)* correctly recovered the shared secret s_j, *(ii)* used the correct per-round secret r_j with respect to $R_j \in \vec{R}$, and *(iii)* correctly removed s_j and added r_j to the tag.

$$PK\{(r_j, s_j) : T_{j-1}^{r_j} = T_j^{s_j} \wedge R_j = g^{r_j} \wedge S_j = S_{j-1}^{s_j}\}$$

Server j can generate such a proof if it knows r_j and s_j. Each honest server knows its own per-round secret r_j, and the secret s_j that relates S_j to S_{j-1}, otherwise if j were unable to reconstruct a correct s_j, then he would have exposed the client by producing a proof $\mathrm{PK}_2^{server_j}$ and would have never produced his tag T_j.

Prover's Steps The prover, server j holding s_j and r_j, performs the following steps to create $P_j = P$.

1. Choose $v_1, v_2 \in_R \mathbb{Z}_q^*$. Calculate $t_1 = T_{j-1}^{v_1} T_j^{-v_2}$, $t_2 = g^{v_1}, t_3 = S_{j-1}^{v_2}$.
2. Calculate $c = \mathcal{H}_2(T_{j-1}, T_j, R_j, g, S_j, S_{j-1}, t_1, t_2, t_3)$, where $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function.
3. Calculate $r_1 = v_1 - cr_j$ and $r_2 = v_2 - cs_j$.
4. Set $P = (t_1, t_2, t_3, c, r_1, r_2)$.

Verifier's Step The verifier, another server, upon receiving P_j can verify the proof as follows.

1. Reconstruct commitments $t_1' = T_{j-1}^{r_1} T_j^{-r_2}, t_2' = g^{r_1} R_j^c, t_3' = S_{j-1}^{r_2} S_j^c$.
2. Check $c \stackrel{?}{=} \mathcal{H}_2(T_{j-1}, T_j, R_j, g, S_j, S_{j-1}, t_1', t_2', t_3')$.

4.9 Server's Proof: Exposing a misbehaving client

To create a tag T_j, server j needs to reconstruct and then remove the secret s_j it shares with the client from the incoming tag T_{j-1}. Server j calculates $s_j = \mathcal{H}_1(Z^{y_j})$ using the client's commitment Z and its own private key y_j, and verifies that the recovered secret is correct by checking $S_j = S_{j-1}^{s_j}$. If the recovered secret is not correct, then j exposed the client as dishonest by providing a proof $\mathrm{PK}_2^{server_j}$ to other servers. To do so, the server reveals the secret s_j it computed and $Z_{s_j} = Z^{y_j}$, the preimage of s_j under $\mathcal{H}_1$. Then, server j prepares a proof that he *(i)* used his private key y_j that corresponds to a public key $Y_j \in \vec{Y}$, and *(ii)* correctly computed Z_{s_j} by raising the client's commitment Z to his private key y_j. Server j prepares the following proof of knowledge:

$$PK\{(y_j) : (Z_{s_j} = Z^{y_j} \wedge Y_j = g^{y_j})\}.$$

After receiving and verifying $\mathrm{PK}_2^{server_j}$, each server can can reconstruct $s_j = \mathcal{H}_1(Z_{s_j})$, check that indeed $S_j \neq S_{j-1}^{s_j}$, and

Prover's Steps The prover, server j holding a private key y_j, performs the following steps and obtains $P_j = (Z_{s_j}, P)$:

1. Choose $v \in_R \mathbb{Z}_q^*$. Calculate $t_1 = Z^v$ and $t_2 = g^v$.
2. Calculate $c = \mathcal{H}_2(Z_{s_j}, Z, Y_j, g, t_1, t_2)$.
3. Calculate $r = v - cy_j$
4. Set $P = (t_1, t_2, c, r)$.

Verifier's Steps The verifier, either a server or the client, upon receiving P_j can verify the proof as follows:

1. Reconstruct commitments $t_1' = Z^r Z_{s_j}^c$ and $t_2' = g^r Y_j^c$.
2. Check $c \stackrel{?}{=} \mathcal{H}_2(Z_{s_j}, Z, Y_j, g, t_1', t_2')$.

5 Extensions of DAGA

In this section we describe several possible extensions of our main protocol. First, we discuss ideas for improving DAGA's performance, then we discuss trading deniability for verifiability, show how to give the servers the ability to collectively revoke a client's anonymity, show how to make DAGA secure on full clients' and servers' key exposure, and lastly we present a variant of DAGA in which the client has a chance to inspect his linkage tag before it is revealed to the servers.

5.1 Improving Efficiency

Currently, the computation and communication overhead of DAGA grows linearly in the number of members in a group G. Ideally, we would like to improve the efficiency from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ to make it independent from the group size n.

One possibility is to use a cryptographic accumulator [28] (or a dynamic accumulator [11] to retain support for evolving groups) that makes it possible to accumulate multiple values into a single one such that for each accumulated value there is a proof that the value was correctly incorporated. Therefore, instead of using a 1-out-of-n "OR" proof, we could first accumulate all public keys and then prove that a client's public key is indeed a part of the the resulting short accumulator. Similar ideas were used to design a short linkage ring signature scheme [3], for example. Another possibility is to use more efficient proofs of knowledge [13] and new, efficient batching verification techniques for proofs of partial knowledge [36,37,49]. We fully expect to obtain a much efficient protocol using the outlined ideas.

5.2 Trading Deniability for Verifiability

DAGA offers a strong zero-knowledge notion of deniability; the protocol does not leave a 'paper trail" that one could use to prove that *some*, and therefore at least one, member participated in the protocol. DAGA achieves deniability by using an interactive rather than non-interactive zero-knowledge proof on the client's side. An interactive proof is not transferable and only "convinces" the party directly involved in the proof. In a non-interactive proof, the verifier is replaced with a hash function [29] to create an unpredictable challenge that a prover cannot anticipate in advance. Therefore, anyone can verify the non-interactive proof, even after the protocol's completion. This property, while useful, goes against the notion of deniability we set out to achieve. However, certain applications might benefit from the transferability of the proof that would allow for a *third-party verifiability* that some user or a certain number of users indeed authenticated.

Consider an anonymous voting scenario, where voters want to remain anonymous but wish for a third-party verifiable proof (independently of the election results) that a specific number of voters participated. A small change to DAGA, changing the client's proof from interactive to non-interactive, easily achieves this goal and each voter's authentication message M_0 becomes such a proof.

Interestingly, trading deniability for verifiability does not affect other properties, specifically forward anonymity and proportionality. Moreover, DAGA still retains a weaker notion of plausible deniability: since DAGA is anonymous and a group G can be created without the listed members' participation or knowledge, any member can plausibly deny participating in the protocol.

5.3 Optional Anonymity Revocation

DAGA provides clients with a strong notion of anonymity. However, the ability to revoke a client's anonymity might be a desirable feature but only if it is done carefully so that the client's anonymity is not inadvertently or maliciously compromised.

Any client's anonymity can be revoked if each server j reveals his per-round secret r_j, in which case the anonymity of all clients is compromised, or each server reveals his secret s_j shared with a client in question, breaking the rule of retaining private input secret, however. Therefore, we wish for a protocol which explicitly allows for anonymity revocation.

To achieve this goal, we use a threshold version of the ElGamal encryption scheme to encrypt the client's ephemeral private key z_i under a public key that is a product of all servers' commitments to their per-round secrets r (if we want to limit anonymity revocation to the lifetime of an authentication context) or under a public key that is a product of all servers' long-term public keys (if we want the ability to revoke clients' anonymity at any point).

After encrypting his ephemeral key z_i under a shared public key K_{all} of all servers, a client i produces a modified version of the PK^{client_i} proof which includes a proof that $E_{K_{all}}(z_i)$ is an encryption of an element committed to as Z_i, using a standard technique of proving a property of a ciphertext from [12]. To reveal a client's identity, all servers collectively decrypt $E_{K_{all}}(z_i)$, retrieve z_i and use it to recover all secrets the client shares with the servers' finally recovering a per-round generator, which corresponds to a unique client i as defined by $\vec{H}$.

This modification does not affect the properties offered by DAGA. Specifically, the anonymity and forward anonymity properties are still guaranteed, unless explicitly revoked, assuming that there is always one honest and never compromised server.

5.4 Secure on Full Key Exposure

Currently, the forward anonymity property holds as long as the honest server's private key is protected. If the long term private key z_h of the honest server is known, an adversary who controls all other servers can recover the ephemeral secret shared with a client i and calculate s_h. Then, if the adversary has access to the previous authentication messages that include the initial linkage tags, the adversary can trivially calculate $T_0^{\prime i} = h_i^{s_1 \dots s_m}$ for every $h_i \in \vec{H}$ and compare with the initial linkage tags. Knowing the association of a client's identity with a per-round generator, the adversary breaks the anonymity and forward anonymity of every client for whom he finds a matching tag.

We can avoid this (rather unlikely but not impossible) attack by adding a server-side per-round randomness into the secret a client shares with each server. This way even if the adversary compromises the server's private key, the additional secret included in s_h has been forgotten.

To do so, we extend the authentication context C to include an additional a vector $\vec{A} = A_1, \dots, A_m$, where $A_j = g^{a_j}$ and modify Step 2 of the client's protocol described in Section 4.5 as follows. For each server j, i uses both A_j and Y_j to compute a shared secret exponent $s_j = \mathcal{H}_1(A_j^{z_i}, Y_j^{z_i}) = \mathcal{H}_1(g^{a_j z_i}, g^{y_j z_i})$. Then, each server j recovers s_j as $\mathcal{H}_1((Z^{a_j}, Z^{y_j}) = \mathcal{H}_1(g^{a_j z_i}, g^{y_j z_i})$.

The protocol works as follows.

1. Client i encrypts his ephemeral key z_i under $R_S = \prod_{j=1}^{m} R_m$: as follows: i chooses $\ell \in_R \mathbb{Z}_q^*$ and calculates $E_{R_S}(z_i) = (A = g^r, B = z_i R_S^{\ell})$.

2. Then, client i creates a modified version of the PK$^{client}i$ proof appending to it a proof that $E_{R_S}(z_i)$ is an encryption of an element committed to as Z_i using a technique of proving a property of a ciphertext from [12].

In order to reveal an identity of a client, the servers perform the following steps.

1. Each server j calculates and publishes $A_j = A^{r_j} = g^{\ell r_j}$ as well as a proof of knowledge that $DL(A_j) = DL(R_j)$, that is j correctly computed A_j by raising it to its private key y_j.

2. All servers retrieve $z_i = B(\prod_{j=1}^{m} A_m)^{-1}$.

3. For each server $i \in G$, j calculates $s_i = \mathcal{H}_((Y)_i x^{z_i})$.

4. The client's identity is reveled by removing all secrets the client shares with the servers from a particular tag was created to: $(T_0)^{\prod_{i=1}^{m} s_i^{-1}}$ and deciding which generator the result is equal to.

$$PK\{(x_i, s, z_i) : \{\vee_{k=1}^{n}(X_k = g^{x_k} \wedge S_m = g^s \wedge T_0 = h_k^s) \wedge B = z_i g^{\sum_{j=0}^{m} r_j}\}$$

While we recognize that the client's identity can be revealed if each server reveals their secret shared with the client, we wish to be able to do so in a way that guarantees that a client is aware of this possibility (by creating a modified proof of knowledge). This modification does not affect the properties offered by DAGA. Specifically, the anonymity and forward anonymity properties are still guaranteed assuming that there is always one honest and never compromised server.

5.5 Delayed Revealing of Final Linkage Tags

After a successful authentication, the severs immediately learn the client's final linkage tag. However, the client might want to have an opportunity to "inspect" the tag first before finishing authentication. This way a client could check if the servers already seen such a tag by looking it up in a server-published list of the linkage tags seen in a particular authentication context C thereby avoiding the potential risk of unintentionally trying to authenticate twice in a linkage context, for example.

This can be easily accomplished by delaying the removal of the client-side secret s from the linkage tag until the client can verify $T_0 = h_i^{sr}$. That is, a client allows the servers to incorporate their per-round secrets r, then verifies the resulting tag, and if the tag is correct, he removes his secret s, proves in zero-knowledge that he did so correctly and sends the final tag back to the servers.

To do so, a client creates the initial linkage tag as before, $T_0 = h_i^s$, however now s is a single secret known by the client, not a product of all secrets i assigns to the servers, and i produces a proof PK^{client_i} as before.

Upon receiving the client's message M_0, the servers iteratively incorporate their per-round secrets r_j as before, but this time *without* removing the client's secret, finally yielding a final linkage tag $T_f = h_i^{s\prod_{k=1}^{m} r_k}$. Each server j prepares a simplified proof of its correctness:

$$PK\{(r_j) : T_j = T_{j-1}^{r_j} \wedge R_j = g^{r_j}$$

After all servers process the tag, the last server sends T_j and the proofs of its correctness back to the client, who can verify the proofs and calculate the client's final linkage $T_f = (T_j)^{-s}$. This way the client has a chance to inspect the tag *before* making it available to the servers. If the client decides to complete the authentication, he can prove the correctness of T_f with respect to T_j as follows

$$PK\{(s) : T_f = T_j^s \wedge S = g^s\}$$

While this approach gives the client more control over his authentication requests, it requires an additional communication round, but might be suitable for applications where clients are limited to a certain number of authentications within a certain authentication context and authenticating more than the allowed number of times has negative consequences.

6 Practical Considerations

6.1 Servers' Liveness

DAGA depends on a set of servers to process each authentication request. Therefore, if a server goes offline or refuses to process a message, the protocol stalls or aborts. While we cannot guarantee that DAGA terminates if one of the above happens, we can employ a wrapper protocol that uses gossip techniques such as those used in PeerReview [35] to ensure liveness.

6.2 Dealing with Dishonest Servers

Before processing an incoming authentication message, each server j verifies all proofs of correctness of every server that comes before j. If an invalid proof $\mathrm{PK}_1^{server_k}$ or $\mathrm{PK}_2^{server_{k(i)}}$ for some server k is discovered, the authentication must be aborted and the client cannot be authenticated. We assume that the issue of dealing with dishonest servers within the anytrust set is done administratively [60, 61].

6.3 Authentication Context

Generating an authentication context In order to establish a new authentication context, the servers need to define the clients who belong to a group G and establish servers' per-round secrets and clients' per-round generators.

Step 1: First, the servers choose a safe prime $p = 2q + 1$ where q is a sufficiently large prime a generator g of a prime order q group $\mathcal{G}$.

Step 2: Each server j picks a per-round secret $r_j \in_R \mathbb{Z}_q^*$, which is kept secret, and then j sends to other servers a commitment $R_j = g^{r_j}$.

Step 3: Servers collectively establish a random per-round generator h_i for each client i such that no one knows the logarithmic relationship between h_i and g, or between h_i and $h_{i'}$ for any pair of clients $i \neq i'$, for example using a technique described in Section 6.5.

Step 4: Servers create a set of the servers' commitments $\vec{R} = (R_1, \ldots, R_m)$ and clients' generators $\vec{H} = (h_1, \ldots, h_n)$. Then, the servers publish an authentication context $\mathrm{C} = (G, \vec{R}, \vec{H}, p, g)$.

Validity of an authentication context An authentication context might be one time, where each client is expected to make exactly one authentication request or a context may remain valid for certain period of time or some maximum number of authentications made by a single clients or all of clients in G. Since the servers can keep track of each anonymous client's authentication request, a client may be allowed to make up to k requests so that each request beyond that is rejected regardless of the validity of the supplied authentication message. After a context expires, all servers securely erase their per-round secrets r making it impossible to process authentication messages within this context.

Updating an authentication context DAGA supports the evolution of the clients is a particular group G included a context C in a way that preserves the proportionality property within that context. A new client k may be efficiently added to G, by simply adding his public key X_k to $\vec{X}$ and adding a new generator h_k to $\vec{H}$. The proportionality property is preserved, because each client's linkage tag only depends on the client's generator and the servers' per-round secrets making it independent of the membership of G. After the context is updated, each client would create PK^{client_i} with respect to the new group G. Care needs to be taken to propagate the updated context to all clients to avoid accidentally compromising the identity of the newly added client as he would be the only one using the updated context.

6.4 Challenge Generation

A client i produces a proof of knowledge using an interactive honest-verifier zero-knowledge proof of knowledge as described in Section 4.7. Because the proof is interactive, a client i must obtain a random challenge c_s from the servers after submitting his commitments. Additionally, because the proof is honest-verifier, the challenge must be indeed randomly chosen. This can be ensured by requiring all servers to collectively generate c_s so that each server, which would include at least one honest server, contributes its randomness towards c_s.

One approach to collectively establish c_s is as follows.

Step 1: Upon receiving a client's request, server j assumes the role of a leader and requests that the other servers generate a new challenge c_s for client i.

Step 2: Each server i chooses $c_i \in_R \mathbb{Z}_q^*$ and then calculates a commitment C_i. Server i signs and publishes C_i.

Step 3: Upon receiving C_i from every other server i, server j verifies if all C_i are of valid form and properly signed, and if yes server j publishes an opening c_j of his commitment C_j and requests other serves to open their commitments.

Step 4: Upon receiving an opening c_i from every other server i, server j verifies if every c_i is indeed a valid opening of C_j. If yes, server j calculates $c_s = c_1 + \cdots + c_m$. Server j collects all commitments C_i, openings c_i, and the calculated challenge c_s and forwards to server $j+1$ who signs c_s after verifying that it was correctly calculated.

Step 5: Upon receiving c_s signed by every other server, server j forwards c_s for the client along with a proof that every other server calculated the same value.

Under our assumption, there is at least one honest server h who will randomly choose his c_j and therefore guarantee that the collective challenge c_s is properly generated.

6.5 Per-Round Generators

For each protocol round, defined by the same context C, we require that there is a set $\vec{H} = (h_1, \ldots, h_n)$ of n per-round generators of $\mathcal{G}$, where there is one unique generator h_i for each client i. The proportionality property depends on the uniqueness of the final linkage tags. Each client i's linkage tag T_f^i is unique and remains fixed within the same C, precisely because each client i creates the initial tag T_i^0 with respect to the same but unique per-round generator h_i.

As defined in Section 4.4, $\mathcal{G}$ is a multiplicative cyclic group of prime order q. Therefore, all elements of $\mathcal{G}$, except for the identity element, are generators of $\mathcal{G}$ so generating $\vec{H}$ reduces to choosing n random elements of $\mathcal{G}$.

However, it is important that $\vec{H}$ is chosen randomly to ensure that the assumption no one knows the logarithmic relationship between any h_i and g or between h_i and $h_{i'}$ for any pair of clients $i \neq i'$ holds. Therefore, the anytrust servers must collectively choose $\vec{H}$ in a way that ensures that none of the servers know the aforementioned logarithmic relationships. An efficient method to find generators is to use a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q^*$ to map a per-round fixed string $(i, \vec{R})$ into each client i's generator.

7 Evaluation

7.1 Implementation

We have implemented DAGA within the context of Dissent [61] using C++ with the Qt framework and the CryptoPP cryptography library. The prototype implements both the client and server aspects of DAGA, but currently does not support exposing misbehaving clients nor any of the extensions to DAGA, discussed in Section 4.9 and Section 5, respectively. The prototype assumes that all keys derive from the same modulus and subgroup, and that all participants have used an outside channel to agree upon a common set of authentication servers and an authentication context. With the introduction of DAGA, Dissent now includes a modular authentication framework that supports pre-exchanged keys using Stinson's two-way authentication protocol [55] (Protocol 9.6), linkable ring signatures (LRS) [44], and DAGA.

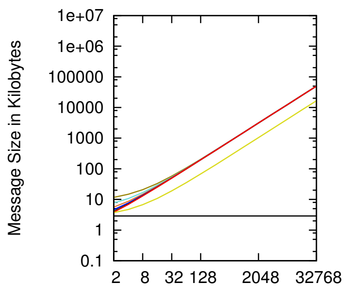
7.2 Micro benchmarks

We evaluate DAGA in comparison to pre-exchanged keys and LRS. The evaluations were performed on a 64-bit x86 machine running Ubuntu 12.04. This evaluation simulates the authentication of a client to one or more servers within a single process. All communication between parties occurs through bytestreams as if they were sent over the network. Both the authentication time for a single client and the amount of data transmitted during this authentication were recorded. All client and servers keys derive from a common 2048-bit DSA key. For both DAGA and LRS, the number of clients varied from 2 to 32768 by powers of 2. Only DAGA depends on more than one server for authentication. Both the LRS and DAGA depend on a linkage context. For this evaluation, we assume that the administrators of the authentication systems have agreed upon and distributed the linkage context along with the set of the group's public keys.

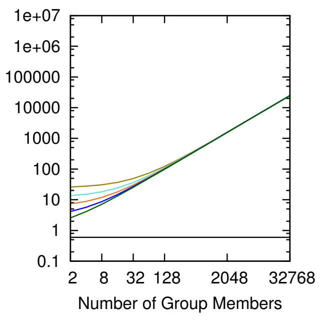
Figure 2d shows the total system traffic during a single client authentication for the various forms of authentication and group configurations. The traffic results have been broken down into client to server, server to client, and server to server traffic in figures 2a, 2b, and 2c, respectively. As expected, pre-exchanged key authentication does not depend on the number of clients in the group. DAGA authentication transfers more data in all three cases and uniquely has the requirement that servers communicate with each other during an authentication. LRS authentication involves a non-interactive zero knowledge proof, therefore has constant traffic from server to client. Finally, all forms of DAGA traffic grow linearly in the number of clients and nearly linearly in the number of servers, while only LRS client to server traffic grows linearly.

The time for authentication, Figure 2e, exhibits similar characteristics to that of the traffic for the respective authentication techniques. In this scenario, however, unlike traffic, DAGA and LRS computation time remain competitive, particularly when using 4 or less DAGA servers.

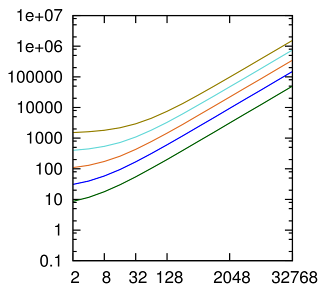
While DAGA compares well with other anonymous authentication schemes, like LRS, the performance concerns remain. In order to remain anonymous among k individuals, anonymous authentication systems traditionally require linear computation. Using more efficient DSA keys, such as, those derived from elliptic curves, would reduce computation



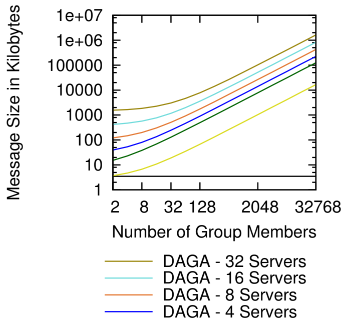
(a) Client to Server Traffic



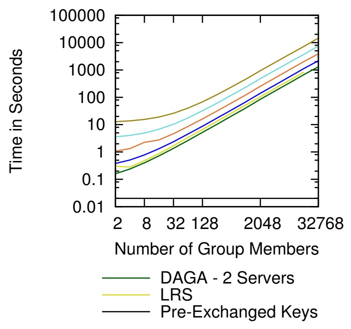
(b) Server to Client Traffic



(c) Server to Server Traffic



(d) System Traffic



(e) Authentication Time

Figure 2: Time and traffic comparison among DAGA, LRS, and pre-exchanged key authentication

and traffic load for these style of protocols including DAGA and LRS.

8 Security Properties and Analysis

This section describes and analyzes DAGA's security properties.

8.1 Assumptions

We assume that the Discrete Logarithm (DL) and Decisional Diffie-Hellman (DDH) assumptions hold, that is, any probabilistic polynomial algorithm solves the DL problem and the DDH problem respectively only with a negligible probability [5]. DAGA assumes a cyclic multiplicative group $\mathcal{G}$ of prime order q, where $p = 2q + 1$, where the Discrete Logarithm and Decisional Diffie-Hellman assumptions hold in $\mathcal{G}$ [5].

8.2 Properties of the Proofs of Knowledge

In this section we show that the client's and server's proofs of knowledge have the properties of completeness, special soundness and special honest-verifier zero-knowledge [22]. Note that $\mathcal{H}$ is modeled as a random oracle.

Definition 1 (Σ-protocol [22]). *A protocol $\mathcal{P}$ is said to be a Σ-protocol for relation R if:*

- *P is of the 3-move form, and if P, V follow the protocol, the verifier always accepts* (completeness)*.*
- *From any x and any pair of accepting conversations on input x, (t, c, r), (t, c', r') where $c \neq c'$, one can efficiently compute w such that $(x, w) \in R$* (special soundness)*.*
- *There exists a polynomial time simulator S_{zk}, which on input x and a random e outputs an accepting conversation of the form (t, c, r), with the same probability distribution as conversations between the honest P, V on input x* (special honest-verifier zero-knowledge)*.*

8.2.1 PK^{client}: Client's Proof of Knowledge

Completeness. If a prover and verifier faithfully follow the protocol on common input C and prover's private input x, then the verifier always accepts the proof generated by the prover. Assume that the proof P_0 is generated by a client i who knows a solution x_i, his private key, and s, the product of all secrets shared with the servers. The verifier checks the commitment t and the challenge c_s. For client i, the commitment verification proceeds as follows.

$$\begin{aligned} t_{i.0} &\overset{?}{=} X_i^{c_i} g^{r_{i.0}} & \qquad t_{i.10} &\overset{?}{=} S_m^{c_i} g^{r_{i.1}} \\ g^{v_{i.0}} &= X_i^{c_i} g^{r_{i.0}} & g^{v_{i.1}} &= S_m^{c_i} g^{r_{i.1}} \\ g^{v_{i.0}} &= g^{c_i x_i} g^{v_{i.0} - c_i x_i} & g^{v_{i.1}} &= h_i^{c_i s} g^{v_{i.1} - c_i s} \\ g^{v_{i.0}} &= g^{c_i x_i} g^{v_{i.0}} g^{-c_i x_i} & g^{v_{i.1}} &= g^{c_i s} g^{v_{i.1}} g^{-c_i s} \\ g^{v_{i.0}} &= g^{v_{i.0}} & g^{v_{i.1}} &= g^{v_{i.1}} \end{aligned}$$

$$\begin{aligned} t_{i.11} &\overset{?}{=} T_0^{c_i} h_i^{r_{i.1}} \\ h_i^{v_{i.1}} &= T_0^{c_i} h_i^{r_{i.1}} \\ h_i^{v_{i.1}} &= h_i^{c_i s} h_i^{v_{i.1} - c_i s} \\ h_i^{v_{i.1}} &= h_i^{c_i s} h_i^{v_{i.1}} h_i^{-c_i s} \\ h_i^{v_{i.1}} &= h_i^{v_{i.1}} \end{aligned}$$

For every client $j \neq i$, the commitment verification proceeds as follows.

$$\begin{aligned} t_{j.0} &\overset{?}{=} X_j^{c_j} g^{r_{j.0}} & t_{j.10} &\overset{?}{=} S_m^{c_j} g^{r_{j.1}} \\ X_j^{w_j} g^{v_{j.0}} &= X_j^{c_j} g^{r_{j.0}} & S_m^{w_j} g^{v_{j.1}} &= S_m^{c_j} g^{r_{j.1}} \\ X_j^{w_j} g^{v_{j.0}} &= X_j^{w_j} g^{v_{j.0}} & S_m^{w_j} g^{v_{j.1}} &= S_m^{w_j} g^{v_{j.1}} \end{aligned}$$

$$\begin{aligned} t_{j.11} &\overset{?}{=} T_0^{c_j} h_j^{r_{j.1}} \\ T_0^{w_j} h_j^{v_{j.1}} &= T_0^{c_j} h_j^{r_{j.1}} \\ T_0^{w_j} h_j^{v_{j.1}} &= T_0^{w_j} h_j^{v_{j.1}} \end{aligned}$$

The challenge verification proceeds as follows.

$$\begin{aligned} c_s &\overset{?}{=} c \\ c_s &= \sum_{i=1}^{n} c_i \\ c_s &= \sum_{i=1}^{k} w_k + c_s - \sum_{i=1}^{k} w_k \\ c_s &= c_s \end{aligned}$$

As shown above, the verifier will be able to successfully verify the commitments t and c based on the challenge c_s, hence, the proof is complete.

Special Soundness. Given common input i and two transcripts of successful conversations $(t, c = (c_1, \ldots, c_n), r = (r_1, \ldots, r_n))$ and $(t, c' = (c'_1, \ldots, c'_n), r' = (r'_1, \ldots, r_n))$, where $c \neq c'$, client i's private input x_i and s can be successfully computed as follows:

$$\begin{aligned} r_{i.0} &= v_{i.0} - c_i x_i & r_{i.1} &= v_{i.1} - c_i s \\ r'_{i.0} &= v_{i.0} - c'_i x_i & r'_{i.1} &= v_{i.1} - c'_i s \\ x_i &= \frac{r_{i.0} - r'_{i.0}}{c_i - c'_i} & s &= \frac{r_{i.1} - r'_{i.1}}{c_i - c'_i} \end{aligned}$$

Special Honest Verifier Zero-Knowledge. There exists a polynomial time simulator S_{zk}, which on common input I generates a conversation transcript that is computationally indistinguishable from a transcript generated by a prover. The simulator S_{zk} works as follows:

1. Choose $w_1, \ldots, w_n \in_R \mathbb{Z}_q^*$ for all i and $v_{1.0}, v_{1.1}, \ldots, v_{n.0}, v_{n.1} \in_R \mathbb{Z}_q^*$ for all i. Compute commitments $t_{i.0} = X_i^{w_i} g^{v_{i.0}}$, $t_{i.10} = S_m^{w_i} g^{v_{i.1}}$, and $t_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$ for each i.
   Set $t = (t_{1.0}, t_{1.10}, t_{1.11}, \ldots, t_{n.0}, t_{n.10}, t_{n.11})$,

2. Compute $c_s = \sum_{k=1}^{n} w_k$. Set $c_i = w_i$ for each i and set $c = (c_1, \ldots, c_n)$.
3. Compute responses $r = (r_{1.0}, r_{1.1}, \ldots, r_{i.0}, r_{i.1})$ using $r_{i.k} = v_{i.k}$ for all $1 \leqslant i \leqslant n$ and $k \in \{0, 1\}$.
4. Set $P = (c_s, t, c, r)$.

The verification of the proof works as follows for every i:

$$\begin{aligned} t_{i.0} &\overset{?}{=} X_i^{c_i} g^{r_{i.0}} & t_{i.10} &\overset{?}{=} S_m^{c_i} g^{r_{i.1}} \\ X_i^{w_i} g^{v_{i.0}} &= X_i^{c_i} g^{r_{i.0}} & S_m^{w_i} g^{v_{i.1}} &= S_m^{c_i} g^{r_{i.1}} \\ X_i^{w_i} g^{v_{i.0}} &= X_i^{w_i} g^{v_{i.0}} & S_m^{w_i} g^{v_{i.1}} &= S_m^{w_i} g^{v_{i.1}} \end{aligned}$$

$$\begin{aligned} t_{i.11} &\overset{?}{=} T_0^{c_i} h_i^{r_{i.1}} & c_s &\overset{?}{=} \sum_{k=1}^{n} c_k. \\ T_0^{w_i} h_i^{v_{i.1}} &= T_0^{c_i} h_i^{r_{i.1}} & \sum_{k=1}^{n} w_k &= \sum_{k=1}^{n} w_k \\ T_0^{w_i} h_i^{v_{i.1}} &= T_0^{w_i} h_i^{v_{i.1}} & & \end{aligned}$$

8.2.2 $\mathbf{PK}_1^{server}$: Proving correctness of its work

Completeness. The verifier reconstructs the commitments as follows:

$$\begin{aligned} t'_1 &= T_{j-1}^{r_1} T_j^{-r_2} \\ &= T_{j-1}^{v_1 - c r_j} T_j^{-(v_2 - c s_j)} \\ &= T_{j-1}^{v_1} T_{j-1}^{-c r_j} T_j^{-v_2} T_j^{c s_j} \\ &= T_{j-1}^{v_1} T_j^{-v_2} \\ &= t_1 \end{aligned}$$

$$\begin{aligned} t'_2 &= g^{r_1} R_j^c & t'_3 &= S_{j-1}^{r_2} S_j^c \\ &= g^{v_1 - c r_j} g^{c r_j} & &= S_{j-1}^{v_2 - c s_j} S_{j-1}^{c s_j} \\ &= g^{v_1} g^{-c r_j} g^{c r_j} & &= S_{j-1}^{v_2} S_{j-1}^{-c s_j} S_{j-1}^{c s_j} \\ &= g^{v_1} & &= S_{j-1}^{v_2} \\ &= t_2 & &= t_3 \end{aligned}$$

Given that $t'_1 = t_1$, $t'_2 = t_2$ and $t'_3 = t_3$, we have

$$\begin{aligned} c &\overset{?}{=} \mathcal{H}(T_{j-1}, T_j, R_j, g, S_j, S_{j-1}, t'_1, t'_2, t'_3) \\ c &= c \end{aligned}$$

Special Soundness. Given common input i and two transcripts of successful conversations $(t_1, t_2, t_3, c, r_1, r_2)$ and $(t_1, t_2, t_3, c', r'_1, r'_2)$, where $c \neq c'$, server j's private input r_j and

s_j can be successfully computed as follows:

$$\begin{array}{ll} r_1 = v_1 - cy_j & r_2 = v_2 - c_i s_j \\ r_1' = v_1 - c' y_j & r_2' = v_2 - c_i' s_j \\ r_j = \dfrac{r_1 - r_1'}{c - c'} & s_j = \dfrac{r_2 - r_2'}{c - c'} \end{array}$$

Special Honest Verifier Zero-Knowledge. The simulator S_{zk} accepts h as input and performs the following steps to produce $P_j = P$:

1. Choose $v_1, v_2 \in_R \mathbb{Z}_q^*$.
2. Set $c = h$.
3. Calculate $t_1 = T_{j-1}^{v_1} T_j^{-v_2}, t_2 = g^{v_1} R_j^c, t_3 = S_{j-1}^{v_2} S_j^c$.
4. Set $r_1 = v_1$ and $r_2 = v_2$.
5. Set $P = (t_1, t_2, t_3, c, r_1, r_2)$.

The verification of the proof works as follows:

$$\begin{array}{lll} t_1' = T_{j-1}^{r_1} T_j^{-r_2} & t_2' = g^{r_1} R_j^c & t_3' = S_{j-1}^{r_2} S_j^c \\ \quad = T_{j-1}^{v_1} T_j^{-v_2} & \quad = g^{v_1} R_j^c & \quad = S_{j-1}^{v_2} S_j^c \\ \quad = t_1 & \quad = t_2 & \quad = t_3 \end{array}$$

Then, we verify the challenge $c \overset{?}{=} h$, which gives $h = h$.

8.2.3 $\mathbf{PK}_2^{server}$: Exposing a misbehaving client

Completeness. The verifier reconstructs the commitments as follows:

$$\begin{array}{ll} t_1' = Z^r Z_{s_j}^c & t_2' = g^r Y_j^c \\ \quad = Z^{v-cy_j} Z^{cy_j} & \quad = g^{v-cy_j} g^{cy_j} \\ \quad = Z^v Z^{-cy_j} Z^{cy_j} & \quad = g^v g^{-cy_j} g^{cy_j} \\ \quad = Z^v & \quad = g^v \\ \quad = t_1 & \quad = t_2 \end{array}$$

Given that $t_1' = t_1$ and $t_2' = t_2$,

$$\begin{aligned} c &\overset{?}{=} \mathcal{H}(Z_{s_j}, Z, Y_j, g, t_1', t_2') \\ \mathcal{H}(Z_{s_j}, Z, Y_j, g, t_1, t_2) &= \mathcal{H}(Z_{s_j}, Z, Y_j, g, t_1', t_2') \end{aligned}$$

Special Soundness. Given common input i and two transcripts of successful conversations (t_1, t_2, c, r) and (t_1, t_2, c', r'), where $c \neq c'$, server j's private input y_j can be successfully computed as follows:

$$\begin{aligned} r &= v - c_i y_j \\ r' &= v - c_i' y_j \\ y_j &= \frac{r - r'}{c - c'} \end{aligned}$$

Special Honest Verifier Zero-Knowledge. The simulator S_{zk} accepts h as input and performs the following stepts to produce $P_j = (Z_{s_j}, P)$:

1. Choose $v \in_R \mathbb{Z}_q^*$. Calculate $c = h$.
2. Calculate $t_1 = Z^v Z_{s_j}^c$ and $t_2 = g^v Y_j^c$.
3. Set $r = v$
4. Set $P = (t_1, t_2, c, r)$.

The verification of the proof works as follows:

$$\begin{aligned} t_1' &= Z^r Z_{s_j}^c & t_2' &= g^r Y_j^c \\ &= Z^v Z_{s_j}^c & &= g^v Y_j^c \\ &= t_1 & &= t_2 \end{aligned}$$

Then, we verify the challenge as follows:

$$\begin{aligned} c &\stackrel{?}{=} h \\ h &= h \end{aligned}$$

8.3 Completeness

We require that servers accept a properly formed authentication request from every honest client i who belongs to a group G defined by a particular authentication context C, unless the protocol is aborted because of a discovered misbehavior of some server. A client i *belongs* to a group G if he knows a private key x_i such that $X_i = g^{x_i} \in \vec{X}$.

Definition 2. *An authentication protocol offers the* completeness *property, if for any client $i \in G = (\vec{X}, \vec{Y})$ who correctly follows the prescribed protocol, the servers accept i 's authentication request with an overwhelming probability.*

Theorem 1. DAGA *offers the* completeness *property.*

Proof. Under our assumptions, a client i belongs to a group G, is in a possession of a private key x_i such that $X_i = g^{x_i}$ and $X_i \in \vec{X}$. Further, we assume that i is in possession of a well-formed authentication context C $= (G, \vec{R}, \vec{H}, p, g)$ where $G = (\vec{X}, \vec{Y})$ is the group definition, $H = (h_i, \ldots, h_n)$ is a set of per-round generators for clients, and $R = (R_1, \ldots, R_m)$ is server-published randomness. From the trust model it follows that all servers participate in the round and all server's but one can behave arbitrarily dishonestly.

In order to make an authentication request, a client i must prepare an authentication message $\mathrm{M}_i^0 = (C, S, T_0, P_0)$, where C is the authentication context, $S = (Z, \vec{S} = S_0, \ldots, S_m)$ consists of the client's ephemeral public key and the set of client's commitments, T_0 is the initial linkage tag, and P_0 is a proof of correctness for T_0. We will show that client i is able to produce a valid message M_i^0 and that this message will be accepted by the servers and therefore result in an accepted authentication request.

To produce a valid message M_i^0, client i needs to produce all of its components.

- *Authentication context* C*.* Client i obtains C before making an authentication request.
- *Ephemeral key and commitments* S*.* Client i can produce S since it depends on i's randomly chosen key and information included in C.
- *Linkage tag* T_0*.* The tag $T_0 = h_i^{s_1 s_2 \ldots s_m}$ depends on i's publicly available per-round generator h_i and secrets $s_1, s_2, \ldots, s_m$ created in the previous step.
- *Proof* P_0*.* The proof P_0 depends on the knowledge of x_i and s. Based on the completeness property of the underlying proof of knowledge, a client i can always produce a valid proof if he's in possession of the private information he tries to prove knowledge of. In our case, i creates a proof of knowledge $PK(x_i, s)$, where x_i is i's private key and $s = s_1 s_2 \ldots s_m$.

Therefore, i can produce a valid message $\mathrm{M}_i^0 = (C, S, T_0, P_0)$. Now, we will show that this message will be accepted by the servers with an overwhelming probability and therefore result in an accepted authentication request.

After creating M_i^0, i sends it to an arbitrarily chosen server j. Each server j verifies the message M_i^0 and either *(i)* accepts it and produces an outgoing tag T_j and a proof $\mathrm{PK}_1^{server_j}$ of correctness of its own work or *(ii)* rejects it and produces a proof $\mathrm{PK}_2^{server_{j(i)}}$ of the client i's misbehavior. By the soundness property of the underlying proof of knowledgePK_2^{server}, none of the servers can expose i as dishonest and therefore reject i's authentication request, except with a negligible probability, given that i's M_i^0 is valid. By the soundness property of PK_1^{server}, none of the servers can produce a valid proof of correctness if they did not follow the protocol, except with a negligible probability. Therefore, if the protocol is not terminated and a faulty server discovered, each server produces a valid intermediate tag T_j, which results in a valid final tag T_f^i. Consequently, i's authentication request is accepted with with an overwhelming probability. □

8.4 Soundness

We require that DAGA is a sound authentication protocol, that is, servers only accept properly formed authentication requests from members who belong to a particular group G as defined in an authentication context C. This means that it should offer soundness and not allow forgeries, where the notions of forgeability from [33] apply.

Typically, the soundness property of a non-anonymous authentication protocol is defined with respect to "legitimate" users that have established credentials with a verifier. In case of DAGA, we define legitimate users as users who belong to a particular group G. Any client that possesses a correct long-lived well-known key pair associated with a non-anonymous identity can belong to any G, even without their knowledge. This is true because any member i can be listed in G if their public key is publicly available since there is no action required on the client's side in order to be added to G. We assume that honest clients keep their private keys secret and dishonest clients can arbitrarily share their private keys. In such a case, we note that if i is in possession of a private key $\hat{\imath}$ such that $X_{\hat{\imath}} = g^{\hat{\imath}}$ and $X_{\hat{\imath}} \in \vec{X}$, then i can successfully impersonate $\hat{\imath}$ by authenticating as a legitimate client and does not violate the soundness property.

Definition 3. *An authentication protocol offers the* soundness *property if an authentication request from any client $i \notin G$ is rejected with an overwhelming probability.*

Theorem 2. DAGA *offers the* soundness *property.*

Proof. Assume that a client i does not belong to a group G, that is, i's public key X_i is not included in $\vec{X}$. Therefore, i does not know any $x_{\hat{\imath}}$ such that $X_{\hat{\imath}} = g^{x_{\hat{\imath}}}$ and $X_{\hat{\imath}} \in \vec{X}$. To successfully authenticate as a member of G, i needs to prepare a message $\mathrm{M}_{\hat{\imath}}^{0}$ on behalf of some $\hat{\imath} \in G$. To do so, i must prepare a valid message $\mathrm{M}_{\hat{\imath}}^{0} = (C, S, T_0, P_0)$ without the knowledge of $x_{\hat{\imath}}$, such that each server accepts i's authentication request.

We will show that i cannot produce a valid message $\mathrm{M}_{\hat{\imath}}^{0}$, except with a negligible probability, and each message $\mathrm{M}_{\hat{\imath}}^{\prime 0}$ i can produce will be rejected by the servers because of the invalid proof. Hence, i's authentication requests will be denied with an overwhelming probability.

In order to forge a message $\mathrm{M}_{\hat{\imath}}^{\prime 0}$, a client i must forge its individual elements, that is C, S, T_0 and P_0.

- *Authentication context C.* C is publicly available and so i has access to a valid authentication context.
- *Ephemeral key and commitments S.* Client i can produce a valid S since it does not depend on $x_{\hat{\imath}}$. To do so, i chooses $z \in_R \mathbb{Z}_q^*$ and calculates $S = (Z, \vec{S})$, where $\vec{S} = S_0, \ldots, S_m$, since it only depends on the knowledge of z, a generator g and the set of servers' public keys $\vec{Y}$ included in C.
- *Linkage tag T_0.* To calculate the tag $T_0 = h_{\hat{\imath}}^{s_1 s_2 \ldots s_m}$, i uses $h_{\hat{\imath}}$ included in C and and secrets $s_1, s_2, \ldots, s_m$ created in the previous step.
- *Proof P_0.* The last piece i must produce is the proof PK^{client} which depends on the knowledge of $x_{\hat{\imath}}$ and s. i knows one of the secrets, namely s, but he does not know client $\hat{\imath}$'s private key $x_{\hat{\imath}}$.

By the soundness property of the underlying proof of knowledge PK^{client_i}, i cannot produce a valid proof P_0 on behalf of some $\hat{\imath} \in G$, except with a negligible probability. Therefore, i must forge P_0 and create an authentication message $\mathrm{M}_{\hat{\imath}}^{\prime 0}$ that includes the forged proof. By the *anytrust* assumption, there exists at least one honest server and therefore i's message will be eventually rejected. If i can forge a proof P_0, however, such that each honest server j accepts the proof as coming from a member $\hat{\imath}$ with non-negligible probability, then i must be able to find $x'_{\hat{\imath}}$ such that $X_{\hat{\imath}} = g^{x'_{\hat{\imath}}}$. This, however, is impossible under the Discrete Logarithm assumption, except with a negligible probability. Therefore, each authentication request from $i \notin G$ is rejected with an overwhelming probability. $\square$

8.5 Anonymity

Informally, we want to ensure that an adversary cannot guess which member has been authenticated with a probability greater than random guessing. We will show that DAGA provides anonymity under the Decisional Diffie-Hellman assumption in the random oracle model [4].

We argue that in order to break a client i's anonymity, an adversary must leverage the linkage tags because he cannot infer the client's identity based on a proof PK^{client} under the zero-knowledge property of the proof.

An adversary cannot infer the identity of a client based on a proof of knowledge PK^{client} because the proof is zero-knowledge and does not leak the identity of its creator. Consequently, the adversary must focus on the initial, intermediate or final tags of a particular client in hopes of discovering the client's identity. After observing a protocol run, the adversary sees the initial tag T_0, all the intermediate linkage tags T_j, a final linkage tag T_f, and all partial intermediate tags of the servers he controls. By using per-round secrets r_j and s_j of every dishonest server j, the adversary obtains $T_0 = h_{\hat{\imath}}^{s_h}$ and $T_f = h_{\hat{\imath}}^{r_h}$, two tags protected by the per-round secrets r_h and s_h of an honest server. For simplicity, assume that there are only two clients $\{i, j\} \in G$, hence, the adversary's goal is to decided whether $\hat{\imath} = i$ or $\hat{\imath} = j$ based on the tags he obtained but *without* the knowledge of either s_h or r_h. Because both h_i and h_j are generators of $\mathcal{G}$, then both generators generate the entire group $\mathcal{G}$ when raised to $x \in \{1, \ldots, q\}$. Therefore, $h_i^{s_h}, h_j^{s_h} \in \mathcal{G}$ as well as $h_i^{r_h}, h_j^{r_h} \in \mathcal{G}$, and each element is a random and indistinguishable element of $\mathcal{G}$. Moreover, by the properties of $\mathcal{G}$, there exists $s_1, s_2, r_1, r_2, h_1, h_2 \in \mathcal{G}$ such that $h_i^{s_1} = h_j^{s_2}$ and $h_i^{r_1} = h_j^{r_2}$, hence, the tag can be created with respect to h_i and h_j equally likely.

Definition 4. *An authentication protocol is* anonymous *if for any probabilistic polynomial time adversary $\mathcal{A}$, the probability $p(n)$ that $\mathcal{A}$ wins the anonymity game is negligible s.t. $|p(n) - \frac{1}{2}| = negl(n)$.*

The following *anonymity* game is played between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

1. The challenger $\mathcal{C}$ randomly generates all private and public keys for every client $i \in G$ and for every server $j \in G$, $(X_i = g^{x_i}, x_i)$ and $(Y_j = g^{y_j}, y_j)$ respectively.

2. The adversary chooses two honest members i and j, both of which belong to G.

3. The challenger gives the adversary the public keys of all clients and all servers, and the private keys of the $n-2$ dishonest clients $(G \backslash \{i, j\})$ and $m-1$ dishonest servers.

4. The adversary is allowed to run the protocol polynomially-many times for any member $k \in G \backslash \{i, j\}$.

5. The challenger chooses a bit $b \in \{0, 1\}$ uniformly at random. If $b = 0$, then the challenger chooses member i to participate in the protocol and chooses member j otherwise.

6. The challenger participates in the authentication protocol playing the role of all honest servers and the chosen honest member. The adversary participates in the authentication protocol playing the role of the dishonest members and the dishonest servers.

7. After the challenge protocol run, the adversary is allowed to run the protocol polynomially many times for any member $k \in G \backslash \{i, j\}$. Finally, the adversary outputs his guess $b' = \{0, 1\}$. The adversary wins the anonymity game if $b' = b$.

Theorem 3. DAGA *offers the* anonymity *property.*

We will show that if there exists a polynomial time adversary $\mathcal{A}_{\text{ANON}}$ that breaks the anonymity property with non-negligible property, then we can use this adversary to create an adversary $\mathcal{A}_{\text{DDH}}$ that solves the Decisional Diffie-Hellman problem with non-negligible probability.

Proof. An adversary $\mathcal{A}$ has two choices for his behavior: (1) $\mathcal{A}$ can play the role of the dishonest entities and deviate from the protocol in an arbitrary way, or (2) $\mathcal{A}$ can follow the prescribed protocols and try to break the anonymity property by observing the protocol runs. Briefly, DAGA requires that each entity produces a proof of correctness, hence, if the adversary does not follow the protocol, he will fail to produce the required output by the soundness property of the underlying proofs of knowledge, and the protocol will abort. Therefore, in order to break the anonymity property, $\mathcal{A}$ follows the prescribed protocols.

Assume that there exists a probabilistic polynomial time $\mathcal{A}_{\text{ANON}}$ that breaks the anonymity property with a non-negligible probability and therefore has a non-negligible advantage ϵ_{ANON} in the anonymity game. We will show that if $\mathcal{A}_{\text{ANON}}$ exists, then we can use $\mathcal{A}_{\text{ANON}}$ as a subroutine to another probabilistic polynomial time adversary $\mathcal{A}_{\text{DDH}}$ that solves the DDH problem with non-negligible probability.

$\mathcal{A}_{\text{DDH}}$ plays the DDH game, receives a challenge tuple (g, g^a, g^b, g^c) from the DDH challenger, and outputs 0 if (g^a, g^b, g^c) is a Diffie-Hellman tuple, that is $c = ab$, otherwise $\mathcal{A}_{\text{DDH}}$ outputs 1. $\mathcal{A}_{\text{DDH}}$ uses $\mathcal{A}_{\text{ANON}}$ as a subroutine and therefore must simulate the $\mathcal{A}_{\text{ANON}}$'s view of its interaction with the challenger in the anonymity game. Because all the client's and server's proof of knowledge are zero-knowledge, $\mathcal{A}_{\text{DDH}}$ can efficiently simulate all proofs in the random oracle model [4]. Therefore, we omit the proofs in the description below keeping in mind that they can be simulated and correctly verified. $\mathcal{A}_{\text{DDH}}$ simulates the view of $\mathcal{A}_{\text{ANON}}$ as follows.

Step 1: $\mathcal{A}_{\text{DDH}}$ creates an authentication context C as prescribed in the protocol except that he uses g^a from the DDH challenge tuple as the public key Y_h of the honest server h. $\mathcal{A}_{\text{DDH}}$ sets the generator g of C to be the same as g from the DDH tuple.

Step 2: $\mathcal{A}_{\text{DDH}}$ proceeds to simulate the initial linkage tag T_0^i by first setting g^b from the DDH tuple as the client's ephemeral key Z_i. Now $\mathcal{A}_{\text{DDH}}$ generates an ephemeral secret s_k for every server $k \neq h$ as follows: $s_k = \mathcal{H}((g^b)^{y_k})$, which he can do because he possesses all private keys of dishonest servers, and $\mathcal{A}_{\text{DDH}}$ uses g^c for the ephemeral secret s_h client i shares with the honest server h. Then, $\mathcal{A}_{\text{DDH}}$ generates the initial linkage tag $T_0^i = h_i^{\prod_{k=1}^{m} s_k}$.

Step 3: For every server $k \neq h$, $\mathcal{A}_{\text{DDH}}$ processes the tag T_0^i as prescribed in the protocol. For server h, $\mathcal{A}_{\text{DDH}}$ uses g^c for s_h and $T_h = (T_{h-1})^{(-s_h)(r_h)}$, finally outputting a final linkage tag T_f^i.

Now, that $\mathcal{A}_{\text{DDH}}$ correctly simulated the view of $\mathcal{A}_{\text{ANON}}$, $\mathcal{A}_{\text{ANON}}$ outputs his guess $b'_{\text{ANON}} \in \{0, 1\}$, which $\mathcal{A}_{\text{DDH}}$ copies and outputs as his own guess $b'_{\text{DDH}} = b'_{\text{ANON}}$.

$\mathcal{A}_{\text{DDH}}$ correctly simulates the view of the challenger in the anonymity game with probability $\frac{1}{2}$ when the challenge tuple is a Diffie-Hellman tuple. Hence, $\mathcal{A}_{\text{DDH}}$'s advantage is $\frac{1}{2}$ of the advantage of $\mathcal{A}_{\text{ANON}}$. Following our assumption, $\mathcal{A}_{\text{ANON}}$ has a non-negligible advantage ϵ_{ANON} in the anonymity game and therefore $\mathcal{A}_{\text{DDH}}$'s advantage $\epsilon_{\text{DDH}} = \frac{\epsilon_{\text{ANON}}}{2}$, which is also non-negligible. Hence, a contradiction. $\square$

8.6 Forward anonymity.

Informally, an authentication protocol is *forward anonymous* if an adversary cannot break any client's anonymity even if the adversary is in possession of some (or even all) group members' private keys obtained after the protocol round completed. Recall that a protocol run is defined in terms of an authentication context. The reason that we can only ensure forward anonymity after the protocol round has ended is because an adversary who possesses the private keys of the clients can run the protocol himself using some private key x_i, successfully impersonating a client i. After a successful authentication request, the adversary would learn the final linkage tag T_f^i that would allow him to distinguish all previous authentication requests made by i as the linkage tag persists throughout the protocol round.

Definition 5. *An authentication protocol is* forward anonymous *if for any probabilistic polynomial time adversary $\mathcal{A}$, the probability $p(n)$ that $\mathcal{A}$ succeeds at the* forward anonymity *game is negligible s.t. $|p(n) - \frac{1}{2}| = negl(n)$.*

The *forward anonymity* game is played between the adversary and the challenger and is exactly as the anonymity game defined in the previous section except that in Step 7 the adversary is given the private keys (x_i, x_j) of both honest members.

Theorem 4. DAGA *offers the* forward anonymity *property.*

Proof. Following that DAGA offers anonymity, we know that an adversary $\mathcal{A}_{\text{ANON}}$ has a negligible advantage in the anonymity game. The only difference between the anonymity and forward anonymity games is the fact that $\mathcal{A}_{\text{FA}}$ receives the clients' private keys. Because the linkage tags and per-round generators are independent of the private keys, $\mathcal{A}_{\text{FA}}$ can at most do as well as $\mathcal{A}_{\text{ANON}}$ by using $\mathcal{A}_{\text{ANON}}$ as a subroutine and simply not using the private keys. Hence, $\mathcal{A}_{\text{FA}}$ advantage $\epsilon_{\text{FA}} = \epsilon_{\text{ANON}}$, which is negligible.

The only element of any authentication message M_0 that depends on the private key is the proof PK^{client}. This is because each client would use the same authentication context C, $\vec{S}$ is generated based on $z \in_R \mathbb{Z}_q^*$ and $\vec{Y} \in C$. We can easily show that a proof P_0 from M_0 could have been produced using any private key in question (x_i or x_j), and the knowledge of both keys does not aid $\mathcal{A}_{\text{FA}}$.

Recall the prover's steps to prepare PK^{client} described in Section 4.7. $P_0 = (c_s, t, c, r)$, where c_s is the random challenge t and c are the sets of commitments, and r is the set of responses as follows

1. $t = (t_{1.0}, t_{1.10}, t_{1.11}, \ldots, t_{n.0}, t_{n.10}, t_{n.11})$ for each $i \in G$, where $t_{i.0} = X_i^{w_i} g^{v_{i.0}}$,, $t_{i.10} = S_m^{w_i} g^{v_{i.1}}$, and $t_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$.

2. $c = (c_1, \ldots, c_n)$, where

$$c_i = \begin{cases} c_s - \sum_{k=1}^{n} w_k & \text{for } i = \hat{\imath} \\ w_i & otherwise \end{cases}$$

3. $r = (r_{1.0}, r_{1.1}, \ldots, r_{i.0}, r_{i.1})$, where $r_{i.k} = v_{i.k} - c_i x_{i.k}$ for all $1 \leqslant i \leqslant n$ and $k \in \{0, 1\}$, and $x_{i.0} = x_{i.1} = 0$ for all $i \neq \hat{\imath}$, $x_{\hat{\imath}.0} = x_{\hat{\imath}}$, and $x_{\hat{\imath}.1} = s$.

We observe that a private key $x_{\hat{\imath}}$ of some prover $\hat{\imath}$ is only used once to calculate $r_{\hat{\imath}.0} = v_{\hat{\imath}.0} - c_{\hat{\imath}} x_{\hat{\imath}}$ since for each $i \neq \hat{\imath}$, $r_{i.0} = v_{1.0}$. Hence, in order to decide the value of b, $\mathcal{A}_{\text{FA}}$ needs to decide that $r_{k.0}$ for some position k is equal to $r_{\hat{\imath}.0}$, where $\hat{\imath} \in \{i, j\}$, in which case the adversary would have to distinguish an element of form $v_{\hat{\imath}.0} - c_{\hat{\imath}} x_{\hat{\imath}}$ from $v_{k.0}$.

However, both $v_{\hat{\imath}.0}$ and $c_{\hat{\imath}}$ are random and unknown to the adversary since they were securely deleted. Thus, even with the knowledge of $x_{\hat{\imath}}$, all elements are indistinguishable. Alternatively, $\mathcal{A}_{\text{FA}}$ can solve each $r_{\hat{\imath}.0} = v_{\hat{\imath}.0} - c_{\hat{\imath}} x_{\hat{\imath}}$, using both x_i and x_j. In this case, however, $\mathcal{A}_{\text{FA}}$ obtains two sets of valid, and therefore indistinguishable, solutions. Hence, the knowledge of the private keys does not aid the adversary. □

8.7 Proportionality

Intuitively, the proportionality property ensures that within an authentication context C each client i can authenticate only once as a particular anonymous client and each subsequent authentication request within the same context will be recognized as coming from that client. Therefore, the verifier will be able to recognize when the same client authenticates but without knowing that client's identity. We achieve this property by assigning a unique linkage tag $T_i = h_i^{\prod_{j=1}^{m} r_j}$ to each client i in a way that ensures that the tag is always the same for each authentication request within the same context C.

The linkage tags enjoy an additional property of unlinkability between different authentication contexts. That is, the same client i receives a different and unlinkable tag T_f within some context C_2 as long as $C_1 \neq C_2$ such that $\vec{R} \in C_1 \neq \vec{R} \in C_2$. This property is important to ensure that clients remain anonymous and unlinkable even after performing authentications within different authentication contexts. It is straightforward to see that two linkage tags of client i from two different contexts are two independent elements of the underlying group $\mathcal{G}$.

Definition 6. *An authentication protocol offers the* proportionality *property if each member i receives exactly one unique final linkage tag T_f^i within the same authentication context C.*

Theorem 5. DAGA *offers the* proportionality *property.*

Proof. We will show that each client i's final linkage tag is unique, and that during each authentication within the same authentication context C, i's final linkage tag is the same.

First, however, we will consider the constraints placed on the behavior of each client i and each server j by the fact that they need to produce a proof of correctness of their work and other assumptions. By the soundness property of the underlying proof of knowledge PK^{client} and the assumption that no client knows x such that $g_j = g_i^x$ for any i, j, any client i must generate his initial linkage tag T_0 with respect to his per-round generator h_i. That is, $T_0 = h_i^s$ for some s. By the soundness property of the underlying proof of knowledge PK_1^{server}, each server j must correctly remove the ephemeral secret s_j assigned by a client and add the correct server's ephemeral secret r_j. That is, each server j calculates $T_j = T_{j-1}^{s_j^{-1} r_j}$, where s_j^{-1} is a multiplicative inverse $\pmod q$ of $s_j = H(Z^{y_j})$ and $r_j = log_g(R_j)$.

Each client's final linkage tag is unique. The fact that each client's final linkage tag is unique, that is, $T_f^i \neq T_f^j$ for any $j \neq i \in G$, follows from the basic number theoretic properties of $\mathcal{G}$. Assume the contrary, that is for i, j such that $i \neq j$ $T_f^i = T_f^j$. Then it must be so that $h_i^r = h_j^r$ where $h_i \neq h_j$, and $r = \prod_{j=1}^{m} r_j$. Because h_i is a generator of $\mathcal{G}$, then by definition every element of $\mathcal{G}$ can be expressed as h_i^x for some $x \in \{0, \ldots, q-1\}$, where q is the order of $\mathcal{G}$. Then, we have $h_j^r = h_i^{xr}$. Consequently, $h_i^r = h_i^{xr}$ only if $r = xr \pmod{q}$. Because $|h_i| = q$, then it must be so that $x = q$. Then, $r = qr \pmod{q}$ and $r = r \pmod{q}$. This contradicts the assumption that $h_i \neq h_j$.

Each client i's final linkage tag is the same for each accepted authentication request. The fact that each client i's final linkage tag is the same is straightforward. Assume two distinct authentication requests from i using the same C that result in two initial linkage tags $T_0^{'}$ and $T_0^{''}$ such that $T_0^{'} = h_i^{s'}$ and $T_0^{''} = h_i^{s''}$, where each $s' \neq s''$. The servers will collectively process both tags as follows: $T_f^{'} = (T_0^{'})^{s'^{-1}r} = (h_i^{s'})^{s'^{-1}r} = h_i^r$ and $T_f^{''} = (T_i^{''})^{s''^{-1}r} = (h_i^{s''})^{s''^{-1}r} = h_i^r$.

Therefore, each client i receives exactly one unique final linkage tag for each accepted authentication request. □

8.8 Deniability

The deniability notion that DAGA provides follows the zero-knowledge notion of deniability first formalized in the context of authentication in [27]. Informally, we can say that an authentication protocol is *deniable* if after a complete protocol run there is no proof that any client participated in the protocol given an authentication transcript of a protocol run and all public information.

The notion of deniability is closely related to anonymity, however, the subtle differences between these two properties might make a significant difference and make a protocol that provides both properties more suitable for certain situations where the mere fact that *some* client from a particular group authenticated anonymously reveals useful information. In case of anonymity, an adversary should not be able to tell *which* member authenticated while in case of deniability the adversary should not be able to tell whether *any* member authenticated based on the authentication transcripts. We can achieve anonymity by ensuring that two valid transcripts T_i and T_j of members $i, j \in G$ respectively are indistinguishable from one another. On the other hand, we achieve deniability by ensuring that a valid transcript T_i of client i is indistinguishable from a simulated transcript T_s that was computed without the help of *any* member $i \in G$.

DAGA inherently offers a weak notion of deniability in the sense that any member listed in G can plausibly deny being an active client because anyone can conscript an arbitrary group G using publicly available public keys and without any help or knowledge of the listed members. However, as pointed out above, this might not be sufficient because the fact that a valid authentication transcript exists implies that at least one of the clients in G authenticated.

DAGA achieves deniability by using an interactive rather than non-interactive zero-knowledge proof. This is because an interactive proof is not transferable and only "convinces" the party involved in the proof. In a non-interactive proof the verifier is replaced with a hash function to create an unpredictable challenge that a prover cannot anticipate in advance. Therefore, anyone at any point, even much later, can verify the non-interactive proof and be convinced (or not) that the prover knows his secret. This property, while useful, goes against the notion of deniability.

Definition 7. *An authentication protocol is* deniable *if for any client $i \in G$ there exists a simulator S_D produces a transcript TR_{sim} of a protocol run such that TR_{sim} is indistinguishable from a real transcript TR_i that resulted from i's run of the protocol.*

Theorem 6. DAGA *offers the deniability property.*

Proof. We will show that there exists a polynomial-time simulator S_D that produces a transcript TR_{S_D} that is computationally indistinguishable from a client generated transcript. We assume that S_D produces a transcript "on behalf" of some client i using an authentication context $\mathrm{C} = (G, \vec{R}, \vec{H}, p, g)$ without the knowledge of any private key x_i that corresponds to some public key $X_i \in \vec{X}$. The simulator S_D works as follows. First, S_D produces a linkage tag T_0 using the client i's prescribed per-round generator $h_i \in \vec{H}$ exactly as client i would. Since S_D has all required information (the per-round generator h_i and the product of all ephemeral secrets shared with the servers) the simulator reproduces the exact tag T_0 as follows.

1. Choose $z \in_R \mathbb{Z}_q^*$ and compute $Z = g^z$.
2. For each server j, compute $s_j = H(Y_j^z)$.
3. Compute $T_0 = h_i^{s_1 s_2 \ldots s_m}$. Then, for each $1 \leqslant j \leqslant m$ compute $S_j = g^{s_1 s_2 \ldots s_j}$ such that $S_0 = g$, $S_1 = g^{s_1}$, $\ldots$, $S_m = g^{s_1 s_2 \ldots s_m}$). Set $\vec{S} = (Z, S_0, \ldots, S_m)$.

At this point S_D has computed $\vec{S}$ and T_0. Next, S must produce a proof P_0, however, *without* the knowledge of i's private key x_i (or any other key). To do so, we leverage the fact that PK^{client_i} is zero-knowledge and therefore there exists a polynomial-time simulator S_{zk} that produces computationally indistinguishable transcripts of PK^{client_i}. S_D uses S_{zk} as a subroutine to produce P_0. S_{zk} takes as input the authentication context C and works as follows.

1. Choose $w_1, \ldots, w_n \in_R \mathbb{Z}_q^*$ for all i.
2. Choose $v_{1.0}, v_{1.1}, \ldots, v_{n.0}, v_{n.1} \in_R \mathbb{Z}_q^*$ for all i.
3. Compute commitments $t_{i.0} = X_i^{w_i} g^{v_{i.0}}$, $t_{i.10} = S_m^{w_i} g^{v_{i.1}}$, and $t_{i.11} = T_0^{w_i} h_i^{v_{i.1}}$ for each i. Set $t = (t_{1.0}, t_{1.10}, t_{1.11}, \ldots, t_{n.0}, t_{n.10}, t_{n.11})$,
4. Compute $c_s = \sum_{k=1}^{n} w_k$.
5. Set $c_i = w_i$ for each i and set $C = (c_1, \ldots, c_n)$.
6. Compute responses $r = (r_{1.0}, r_{1.1}, \ldots, r_{i.0}, r_{i.1})$ using $r_{i.k} = v_{i.k}$ for all $1 \leqslant i \leqslant n$ and $k \in \{0, 1\}$.
7. Output $P = (C, R)$.

After obtaining P, S_D sets $P_0 = P$.

It is straightforward to see that the tag produced by S_D is identical to a tag a client i would have produced, hence, it will be accepted by servers. Also, the proof P_0 is correct and can be successfully verified with respect to the simulated challenge.

$$\begin{aligned} T_{i.0} &\overset{?}{=} X_i^{c_i} g^{r_{i.0}} & T_{i.10} &\overset{?}{=} S_m^{c_i} g^{r_{i.1}} \\ X_i^{w_i} g^{v_{i.0}} &= X_i^{c_i} g^{r_{i.0}} & S_m^{w_i} g^{v_{i.1}} &= S_m^{c_i} g^{r_{i.1}} \\ X_i^{w_i} g^{v_{i.0}} &= X_i^{w_i} g^{v_{i.0}} & S_m^{w_i} g^{v_{i.1}} &= S_m^{w_i} g^{v_{i.1}} \end{aligned}$$

$$\begin{aligned} T_{i.11} &\overset{?}{=} T_0^{c_i} h_i^{r_{i.1}} & c_s &\overset{?}{=} \sum_{k=1}^{n} c_k. \\ T_0^{w_i} h_i^{v_{i.1}} &= T_0^{c_i} h_i^{r_{i.1}} & & \\ T_0^{w_i} h_i^{v_{i.1}} &= T_0^{w_i} h_i^{v_{i.1}} & \sum_{k=1}^{n} w_k &= \sum_{k=1}^{n} w_k \end{aligned}$$

Finally, S_D prepares an authentication message $M_0 = (C, S, T_0, P_0)$ and sets $TR_{S_D} = M_0$.

Now we argue that the transcript $TR_{S_D} = (C, S, T_0, P_0)$ is computationally indistinguishable from a client generated one.

- C, the authentication context, has an identical distribution,
- $\vec{S}$, client's ephemeral key and commitments, has an identical distribution,
- T_0, the linkage tag, has an identical distribution,
- P_0 is produced by a simulator S_{zk} that produces proofs that are computationally indistinguishable from a client generated one as the underlying proof of knowledge is honest-verifier zero-knowledge.

□

8.9 Forward Deniability

One of the goals of DAGA is to retain anonymity even under the exposure of the clients' long term private keys. This raises an interesting idea to apply the same requirement of forward security to the deniability property. That is, we would like to ensure that a pair of transcripts TR_{sim} and TR_{real} generated using an authentication context C, remains indistinguishable even given the additional knowledge of the compromised private keys. We call this notion of deniability *forward deniability.* Intuitively, forward deniability should hold given that deniability holds as we were able to show that we can generate an indistinguishable transcript T_{sim} *without* the knowledge of any private key. The proof of forward deniability follows similarly to the proof of forward anonymity where we argue that the additional knowledge of the private key does not aide the adversary in distinguishing the transcripts.

Definition 8. *An authentication protocol is* forward deniable *if for any client i a simulated transcript TR_{S_D} remains (computationally) indistinguishable from a real transcript TR_i that resulted from i's run of that protocol even given a private key x_j of every client $j \in G$.*

Theorem 7. DAGA *offers the* forward deniability *property.*

Proof. Assume we have an authentication context C and two transcripts TR_i and TR_{S_D} where each transcript consists of an authentication message $\mathrm{M}_0^i = C, \vec{S}, T_0, P_0$ and $\mathrm{M}_0^{S_D} = C', \vec{S}', T_0', P_0'$ respectively created using C. We previously argued, in the proof of deniability, that these two transcripts are (computationally) indistinguishable. Now we wish to revisit this claim and verify if the knowledge of all private keys x_i aids to distinguish the two transcripts.

- $C = C'$ are identical and therefore have an identical distribution.
- $\vec{S}$ and $\vec{S}'$ are randomly generated based on $z, z' \in_R \mathbb{Z}_q^*$ and have an identical distribution, and do not depend on a client's private key.
- T_0 and T_0' have an identical distribution and also do not depend on a client's private key.
- P_0 is produced by a simulator S_{zk} without the knowledge of any private key and P_0' is a client generated transcript using his private key x_i.

Therefore, the only element of the transcript that could be affected by the knowledge of the private keys is the proof of knowledge P_0 as P_0 is created using x_i and P_0' without x_i. Therefore, we observe the following differences in the set of responses of r and r': $r_{i.0} = v_{i.0} - c_i x_i$ and $r'_{i.0} = v'_{i.0}$. In order to distinguish between P_0 and P_0', it must be possible to distinguish between $r_{i.0}$ and $r'_{i.0}$. However, both $v_{i.0}$ and c_i are random and unknown and therefore even with the knowledge of x_i $r_{i.0}$ and $r'_{i.0}$ are indistinguishable. □

9 Related Work

There are several approach to realizing anonymous authentication, a broad class of schemes offering varying sets of properties. Some of them focus on providing properties, such as unlinkability or anonymity revocation by a third party, that are in contradiction to the properties DAGA is design to achieve.

Group and ring signatures Group signatures [13, 20] allow a member to anonymously sign a message on behalf of a pre-defined group. However, user's anonymity is revocable by a group manager. Ring signatures [1, 7, 25, 50, 51] offer greater flexibility by allowing ad-hoc group creation thereby supporting a weaker notion of deniability. Linkable ring signatures [44, 45] and short linkable signatures [3, 59] further improve upon ring signatures by adding linkability. The schemes of [1, 50] support heterogeneous keys.

E-cash E-cash schemes [6, 9, 16, 18] are designed with anonymity (also referred to as untraceability) in mind and often achieve deniability as well. However, normally these schemes prevent double-spending as using a coin twice reveals the owner's identity, rendering the schemes useable for one-time authentication only.

Anonymous credentials The credential system proposed in [10] allows users to obtain credentials from organizations and later demonstrate their possession in an anonymous and unlinkable way as many times as desired. The lack of linkability was later addressed by one-time credentials [43] in form of coins that if spent twice would reveal user's identity. The schemes proposed in [8, 41, 48, 58] bridge this gap and offer credentials that a user can show up to k times, offering limited linkability which in case of [8] applies to certain periods of time.

Other schemes Deniable authentication [27] defines the idea of deniability in the context of authentication. Their notion of deniability assures that the protocol does not leave any paper trail, however, the scheme is not anonymous. Deniable Ring Authentication [47] combines deniable authentication with ring signatures [50]. While it offers protection against compromised private keys, it still lacks proportionality. [56, 57] makes the protocol of [27] non-interactive. [40] proposes another protocol to achieve deniable ring signature, however, the deniability property is viewed as non-frameability of honest client.

10 Conclusions and Future Work

DAGA is a new anonymous group authentication protocol that offers a unique set of properties: anonymity, deniability, and proportionality that persists even in a case of private key exposure. Our initial evaluation suggests that DAGA compares reasonably well to LRS and non-anonymous authentication given the functionality gain, however, the performance concerns remain.

Future work includes extending DAGA to handle heterogeneous keys, use batching techniques for proofs of knowledge [37] and elliptic curves to improve performance, and extend the current implementation to a fully distributed system.

References

[1] M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In *'02 ASIACRYPT*, 2002.

[2] R. Aditya, B. Lee, C. Boyd, and E. Dawson. An efficient mixnet-based voting scheme providing receipt-freeness. In *Trust and Privacy in Digital Business*. 2004.

[3] M. H. Au, S. S. M. Chow, W. Susilo, and P. Tsang. Short linkable ring signatures revisited. In *'06 EUROPKI*, 2006.

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st CCS*, pages 62–73, 1993.

[5] D. Boneh. The decision diffie-hellman problem. In *Algorithmic number theory*. Springer, 1998.

[6] S. Brands. Untraceable off-line cash in wallet with observers. In *'94 CRYPTO*, 1994.

[7] E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *'02 CRYPTO*, 2002.

[8] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *'06 CCS*, 2006.

[9] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *'05 EUROCRYPT*, 2005.

[10] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *'01 EUROCRYPT*, 2001.

[11] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76. Springer, 2002.

[12] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *'03 CRYPTO*. 2003.

[13] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *'97 CRYPTO*, 1997.

[14] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Dept. of Computer Science, ETH Zurich, March 1997.

[15] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[16] D. Chaum. Blind signatures for untraceable payments. In *Crypto*, 1982.

[17] D. Chaum. The Dining Cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, pages 65–75, Jan. 1988.

[18] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *'88 CRYPTO*, 1990.

[19] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *'92 CRYPTO*, 1992.

[20] D. Chaum and E. Van Heyst. Group signatures. In *'91 EUROCRYPT*, 1991.

[21] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *'94 CRYPTO*, 1994.

[22] I. Damgard. Lecture notes on cryptographic protocol theory: On sigma protocols, 2004.

[23] A. Davidson. Introducing strongbox. *The New Yorker*, May 2013.

[24] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *12th USENIX Security Symposium*, Aug. 2004.

[25] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In *'04 EUROCRYPT*, 2004.

[26] J. R. Douceur. The Sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, 2002.

[27] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *'98 STOC*, 1998.

[28] N. Fazio and A. Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*, 2002.

[29] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *'87 CRYPTO*, 1987.

[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *'96 EUROCRYPT*, 1996.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. *Information and Computation*, 2001.

[32] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 1989.

[33] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 1988.

[34] D. Greene. EFF files 22 firsthand accounts of how NSA surveillance chilled the right to association, Nov. 2013. `https://www.eff.org/press/releases/eff-files-22-firsthand-accounts-how-nsa-surveillance-chilled-right-association`.

[35] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *'07 SOSP*, 2007.

[36] R. Henry and I. Goldberg. Batch proofs of partial knowledge. *University of Waterloo, Technical report CACR*, 2012.

[37] R. Henry and I. Goldberg. Thinking inside the blac box: smarter protocols for faster anonymous blacklisting. In *'13 WPES*, 2013.

[38] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *'00 EUROCRYPT*, 2000.

[39] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *'05 WPES*, 2005.

[40] Y. Komano, K. Ohta, A. Shimbo, and S. Kawamura. Toward the fair anonymous signatures: Deniable ring signatures. In *'06 CT-RSA*, 2006.

[41] M. Layouni and H. Vangheluwe. Anonymous k-show credentials. In *'07 EuroPKI*, 2007.

[42] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *'03 ICISC*. 2004.

[43] J. K. Liu, P. P. Tsang, and D. S. Wong. Recoverable and untraceable e-cash. In *'05 EuroPKI*, 2005.

[44] J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *ACISP*, 2004.

[45] J. K. Liu and D. S. Wong. Linkable ring signatures: Security models and new schemes. In *'05 ICCSA*, 2005.

[46] J. Maheswaran, D. I. Wolinsky, and B. Ford. Crypto-book: An architecture for privacy preserving online identities. In *Hot Topics in Networks*, 2013.

[47] M. Naor. Deniable ring authentication. In *'02 CRYPTO*, 2002.

[48] L. Nguyen and R. Safavi-naini. Dynamic k-times anonymous authentication. In *'05 ACNS*, 2005.

[49] K. Peng, C. Boyd, and E. Dawson. Batch zero-knowledge proof and verification and its applications. In *'07 TISSEC*, 2007.

[50] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT*, 2001.

[51] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret: Theory and applications of ring signatures. In *Theoretical Computer Science*, 2006.

[52] C. Savage. Court rejects appeal bid by writer in leak case. *New York Times*, Oct. 2013.

[53] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[54] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding carnivores: File sharing with strong anonymity. In *11th ACM SIGOPS European Workshop*, Sept. 2004.

[55] D. R. Stinson. *Cryptography: Theory and Practice.* 2005.

[56] W. Susilo and Y. Mu. Deniable ring authentication revisited. In *'04 ACNS*, 2004.

[57] W. Susilo and Y. Mu. Non-interactive deniable ring authentication. In *'04 ICISC*, 2004.

[58] I. Teranishi, J. Furukawa, and K. Sako. K-times anonymous authentication. In *'04 ASIACRYPT*, 2004.

[59] P. P. Tsang and V. K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In *'05 ISPEC*, 2005.

[60] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *EuroSec*, 2012.

[61] D. I. Wolinsky, H. Corrigan-Gibbs, A. Johnson, and B. Ford. Dissent in numbers: Making strong anonymity scale. In *10th OSDI*, Oct. 2012.