Vote to Link: Recovering from Misbehaving Anonymous Users*

Wouter Lueks
Institute for Computing and Information Sciences (iCIS), Radboud University
Nijmegen, The Netherlands
lueks@cs.ru.nl

Maarten H. Everts
TNO Netherlands
Organisation for Applied Scientific Research
The Hague, The Netherlands
maarten.everts@tno.nl

Jaap-Henk Hoepman
Institute for Computing and Information Sciences (iCIS), Radboud University
Nijmegen, The Netherlands
jhh@cs.ru.nl

ABSTRACT

Service providers are often reluctant to support anonymous access, because this makes it hard to deal with misbehaving users. Anonymous blacklisting and reputation systems can help prevent misbehaving users from causing more damage. However, by the time the user is blocked or has lost reputation, most of the damage has already been done. To help the service provider to *recover* from abuse by malicious anonymous users, we propose the *vote-to-link* system.

In the vote-to-link system, moderators (rather than a single trusted third party) can cast votes on a user's action if they deem it to be bad. After enough moderators have voted on the action, the service provider can use these votes to link all the actions by the same user within a limited time frame and thus recover from these actions. All the user's actions in other time frames, however, remain unlinkable.

To protect the voting moderators from retaliation, we also propose a (less efficient) variant that allows moderators to vote anonymously. We implemented and evaluated both variants to show that they are practical. In particular, we believe this system is suitable to combat malicious Wikipedia editing.

CCS Concepts

•**Security and privacy** → **Privacy-preserving protocols;** *Pseudonymity, anonymity and untraceability;* Privacy protections;

*This research is supported by the research program Sentinels (www.sentinels.nl) as project 'Revocable Privacy' (10532). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs. In addition, this research is in part supported by the THeCS project of the Dutch National Program COMMIT. This research is conducted within the Privacy and Identity Lab (www.pilab.nl).

WPES'16, October 24, 2016, Vienna, Austria

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4569-9/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2994620.2994634

Keywords

Privacy; privacy-enhancing technologies; cryptography

1. INTRODUCTION

Many websites either completely disallow anonymous access or block anonymous users from making changes—for example, Wikipedia does not allow edits by users using the Tor network.[1] While there are benefits to allowing anonymous access, websites apparently feel that the cost in terms of abuse outweighs these benefits.

In this paper we present the vote-to-link system. It shifts the balance between the costs and benefits of anonymous access. The system allows fully anonymous actions, while, at the same time, enabling linking of a malicious user's actions to make it much easier to recover from abuse. The vote-to-link system is constructed in such a way that, after at least k moderators mark an action as bad, the system can identify all other actions by the same user within a limited time frame (which we call an epoch). Without the cooperation of at least k moderators, users' actions remain anonymous and unlinked.

To further reduce the impact of linking a user's actions—the judgement might have been erroneous, or a user's earlier actions might have been benign—the linking is limited to a predefined time window. Note that even if many moderators are malicious, the worst they can do is link a user's actions within epochs, never across epochs, nor can they directly deanonymize her.

A leading example: Wikipedia.

To see why linking an abusive user's edits within a limited time frame is useful, we return to the example of editing Wikipedia. We wish to emphasize that it makes sense for editors to be anonymous—for example when they edit controversial articles. However, this anonymity also enables abuse by anonymous editors that change pages incorrectly.

To reduce abuse, Wikipedia could deter users from being abusive. For example, Wikipedia could block abusive users from further accessing the system (for example, BLAC [28]

[1]https://en.wikipedia.org/wiki/Wikipedia:Advice_to_users_using_Tor accessed July 4, 2016. Naturally, Wikipedia also does not allow registering sock puppet accounts via Tor (that would defeat the purpose of blocking Tor in the first place). While registering accounts outside of Tor is possible, even those accounts are normally not exempt from the Tor block (see https://en.wikipedia.org/wiki/Wikipedia:IPBE, accessed July 4, 2016).

allows a service provider to block anonymous users without identifying them). Alternatively, it could use a reputation system that reduces an abusive user's reputation so that making edits in the future becomes harder. Or, it could even fully deanonymize abusive users in a name-and-shame fashion. However, when such measures fail to actually deter a user from being malicious, she can still do a lot of damage. Our vote-to-link system helps mitigate this damage.

A combination of four factors ensures that undeterred users can still do a lot of damage. One, actual people are needed to detect abuse, so, time passes before the first violation is detected and the deterrent is effected. Two, this delay is exacerbated if only a small percentage of actions is bad, requiring moderators to examine many edits before finding one abusive edit. Three, until the first violation is detected, the abusive user is free to continue acting anonymously (and, these actions are, by nature of the anonymity, unlinkable). Four, an abusive user can perform many actions, as rate limiting would also adversely affect honest users.[2]

The vote-to-link system reduces the damage by invalidating the second and third factor: after sufficiently many moderators agree that a particular edit is bad, all that user's edits within a time frame (maybe 24 hours is appropriate here) become linkable. Hence, the moderation effort can be focussed on those edits that are already highly suspect.

While the actions of abusive editors can be linked (after enough moderators vote to do so) to make it easy to find all other abusive actions by that user, the anonymity of honest users is guaranteed: actions are unlinkable if there are too few votes. Moreover, the identity of an editor is never revealed. To the best of our knowledge this is a novel approach to combat malicious edits.

Our contributions.
Our first contribution is the idea of a vote-to-link scheme. At a high-level it works as follows. For every transaction a user wants to perform at a service provider (for example, editing a Wikipedia page), she must create a linking token. The linking token is bound to her cryptographic identity. Given this token, it is possible to identify all other transactions by that user within the current epoch. A group manager provides each user with at most one anonymous credential certifying the user's cryptographic identity.

To protect the linking token, the user encrypts it to the moderators' public key using a threshold encryption scheme. Finally, to perform the transaction, she sends the encrypted linking token and a zero-knowledge proof of correct encryption (i.e., she anonymously proves that the encrypted linking token belongs to the identity in her anonymous credential) to the service provider. Moderators can vote to link the user's transaction by partially decrypting the linking token (they can do so without communicating with other moderators). If, at some point in time, enough moderators have voted, the service provider can decrypt the linking token (using the partial decryptions provided by the moderators) and hence link all the user's actions within that epoch. The anonymity of the user is guaranteed by distributing the voting power invested in the moderators. The service provider itself has a purely facilitating role; even if it is malicious, it cannot link a user's actions. We first introduce the high level structure of our approach in Section 2, then describe preliminaries in Section 3, and, finally, describe our vote-to-link scheme in full in Section 4.

Our second contribution is to allow moderators to vote anonymously (normally, the structure of the threshold encryption scheme identifies voting moderators). While procedural measures can protect the votes to some extent, we prefer to not rely on those to protect the identity of the voting moderators. Hence, we have created a variant of our vote-to-link scheme that allows moderators to vote anonymously at the cost of reduced efficiency. We present this scheme in Section 5.

Our third and final contribution is an implementation of our protocols, showing the practicality of our approach. We present these results, and other practical considerations for deploying a vote-to-link system, in Section 6.

While the vote-to-link scheme may seem similar to group signatures with a distributed group manager—in fact, we borrow ideas from group signatures—we wish to emphasize that there are significant differences. First, the goal of our scheme is to make a user's actions linkable within an epoch, rather than identify the signer of a single message. Second, while distributed tracing managers allow a similar voting process, they do not allow anonymous voting. We cover related work more extensively in Section 7, before concluding this paper in Section 8.

2. SYSTEM DESIGN AND ASSUMPTIONS

In this section we describe the architecture of our vote-to-link system, and its assumptions.

2.1 Architecture

The system consists of one group manager and several users, service providers, and moderators, each with their own role in the system. For clarity, we focus on a system with only one service provider. In practice, it might be desirable to have multiple service providers. Our solution is easily extended in this direction.

Group Manager (GM) The group manager (GM) sets up the system. It ensures that a user in the system has at most one identity (for example, by requiring that a user has access to a scarce resource, see below). This ensures that the user cannot prevent linking of her actions by creating new identities. To allow the user to use this identity anonymously, the GM provides the user with an anonymous credential.

User (U) Users access services offered by the service provider. To enable access they need an anonymous credential from the GM. The users' actions at a service provider can be linked only when enough moderators vote to enable this; normally, users' actions are anonymous. Users are not assumed to be always online.

Service Provider (SP) The service provider allows users to anonymously perform transactions (provided they supply the correct information). As part of performing

[2]For example, the Wikipedia editor Giraffedata regularly makes about 80 edits within one hour (see https://backchannel.com/meet-the-ultimate-wikignome-10508842caad, last accessed July 5, 2016), while the most prolific Wikipedia editor Ser Amantio di Nicolao averages 60 edits an hour assuming 8 hours of editing every day of the year (see https://en.wikipedia.org/wiki/Wikipedia:List_of_Wikipedians_by_number_of_edits, last accessed July 5, 2015), while his/her burst rate is most likely much higher.

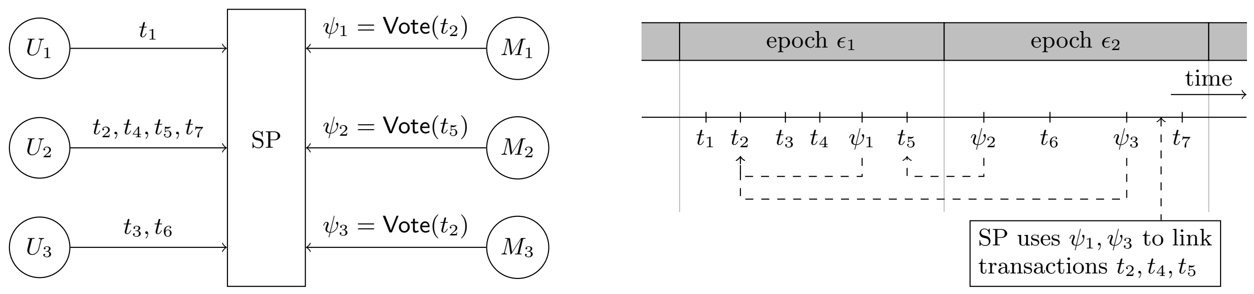
U_1
U_2
U_3
t_1
t_2, t_4, t_5, t_7
t_3, t_6
SP
$\psi_1 = \mathsf{Vote}(t_2)$
$\psi_2 = \mathsf{Vote}(t_5)$
$\psi_3 = \mathsf{Vote}(t_2)$
M_1
M_2
M_3
epoch ϵ_1
epoch ϵ_2
time
t_1 t_2 t_3 t_4 ψ_1 t_5 ψ_2 t_6 ψ_3 t_7
SP uses ψ_1, ψ_3 to link transactions t_2, t_4, t_5

Figure 1: An example of the vote-to-link system with 3 users (U_1, U_2, U_3), 3 moderators (M_1, M_2, M_3) and a threshold $k = 2$. The left side of the picture shows how the users and the moderators interact with the service provider: users send transactions ($t_1, \dots, t_7$) to the service provider. The SP publishes these transactions in a public record after checking their validity. The moderators monitor these transactions, and, whenever they detect a malicious one, send a vote to link to the service provider (ψ_1, ψ_2, ψ_3). The right side of the picture shows a timeline of events at the service provider. After the SP receives $k = 2$ votes on transaction t_2, it can link it to the other transactions t_4, t_5 made by the same user in that epoch (note that t_7 is not linked because it was performed in another epoch).

the transaction, the user proves to the SP that she has an anonymous credential. The service provider determines the length of an epoch.

Moderator (M) The moderators monitor the users' activities. They can vote to link the actions of a user. When sufficient moderators have voted to do so, the user's actions within this epoch become linkable. The group of moderators can be specific to each service provider. Moderators are also not assumed to be always online.

The groups of users and moderators do not have to be mutually exclusive. In fact, moderators may also be users, or even, all users may be moderators as well.

Because the users' credentials are anonymous, users can use them to prove that they correctly constructed the encrypted linking tokens, without otherwise revealing anything about their identity, nor becoming linkable because of using the credential. To maintain anonymity, users (and moderators) communicate with the SP via an anonymous channel.

Figure 1 gives an overview of an example vote-to-link system. The figure shows how a user's actions become linkable as a result of the moderators' votes. For clarity, we omitted the group manager.

Only the service provider and the group manager are assumed to be always online (but note that the only consequence of a group manager being offline is that new users cannot join). The moderators can cast votes completely noninteractively. This ensures that the system can still function even when the vast majority of the moderators are offline.

2.2 Threat model and security goals

Our system has two security goals (we formalize them in Game 2 and Game 3 in Appendix B respectively).

1. **User anonymity.** As long as an adversary controls at most $k - 1$ moderators, it cannot link an honest member's transactions with probability non-negligibly better than random guessing.
2. **Moderator anonymity.** Users that collude with up to $n-2$ moderators cannot determine the identity of a non-colluding voting moderator with probability non-negligibly better than random guessing.

The user anonymity goal strictly implies that users cannot be identified either. Furthermore, for user anonymity, we make no security assumptions about the users and the service provider. In fact, even if they are completely malicious, the anonymity of the users remains fully protected. Clearly, we can allow at most $k - 1$ moderators to be malicious. For moderator anonymity, we assume that the user performing the transaction and the SP are not both malicious.

About Sybil attacks.

To ensure that all user's actions can be linked after voting, we require, like many other anonymous systems with an accountability feature, that users have only one identity. Our system by itself is not robust against Sybil attacks [12], and protecting against them is outside the scope of this paper.

We do acknowledge that protecting against Sybil attacks is difficult. However, some protection can be achieved when the group manager issues credentials based on some scarce resource (for example, IP address, phone number or national electronic ID card). See Henry and Goldberg for an overview [18].

3. PRELIMINARIES

In this section we cover some of the (cryptographic) preliminaries necessary to describe our schemes.

3.1 Notation and cryptography

We first introduce some notation. We write $\mathbb{Z}_q$, with q a prime, to denote the field of integers modulo q. We write $r \in_R A$ to indicate that r is chosen uniformly at random from the (finite) set A. Finally, we write $[n]$ to denote the set $\{1, \dots, n\}$.

Our protocols rely on Shamir's secret sharing scheme [26], so we often use Lagrange polynomials and coefficients. They are defined as follows.

Definition 1 (Lagrange interpolation). *Let $\mathcal{I} \subset \mathbb{Z}_q$ be a set of indices of cardinality k, then the corresponding* Lagrange polynomials *are given by $\lambda_i^{\mathcal{I}}(t) = \prod_{j \in I, j \neq i} \frac{t-j}{i-j}$. Any polynomial $f(t) \in \mathbb{Z}_q[t]$ of degree $k-1$ can then be interpolated by k points $(i, f(i))$, $i \in \mathcal{I}$, as $f(t) = \sum_{i \in \mathcal{I}} \lambda_i^{\mathcal{I}}(t) f(i)$. We define the* Lagrange coefficients *as $\lambda_i^{\mathcal{I}} = \lambda_i^{\mathcal{I}}(0)$. For a* *polynomial f as before, we have $f(0) = \sum_{i\in\mathcal{I}} \lambda_i^{\mathcal{I}} f(i)$.*

We use a simplified version the notation of Camenisch and Stadler [8] to denote zero-knowledge proofs. More specifically, we write

$$SPK\{(x,y) : A = g^x \wedge B = g^x h^y\}(m)$$

to denote the non-interactive proof of knowledge of the values x and y, such that $A = g^x$ and $B = g^x h^y$ over the message m obtained using the Fiat-Shamir heuristic [13]. The verifier only learns the validity of this proof and the values of A, B, g, h, but not of the values before the colon, x and y, of which knowledge is being proven.

3.1.1 Bilinear maps

Consider a pair of cyclic groups $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order q, with generators g, h respectively. We call such a pair a bilinear group pair when there exists a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ mapping these groups into a group $\mathbb{G}_T$ such that: (1) the map is bilinear, i.e., for all $a, b \in \mathbb{Z}_q$ we have $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$; (2) the map is non-degenerate, i.e., $\hat{e}(g, h)$ is a generator of $\mathbb{G}_T$; and (3) the map is efficiently computable. Denote by $H : \{0,1\}^* \to \mathbb{G}_1$ a cryptographic hash function mapping strings to $\mathbb{G}_1$.

3.1.2 Cryptographic assumptions

The security of our schemes relies on a number of cryptographic assumptions which we will briefly cover here.

Definition 2 (DDH problem). *The Decisional Diffie-Hellman (DDH) problem in a cyclic group $\mathbb{G}$ is defined as follows. On input of a tuple $(g, A = g^a, B = g^b, C = g^c) \in \mathbb{G}^4$ output 'yes' if $c = ab$ and 'no' otherwise.*

The following definition is the generalization of the original Decisional Bilinear Diffie-Hellman problem [19] to the asymmetric bilinear map setting.

Definition 3 (DBDH problem). *The Decisional Bilinear Diffie-Hellman (DBDH) problem in a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ with generators $g_1 \in \mathbb{G}_1, h_1 \in \mathbb{G}_2, h_2 \in \mathbb{G}_2, g_T \in \mathbb{G}_T$ is defined as follows. Given $g_1^a, h_1^b, h_2^c, g_T^z$ output 'yes' if $z = abc$ and 'no' otherwise.*

3.1.3 Anonymous credentials

Every user in the system is given an anonymous credential to prove that they have joined the system and to assign a unique identity to each user.

There are many methods to construct anonymous credentials, one could use a system like U-Prove [5] or Idemix [7], or directly construct them using an appropriate signature scheme like BBS+ signatures [3]. Our schemes are agnostic to the specific choice of credential scheme, so we just write $C(x)$ to denote a credential over the secret key x.

Credentials are issued by an issuer, in our case the group manager, to a user. The user can then show a credential to a service provider, who can determine that the user indeed has a credential issued by an issuer. Our scheme requires the following properties:

Unforgeability It is not possible for any party in the system to forge a credential $C(x)$ over a new private key x without the help of the issuer.

Unlinkability The showing of a credential gives no information about the owner to the service provider. In particular, it is not possible for an adversary to distinguish between two users (of its choosing) when it is shown a credential of one of them.

Zero-knowledge proofs The showing of a credential $C(x)$ can be combined with a zero-knowledge proof that proves a statement about x.

3.2 CCA secure threshold encryption

To encrypt the linking tokens, we need need an encryption scheme that is both verifiable—to ensure that the encrypted linking tokens are well-formed—and threshold decryptable—to allow the moderators to create decryption shares when necessary. Finally, in our user anonymity game (see Game 2) the adversary can request decryption shares of any ciphertext except the challenge ciphertext. To ensure that it cannot learn anything about the challenge ciphertext, we require CCA security.

In this section we show a variant of the Shoup and Gennaro's TDH2 threshold encryption scheme [27] that can verifiably encrypt group elements, rather than bit strings. The core of the original scheme is similar to hashed ElGamal in a cyclic group $\mathbb{G}$ generated by g where the ciphertext of a message m for a public key $w = g^\kappa$ is of the form $(c, u) = (m \oplus H(w^r), g^r)$. In our TDH2' variant we replace c with $m \cdot w^r$.

Scheme 1 (TDH2'). *The setting of this scheme is in a cyclic group $\mathbb{G}$ of prime order q. Let $H' : \{0,1\}^* \to \mathbb{Z}_q$ be a cryptographic hash function. The encryption function takes a label. This label is not encrypted, but it is bound to the ciphertext. The decryptors use this label to decide whether they want to decrypt the ciphertext. The scheme is given by the following algorithms.*

- *TDH.KeyGen$(n, k, \mathbb{G}, g, q)$ The TDH.KeyGen algorithm is run by a group of n decryptors. On input of the number of decryptors n, the threshold k and a cyclic group $\mathbb{G}$ with generator g and prime order q the decryptors proceed as follows. They jointly run a verifiable secret-sharing protocol (for example using Pedersen's verifiable secret-sharing algorithm [25]) to generate a secret sharing polynomial f of degree $k-1$ such that decryptor i holds exactly one share, $\kappa_i = f(i)$, of the private key $\kappa = f(0)$ and such that the public key $w = g^\kappa$ and verification keys $w_i = g^{\kappa_i}$ are publicly known.[3] Moderator i stores the decryption key $\delta_i = (i, \kappa_i)$. Next, the moderators jointly generate a random generator $\bar{g} \in_R \mathbb{G}$ and publish the public parameters $(\mathbb{G}, g, \bar{g}, q)$, the ciphertext space $\mathcal{C} = \mathbb{G} \times \{0,1\}^* \times \mathbb{G}^2 \times \mathbb{Z}_q^2$, the public key w, and the verification keys $w_1, \ldots, w_n$.*

- *TDH.Enc(w, m, L) On input of a public key w, a message m and a label L, the TDH.Enc algorithm creates a ciphertext as follows. First, it generates $r, s \in_R \mathbb{Z}_q$ and sets*

$$c = m \cdot w^r, \quad u = g^r, \quad \hat{u} = g^s, \quad v = \bar{g}^r, \quad \hat{v} = \bar{g}^s.$$

[3]In this paper, we only use the verification keys to check the validity of the user-generated TDH2' key in Section 5. We omit the verification of decryption shares as our schemes do not require it.

Then, it calculates $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$ and sets $d = s + re$. The ciphertext is $\psi = (c, L, u, v, e, d)$.

- $\mathsf{TDH.Dec}(\psi, \delta_i)$ *On input of a ciphertext $\psi = (c, L, u, v, e, d)$ and a decryption key $\delta_i = (i, \kappa_i)$ the decryptor first verifies that the ciphertext is well-formed. To this end it calculates:*

$$\hat{u} = g^d u^{-e} \qquad \hat{v} = \bar{g}^d v^{-e}$$

and checks that $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$. If this check fails, it returns $\psi_i = (i, \perp)$. Otherwise, it returns $\psi_i = (i, u_i) = (i, u^{\kappa_i})$.

- $\mathsf{TDH.Combine}(\psi, \{\psi_{i_1}, \ldots, \psi_{i_k}\})$ *Given a ciphertext ψ and a set of of k shares the combine algorithm proceeds as follows. It tests the validity of its inputs: letting $\mathcal{I} = \{i_1, \ldots, i_k\}$, the algorithm checks that $|\mathcal{I}| = k$; and, it checks that ψ is well-formed. It returns $\perp$ if either test fails. Otherwise, every decryption share ψ_i is of the form (i, u_i) for $i \in \mathcal{I}$, so*

$$m = c \prod_{i \in \mathcal{I}} u_i^{-\lambda_i^{\mathcal{I}}}$$

is the plaintext. Return m.

It is easy to verify that the scheme is correct, i.e., for every message m and ciphertext $\psi = \mathsf{TDH.Enc}(w, m, L)$ and for every set of k decryption shares $\psi_{i_1}, \ldots, \psi_{i_k}$ we have that $\mathsf{TDH.Combine}(\psi, \{\psi_{i_1}, \ldots, \psi_{i_k}\}, VK) = m$. In Appendix A we prove the following theorem.

THEOREM 1. *The TDH2' scheme is CCA secure in the random oracle model for H', assuming that the DDH assumption holds in $\mathbb{G}$.*

3.3 ElGamal encryption

In our fully anonymous scheme, we use use ElGamal encryption to encrypt the moderators' decryption shares.

SCHEME 2 (ELGAMAL ENCRYPTION). *The ElGamal encryption scheme in a cyclic group $\mathbb{G}$ with generator g of prime order q is defined as follows.*

- KeyGen $(\mathbb{G}, g, q)$ *Given a cyclic group $\mathbb{G}$ of order q generated by g, choose a private key $x \in_R \mathbb{Z}_q$ and set the corresponding public key $h = g^x$. Return (x, h).*

- Enc (m, h) *To encrypt a message $m \in \mathbb{G}$ against a public key h pick a randomizer $r \in_R \mathbb{Z}_q$ and create the ciphertext $c = (c_1, c_2) = (m \cdot h^r, g^r)$.*

- Dec (c, x) *To decrypt a ciphertext $c = (c_1, c_2)$ using the private key x compute $m = c_1/c_2^x$.*

Due to ElGamal's simple structure it is possible to randomize a public key and to transform ciphertexts for this randomized public key into ciphertexts for the original public key. We use this in our fully anonymous protocol to hide the identity of the moderators.

- $\mathsf{Randomize}(h, \alpha)$ To randomize a public key $h \in \mathbb{G}$ using a randomizer $\alpha \in \mathbb{Z}_q$ compute $\bar{h} = h \cdot g^\alpha$.

- $\mathsf{Derandomize}(\bar{c}, \alpha)$ To derandomize a ciphertext $\bar{c} = (\bar{c}_1, \bar{c}_2)$ that is encrypted against a randomized public key $\bar{h} = \mathsf{Randomize}(h, \alpha)$ with randomizer α, calculate $c = (\bar{c}_1/\bar{c}_2^\alpha, \bar{c}_2)$.

It is easy to check that for $\bar{h} = \mathsf{Randomize}(h, \alpha)$ and $\bar{c} = \mathsf{Enc}(m, \bar{h})$ we have $c = \mathsf{Derandomize}(\bar{c}, \alpha) = \mathsf{Enc}(m, h)$.

4. A VOTE-TO-LINK SCHEME

In this section we introduce our basic vote-to-link scheme. We first present the idea, then given the full scheme, and, finally, prove user anonymity.

4.1 The idea

We follow an idea by Nakanishi and Funabiki [24] to create tokens that can be used to link a user's actions. As a setting we use a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ of prime order q with $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ the corresponding bilinear map and $g \in \mathbb{G}_1, h \in \mathbb{G}_2$ generators. Let x be the user's secret key and $H : \{0,1\}^* \to \mathbb{G}_1$ a cryptographic hash function. We use this hash function to transform a description of the epoch ϵ (for example, ϵ could be the current date to get 24 hour epochs) into a generator $g_\epsilon = H(\epsilon)$.

The user's linking token for epoch ϵ is given by $r = g_\epsilon^x$. With each transaction, the user publishes auxiliary values $t_1 = h^z$ and $t_2 = \hat{e}(g_\epsilon, h^z)^x$ for a random $z \in_R \mathbb{Z}_q$. The user's linking token r—which is normally unknown—can be used to link transactions by this user. To check whether a transaction with auxiliary values t'_1, t'_2 was made by a user with linking token r, the SP can simply check whether $\hat{e}(r, t'_1) = t'_2$.

So, given the linking token, the SP can link the actions of the user within an epoch. To enable linking when the moderators vote to do so, the user creates an encrypted linking token $T = \mathsf{TDH.Enc}(\Delta, r, \tau)$ by encrypting the token r labeled with the transaction τ against the moderators' public key Δ using the k-out-of-n threshold encryption scheme. To ensure that the user cannot cheat, she has to prove in zero-knowledge that T, t_1, and t_2 are correct and correspond to her anonymous credential $C(x)$ certifying her cryptographic identity or secret key x.

If a moderator later feels that a transaction is inappropriate, the moderator can cast a vote to link the user's actions by partially decrypting the encrypted linking token T. When a sufficient number of decryption shares are published, the vote passes, and the SP can use the decryption shares to recover the linking token r, allowing the user's transactions to be linked.

4.2 Our scheme

We are now ready to formalize our vote-to-link idea discussed above.

SCHEME 3 (VOTE-TO-LINK). *Our* vote-to-link *scheme with threshold k and n moderators is given by the following algorithms.*

- $\mathsf{Setup}(1^\ell, n, k)$ *To setup the system, the group manager first runs $\mathsf{SetupGM}(1^\ell)$ and then the moderators run $\mathsf{SetupModerators}(1^\ell, n, k)$.*

- $\mathsf{SetupGM}(1^\ell)$ *The $\mathsf{SetupGM}$ algorithm is run by the group manager responsible for adding users. The group manager first sets up an anonymous credential scheme with security level ℓ in which the GM is an issuer. Next, it generates a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order q such that q is ℓ-bits, with generators g, h respectively, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the DDH problem is hard in $\mathbb{G}_1$. It publishes the public information for the anonymous credential scheme, and a description of the groups, generators and bilinear map.*

- *SetupModerators*$(1^\ell, n, k)$ *The moderators jointly run the* *TDH.KeyGen*$(n, k, \mathbb{G}_1, g, q)$ *algorithm. This gives each moderator a voting key* δ_i*, and publishes a moderator public key* Δ*.*

- *UserJoin*$()$ *The* *UserJoin* *protocol is run by the group manager and a user. The GM authenticates the user and confirms that she is eligible to join the system. If so, the GM issues a credential* $C(x)$ *over the user's secret key* x *to the user. The group manager stores additional information to ensure that the user only receives one credential.*

- *PerformTransaction*(ϵ, τ) *The* *PerformTransaction* *protocol is run between the user and a service provider. Let* x *be the user's secret key,* ϵ *the current epoch, and* τ *the transaction that the user wants to perform. The user calculates the current epoch generator* $g_\epsilon = H(\epsilon) \in \mathbb{G}_1$*, generates* $z \in_R \mathbb{Z}_q$*, and calculates the linking token* $r = g_\epsilon^x$ *and auxiliary information* $t_1 = h^z$ *and* $t_2 = \hat{e}(g_\epsilon, h^z)^x$*. Next, she creates the encrypted*[4] *linking token* $T = \mathsf{TDH.Enc}(\Delta, r, \tau)$ *and generates a signature proof of knowledge that she generated all these values correctly:*[5]

$$\pi = SPK\{(C, x, z, \alpha) : C(x) \wedge t_1 = h^z \wedge t_1^x = h^\alpha \wedge \\ T = \mathsf{TDH.Enc}(\Delta, g_\epsilon^x, \tau) \wedge t_2 = \hat{e}(g_\epsilon, h)^\alpha\}(\tau). \quad (1)$$

  In this proof $\alpha = xz$*. The user sends the transaction record* $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ *to the SP. The SP executes the transaction if the proof* π *is correct. The SP stores the transaction record* t*.*

- *VoteToLink*(δ_i, t) *On input of a transaction record* $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ *and voting key* δ_i *the moderator checks that the label in ciphertext* T *matches the transaction* τ*. If the label is correct, it calculates the decryption share* $\psi_i = \mathsf{TDH.Dec}(T, \delta_i)$ *and sends the resulting decryption share* ψ_i *to the service provider.*

- *Link*$(t, \{\psi_{i_1}, \ldots, \psi_{i_k}\})$ *On input of* k *decryption shares* $\{\psi_{i_1}, \ldots, \psi_{i_k}\}$ *and a transaction record* $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ *the service provider recovers the linking token* $r = \mathsf{TDH.Combine}(T, \psi_{i_1}, \ldots, \psi_{i_k})$ *(or aborts if the decryption fails). The SP now uses this linking token to find all other transactions by the same user in epoch* ϵ*: for each transaction* $t' = (\tau', T', t_1', t_2', \epsilon, \pi')$ *in epoch* ϵ *the SP tests whether*

$$\hat{e}(r, t_1') = t_2'.$$

  If the equation holds, the SP adds transaction τ' *to the list of transactions by the same user.*

The vote-to-link mechanism is effected as follows. When a moderator detects a bad transaction, it calls VoteToLink and sends the resulting decryption share to the SP (the decryption shares are stored with the transaction). When the SP has collected enough decryption shares it runs Link to find all other transactions by the same user in that epoch.[6] In addition, if a user's linking token is recovered within the current epoch, the SP can use the test in Link to block transactions from that user for the remainder of this epoch by testing the submitted t_1'', t_2'' values.

[4]It is essential that the user verifies that the moderators' public key Δ indeed belongs to the moderators.

[5]The part of the proof involving t_1 and t_2 has been adapted from Nakanishi and Funabiki [24].

Correctness of this scheme is easy to verify. The service provider verifiers the proof π which ensures that the user has a valid credential, that T contains an encrypted linking token, and that the auxiliary values t_1, t_2 are well-formed. Since credentials are unforgeable, a user is forced to create linking tokens belonging to her own identity. Because of the security of the revocation scheme by Nakanishi and Funabiki [24], the linking tokens can be used to link all the user's transactions within this epoch.

4.3 User anonymity

We now show that well-behaving users remain anonymous. More precisely, we show that unlinked users in epoch ϵ, i.e., users for which the linking token for epoch ϵ has not been recovered, are anonymous. The full user anonymity game is given in Appendix B, see also Section 2.2.

THEOREM 2. *The vote-to-link scheme has user anonymity in the random oracle model provided that the DDH assumption holds in* $\mathbb{G}_1$ *and the DBDH assumption holds.*

PROOF SKETCH. The proof consists of two steps. One, because of the CCA security of TDH2' encryption scheme the adversary does not learn anything about the plaintext of the encrypted linking token T. Hence, it does not learn anything about the linking token. In fact, we can replace the real encrypted linking token by a random one without the adversary noticing. We use the random oracle to simulate the proof π. Two, because the linking token is unknown, the DBDH assumption and the proof by Nakanishi and Funabiki [24] ensures that the adversary cannot link users through the auxiliary information either. □

5. A VOTE-TO-LINK SCHEME WITH MODERATOR ANONYMITY

In this section, we present a scheme that allows the moderators to vote anonymously. We first present the idea of the scheme before describing the scheme in full.

5.1 The idea

In the scheme described in the previous section, moderators cannot vote anonymously. This is because Shamir's secret sharing scheme is used (as part of the threshold encryption scheme) to share the moderators' private key κ. That is, we create a polynomial f of degree $k-1$ such that $f(0) = \kappa$. Each moderator is then given a share $(i, f(i))$ as its decryption key. While partially decrypting typically hides $f(i)$, the index i (which identifies the moderator) is essential to recover the plaintext.

To achieve full anonymity for the moderators, the user and the service provider run a simple three step protocol.

[6]The SP should guide the voting process to prevent the situation where there are many bad transactions with only a few votes. Instead, the SP should rank suspicious transactions by the number of votes so that if a transaction is indeed bad, the threshold is reached quickly. In fact, even non-moderators could report bad actions to bring them to the moderator's attention more quickly.

In essence, the user and the SP run a small mix network, ensuring that as long as the service provider and the user do not collude, no party can determine the identity of the voting moderators. The enable this protocol, each moderator has a public key encryption key. The idea is as follows:

1. The service provider shuffles and randomizes the moderators' public keys and sends them to the user.

2. The user creates a new threshold encryption key and encrypts its linking token against this new key. To facilitate decryption, she encrypts a shuffled decryption share for each of the shuffled and randomized public keys she received from the service provider.

3. The service provider unshuffles and derandomizes the ciphertexts containing the encrypted decryption shares and publishes them with the transaction record. Moderators will simply decrypt their ciphertexts to recover their decryption shares and cast their votes.

Assuming, for the moment, that both the user and the service provider are honest, it is easy to see that this gives anonymity for the voting moderators. First, because the service provider shuffles and randomizes the moderators' public keys, the user does not know to which moderator it gave which share. So, if a moderator reveals or uses a share, the user does not learn anything about the identity of the moderator. Second, because the user assigns random shares to each moderator, the service provider does not learn which moderator received which share. Hence, the service provider cannot recognize the voting moderators either.

In practice, we cannot assume that the user is honest—she might want to avoid consequences of bad behavior. Therefore, the user has to prove that she acted honestly.

Similarly, we also cannot assume that the service provider is honest. In particular, the SP can deanonymize the user if it itself generated new keys for the moderators, hence the service provider too needs to prove that it shuffled and randomized the original public keys correctly.

5.2 Full anonymity for moderators

We now present a vote-to-link scheme where the moderators are fully anonymous.

Scheme 4. *The fully anonymous protocol is based on our original vote-to-link scheme (see Scheme 3). We only show the modifications.*

- $\mathsf{SetupModerators}(1^\ell, n, k)$ *Each moderator i generates an ElGamal key-pair $(x_i, h_i) = \mathsf{KeyGen}(\mathbb{G}, g, q)$ and publishes its public key h_i. It privately stores its voting key $\delta_i = x_i$.*

- $\mathsf{PerformTransaction}(\tau)$ *The $\mathsf{PerformTransaction}$ protocol is run between a user and a service provider on input of a transaction τ. It proceeds in three steps.*

  Step 1. *The user registers the transaction τ, the service provider replies by sending a list of randomized and shuffled moderator public keys $\hat{h}_1, \ldots, \hat{h}_n$, i.e., the SP picks a permutation $\sigma_{SP} : [n] \to [n]$ and randomizers $\alpha_i \in_R \mathbb{Z}_q$ and sets:*

$$\hat{h}_i = \mathsf{Randomize}(h_{\sigma_{SP}(i)}, \alpha_i).$$

  The service provider then sends $\hat{h}_1, \ldots, \hat{h}_n$ to the user together with a proof

$$\pi_{SP} = SPK\big\{((\alpha_i)_{i\in[n]}, \sigma_{SP}) \ : \ \hat{h}_i = \mathsf{Randomize}(h_{\sigma_{SP}(i)}, \alpha_i)\big\}(\tau)$$

  that it randomized and shuffled actual moderator keys, see Appendix C for how to construct π_{SP}.

  Step 2. *The user checks proof π_{SP} and aborts if it is incorrect.*[7] *Then, she runs $\mathsf{TDH.KeyGen}(\mathbb{G}_1, g, q)$ to create a fresh TDH2' private key κ and public key w with verification keys $w_i = g^{\kappa_i}$ corresponding to the decryption key shares (i, κ_i) (since she runs it by herself, she can immediately generate the secret-sharing polynomial f). Next, she creates her encrypted linking token $T = \mathsf{TDH.Enc}(r, w, \tau) = (c, L, u, v, e, d)$ and auxiliary information t_1, t_2, and proves that she did so correctly by calculating proof π as in equation 1 in the original protocol, except for the fact that she uses a fresh key for the moderators.*

  Next, she encrypts decryption key shares for each of the randomized public keys $\hat{h}_1, \ldots, \hat{h}_n$:

  1. *Let $\psi_i = (i, u_i) = \mathsf{TDH.Dec}(T, (i, \kappa_i))$ be the decryption shares. The user generates an ElGamal key-pair $(x', h') = \mathsf{KeyGen}(\mathbb{G}, g, q)$ and encrypts the decryption share's components:*[8]

$$\hat{c}_i = \mathsf{Enc}(g^i, u_i; h')$$

     She proves that these are generated correctly, i.e., that $u_i = u^{\kappa_i}$, using the following proof:

$$\pi_a = SPK\Big\{(x', (\kappa_i)_{i\in[n]}) \ : \ h' = g^{x'} \wedge \forall i \in [n]\big[w_i = g^{\kappa_i} \wedge \hat{c}_i = \mathsf{Enc}(g^i, u^{\kappa_i}; h')\big]\Big\}(\tau).$$

  2. *She chooses a random permutation $\sigma_U : [n] \to [n]$ and permutes the ciphertexts $\hat{c}_i$ according to σ_U:*

$$\tilde{c}_i = \hat{c}_{\sigma_U(i)}\mathsf{Enc}(1, 1; h'),$$

     and proves that she shuffled correctly:

$$\pi_b = SPK\Big\{(\sigma_U) \ : \ \forall i \in [n]\big[\tilde{c}_i = \hat{c}_{\sigma_U(i)}\mathsf{Enc}(1, 1; h')\big]\Big\}(\tau)$$

     using, for example, Groth's verifiable shuffle protocol [16].

  3. *Finally, the user reencrypts the shuffled ciphertexts $\tilde{c}_i$ to the randomized and shuffled moderators' public keys $\hat{h}_i$ to get ciphertexts c_i and proves that she did so correctly:*

$$\pi_c = SPK\Big\{((j_i, \kappa'_i)_{i\in[n]}) \ : \ \forall i \in [n]\big[\tilde{c}_i = \mathsf{Enc}(g^{j_i}, u^{\kappa'_i}; h') \wedge c_i = \mathsf{Enc}(g^{j_i}, u^{\kappa'_i}; \hat{h}_i) \ \big]\Big\}(\tau).$$

     Here the user uses that she knows the content of all the ciphertexts.

[7] In addition, as with the previous scheme, the user needs to verify the authenticity of the moderators' public keys.

[8] Throughout the remainder of this scheme, when we operate on tuples of messages, we simply write $\mathsf{Enc}(m_0, m_1; h)$ instead of the longer $(\mathsf{Enc}(m_0, h), \mathsf{Enc}(m_1, h))$.

The user sends $t_U = (T, t_1, t_2, w, h', \pi, \pi_a, \pi_b, \pi_c, (\hat{c}_i, \tilde{c}_i, c_i, w_i)_{i \in [n]})$ *to the SP.*

Step 3. *The service provider receives* t_U *and:*

– *it checks that the verification keys* $w_1, \ldots, w_n$ *and* $w_0 := w$ *are consistent using Lagrange interpolation. In particular, it sets* $\mathcal{I} = \{0, \ldots, k-1\}$ *and checks that, for all* $i \in \{k, \ldots, n\}$,

$$w_i = \prod_{j \in \mathcal{I}} w_j^{\lambda_j^{\mathcal{I}}(i)};$$

– *it verifies the correctness of the proofs* π, π_a, π_b, *and* π_c*; and*

– *it recovers ciphertext* C_i *for moderator* i*:*

$$C_i = \mathsf{Derandomize}(c_{\sigma_{SP}^{-1}(i)}, \alpha_{\sigma_{SP}^{-1}(i)}), \qquad (2)$$

and publishes these together with t_U *as the transaction record.*

- $\mathsf{VoteToLink}(\delta_i, \tau)$ *Moderator* i *decrypts* C_i *using its voting key* δ_i *to recover the pair* (g^j, u^{κ_j}) *for some* j*. The moderator uses an algorithm like baby-step giant-step to recover* j *(this only takes* $O(\sqrt{n})$ *time). Then, it publishes* (j, u^{κ_j})*.*

Anonymity of the scheme.

First, we show anonymity for the user. The user sets up a completely new system for every transaction, but while t_U contains many values, these are either already present in the original protocol (like the public key w and verification keys w_i) or zero-knowledge proofs. So, we only need to concern ourselves with the ciphertexts. Of these, only moderator i can decrypt C_i. This ensures that every moderator receives at most one share, and hence guarantees user anonymity.

We already argued that the shuffling by both the user and the service provider ensures anonymity for the moderators as long as the user and the SP do not collude.

6. VOTE-TO-LINK IN PRACTICE

In this section we explain how to choose parameters, and analyse the efficiency of our vote-to-link schemes.

6.1 Choosing parameters

The length of an epoch determines the utility of linking a user's action. Choosing a longer epoch ensures that a malicious user's actions become linkable over a longer time span. Hence, it is easier to locate all other bad actions by that same user. On the other hand, choosing longer epochs also increases the invasiveness of linking actions. Special care has to be taken when the decision to vote is very subjective. Since a vote to link effectively reduces a user's anonymity to an epoch-dependent pseudonym, the length of an epoch should be smaller in this case.

For our leading example, editing Wikipedia, we believe that an epoch of 24 hours strikes a good balance: the system can easily recover from bursty misbehavior, while inadvertent linking is not too damaging.

To reduce inadvertent linking to a minimum, moderators and the voting threshold must be carefully selected. Ideally, a system has only a few trusted moderators, in which case

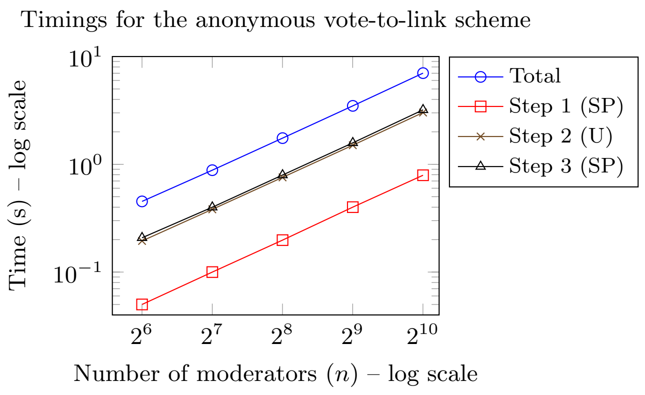

Figure 2: Running time of the total $\mathsf{PerformTransaction}$ protocol of the anonymous vote-to-revoke scheme for a threshold $k = 32$, as well as running times for the individual steps. As expected, the running time increases linearly in the size of n. (Communication cost is not taken into account.)

the threshold can be small as well. If the number of moderators cannot be kept low, the threshold should be set to a sizable percentage of the number of moderators to ensure that the 'bad apples' cannot influence the vote too much.

6.2 Prototype implementation

To evaluate the performance of our two schemes, we built and tested a proof-of-concept implementation[9] in C using the RELIC cryptographic library [2].[10] We implemented only the protocols' cryptographic parts, but the communication parts are easily added. We ran all experiments on a single core of an Intel i5-6200U running at 2.30 GHz. Most code can be optimized further, and is easily parallelizable.

We used BBS+ signatures [3] as anonymous credential scheme (recent work [6] shows that these are also secure in the type III pairing setting that our schemes require).

The non-anonymous vote-to-revoke scheme is very fast. The $\mathsf{PerformTransaction}$ protocol takes about 5 ms for both the user and the service provider. The size of a single transaction record is about 1 KiB. $\mathsf{VoteToLink}$ takes about 0.5 ms. After the SP receives k votes, it recovers the linking token in about 20 ms (for $k = 32$) and can then check about 2800 transactions per second against the recovered linking token.

Figure 2 shows the running times of the $\mathsf{PerformTransaction}$ protocol for the anonymous protocol. The $O(n)$ complexity of the proofs[11] causes a significant slow-down. However, a running time of 7 seconds for a large number of moderators like 1024 is still practical. The size of the transaction record increases linearly from 47 KiB for $n = 64$ to 732 KiB for $n = 1024$. Moderators can run $\mathsf{VoteToLink}$ in 5 ms for $n = 1024$.

7. RELATED WORK

Many systems have been developed that cover methods for dealing with misbehaving users. Perhaps one of the most well known approaches are group signatures [9], which allow

[9]https://github.com/wouterl/vote-to-link

[10]We setup RELIC to use an optimized 254 bits BN curve.

[11]We replaced the verification of the w_i's in step 3 by a $O(n)$ probabilistic variant in our prototype implementation, so the complexity is independent of k.

group members to anonymously sign messages on behalf of the group. We refer to for example Manulis et al. [23] for an overview. An essential aspect to group signatures is the ability to open or trace a signature to determine which group member created it. Typically, this power rests with either the group manager or a separate tracing manager.

Recent work has looked into creating group signatures where the tracing manager is distributed among many parties [14, 15, 22, 31]. Only when a certain number of parties agree can the signer of a signature be traced, i.e., identified. Another extension to group signatures is that of traceable signatures [20], which allows the group manager to produce tracing tokens that can be used to recognize signatures by the same user—similar to our vote-to-link scheme. The novelty of our scheme is the combination of the idea of a distributed tracing manager—the moderators in our case—and the linking of a user's actions within a *limited* time window (in the traceable signatures scheme [20] linking is global).

Many recent group signature schemes also offer revocation: giving a revocation manager the ability to block misbehaving users. In particular, we wish to highlight Nakanishi and Funabiki's scheme [24]. It offers backwards unlinkable revocation by using revocation tokens that are different for every epoch. Upon revocation the tokens for all future epochs are published. We employ this mechanism, however, the revocation token for the current epoch is instead used to temporarily *link* a user's actions.

Also outside the realm of group signatures researchers looked into methods for deterring misbehavior. For example, the Nymble system seeks to block misbehaving Tor users, but uses a trusted party to do so [29]. The blacklistable anonymous credentials system (BLAC) [28] instead blocks users without using a trusted party. In both cases the goal is to simply block the misbehaving user. But, as we have indicated, often other methods are required to recover from abuse. If this is the case, our scheme can be used on top of such an anonymous blocking scheme.

Anonymous reputation systems like RepCoin [1] and AnonRep [30] offer another method to hold users accountable for their actions. Their behavior is reflected in their reputation. While RepCoin only supports positive feedback, AnonRep also allows actions to be downvoted. In both cases, the change in reputation only affects future actions. Hence, past misbehavior remains unidentified, contrary to our system.

Desmedt and Frankel were among the first to mention threshold cryptosystems and threshold encryptions [11]. In this paper we created a verifiable variant of the more recent Shoup and Gennaro [27] scheme which is CCA secure. Of more recent interest is the scheme by Delerablée and Pointcheval [10] which allows encryptors to precisely select the threshold under which the message is to be encrypted. To make this possible, a trusted party—similar to the group manager in group signatures—assigns decryption keys to decryptors. This makes this scheme not applicable to our scenario because we do not want such an all-powerful party to exist. Another threshold encryption scheme is the one by Libert and Yung [21], which is secure against adaptive adversaries and is secure in the standard model. The downside is that they have to rely on a less common setting—a composite order bilinear group. Furthermore, the message is embedded in the target group, making it impossible to apply the linking mechanism which itself relies on pairings.

Either by design or as a result of the specific construction, these threshold cryptosystems identify the decryptors—the moderators in our scheme. In most, this is a direct result of using Shamir's secret sharing scheme. There has been some research into anonymous secret sharing schemes, but none of these are applicable to our scenario. Blundo and Stinson's anonymity of secret sharing schemes [4] merely deals with the theoretical optimization of secret share sizes when you have to include the identity of the share-holder in the share. The shares themselves may still identify the share-holder, in fact, the Shamir share $(i, f(i))$ (rather than just $f(i)$) is 'anonymous' by their definition. Guillermo et al. present some truly anonymous schemes [17]—they prove that given the shares you cannot determine the share-holders from which these shares originated. However, the schemes they present are theoretical and lack efficient secret recovery algorithms. This makes it impossible to use them to construct an anonymous threshold cryptosystem.

8. CONCLUSIONS

In this paper we have introduced a new efficient vote-to-link scheme in which a group of moderators can decide to link a user's actions within an epoch when they detect abuse. We think that this scheme is especially useful in combination with an anonymous blocking scheme like BLAC: BLAC ensures that the user cannot continue her abuse, while the vote-to-link system ensures that abuse prior to the block can be revealed if the moderators decide this is necessary.

The concept of linking a user's actions within epochs also sets our solution apart from the open mechanism of group signatures, that always work on a single signature—and future ones in case of revocation—rather than identifying a set of signatures belonging to the same user, and traceable signatures, that do not limit the tracing capabilities.

In addition to this, we introduced a less efficient, but still practical variant that offers anonymity for the moderators in addition to anonymity for the users at the cost of efficiency. We believe that this enables interesting scenarios, in particular if the moderators could otherwise experience retaliation because of their actions, or could be coerced into acting. We do wonder if it is possible to build a non-interactive fully anonymous version of our scheme. In effect, this would give an anonymous threshold cryptosystem.

9. REFERENCES

[1] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin. Reputation Systems for Anonymous Networks. In *PETS 2008*, LNCS 5134, pages 202–218. Springer, 2008.

[2] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[3] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In *SCN 2006*, LNCS 4116, pages 111–125. Springer, September 2006.

[4] C. Blundo and D. R. Stinson. Anonymous secret sharing schemes. *Discrete Applied Mathematics*, 77(1):13–28, 1997.

[5] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* PhD thesis, 2000.

[6] J. Camenisch, M. Drijvers, and A. Lehmann. Anonymous Attestation Using the Strong Diffie

Hellman Assumption Revisited. *IACR Cryptology ePrint Archive*, 2016:663, 2016.

[7] J. Camenisch and A. Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO 2002*, LNCS 2442, pages 61–76. Springer, 2002.

[8] J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. In *CRYPTO 1997*, LNCS 1294, pages 410–424. Springer, 1997.

[9] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT '91*, LNCS 547, pages 257–265. Springer, 1991.

[10] C. Delerablée and D. Pointcheval. Dynamic Threshold Public-Key Encryption. In *CRYPTO 2008*, LNCS 5157, pages 317–334. Springer, 2008.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO '89*, LNCS 435, pages 307–315. Springer, 1989.

[12] J. R. Douceur. The Sybil Attack. In *IPTPS 2002*, LNCS 2429, pages 251–260. Springer, 2002.

[13] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO '86*, LNCS 263, pages 186–194. Springer, 1986.

[14] J. Furukawa and S. Yonezawa. Group Signatures with Separate and Distributed Authorities. In *SCN 2004*, LNCS 3352, pages 77–90. Springer, 2004.

[15] E. Ghadafi. Efficient Distributed Tag-Based Encryption and its Application to Group Signatures with Efficient Distributed Traceability. In *LATINCRYPT 2014*, LNCS 8895, pages 302–322. Springer, 2014.

[16] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *J. Cryptology*, 23(4):546–579, 2010.

[17] M. Guillermo, K. M. Martin, and C. M. O'Keefe. Providing Anonymity in Unconditionally Secure Secret Sharing Schemes. *Des. Codes Cryptography*, 28(3):227–245, 2003.

[18] R. Henry and I. Goldberg. Formalizing Anonymous Blacklisting Systems. In *S&P 2011*, pages 81–95. IEEE Computer Society, 2011.

[19] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Algorithmic Number Theory 2000*, LNCS 1838, pages 385–394. Springer, 2000.

[20] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable Signatures. In *EUROCRYPT 2004*, LNCS 3027, pages 571–589. Springer, 2004.

[21] B. Libert and M. Yung. Non-interactive CCA-Secure Threshold Cryptosystems with Adaptive Security: New Framework and Constructions. In *TCC 2012*, LNCS 7194, pages 75–93. Springer, 2012.

[22] M. Manulis. Democratic Group Signatures: on an Example of Joint Ventures. In *ASIACCS 2006*, page 365. ACM, 2006.

[23] M. Manulis, N. Fleischhacker, F. Günther, F. Kiefer, and B. Poettrering. Group Signatures: Authentication with Privacy. Technical report, Bundesamt für Sicherheit in der Informationstechnik, 2012.

[24] T. Nakanishi and N. Funabiki. Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps. In *ASIACRYPT 2005*, LNCS 3788, pages 533–548. Springer, 2005.

[25] T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In *EUROCRYPT '91*, LNCS 547, pages 522–526. Springer, 1991.

[26] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[27] V. Shoup and R. Gennaro. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. In *EUROCRYPT '98*, LNCS 1403, pages 1–16. Springer, 1998.

[28] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs. *ACM Trans. Inf. Syst. Secur.*, 13(4):39, 2010.

[29] P. P. Tsang, A. Kapadia, C. Cornelius, and S. W. Smith. Nymble: Blocking Misbehaving Users in Anonymizing Networks. *IEEE Transactions on Dependable and Secure Computing*, 8(2):256–269, 2011.

[30] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford. Anonrep: Towards tracking-resistant anonymous reputation. In *NSDI 2016*, pages 583–596. USENIX Association, 2016.

[31] D. Zheng, X. Li, C. Ma, K. Chen, and J. Li. Democratic group signatures with threshold traceability. Cryptology ePrint Archive, Report 2008/112, 2008. http://eprint.iacr.org/.

APPENDIX

A. CCA SECURITY OF TDH2'

The proof of security for the original TDH2 scheme strictly relies on the hash-function used to create the ciphertext element $c = m \oplus H(w^r)$. Since we do not have such a hash function, we give a new proof for the following theorem. But, before we can do so, we need to define the CCA security game for threshold encryption schemes. We specify the game directly for our TDH2' scheme.

GAME 1 (THRESHOLD CCA SECURITY [27]). *The threshold CCA game between an adversary and the challenger proceeds as follows.*

Setup phase *The adversary chooses the number of decryptors n and the threshold k. It also chooses $k-1$ decryptors it wants to corrupt.*[12] *The challenger runs the* $\mathsf{TDH.KeyGen}(n, k, \mathbb{G}, g, q)$ *algorithm. It gives the private decryption keys of the $k-1$ corrupted servers to the adversary, and keeps the others for its own use.*

Query phase *In the query phase, the challenger can make any* $\mathsf{TDH.Dec}$ *queries of any of the non-corrupted host with ciphertexts and labels of its choosing.*

Challenge phase *At some point the adversary chooses messages $m_0, m_1 \in \mathbb{G}$ and a label L and sends them to the challenger. The challenger randomly picks $b \in_R \{0,1\}$ and sends $\psi = \mathsf{TDH.Enc}(w, m_b, L)$ to the adversary.*

Restricted query phase *The adversary can make* $\mathsf{Decrypt}$ *queries as before, except at ψ.*

Output phase *Finally, the adversary outputs a bit b' as its guess for b.*

[12]Note this is the static model where the to be corrupted servers have to be announced up front.

The adversary wins if $b' = b$.

We now give a small lemma that we will use in our proof.

Lemma 1. *The implicit proof of knowledge in* TDH.Enc *can be simulated in the random oracle model for H'.*

Proof. Consider the zero-knowledge proof in TDH.Enc (given by e and d). Given any $u, v \in \mathbb{G}$ proceed as follows. Generate a random $e, d \in \mathbb{Z}_q$ and set

$$\hat{u} = g^d u^{-e} \qquad \hat{v} = \bar{g}^d v^{-e}.$$

Then, using the fact that we operate in the random oracle model, back patch H' such that $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$. It is easy to see that this proof verifies. Also, since $\hat{u}, \hat{v}$ are random and generated by us, the back patching fails (i.e., because the adversary already queried the hash function on this precise input) with negligible probability. □

Proof of theorem 1. Recall, the ciphertext is given by $\psi = (c = m \cdot w^r, L, u = g^r, v = \bar{g}^r, e, d)$, where (c, u) forms the actual encoding part, and (v, e, d) the proof of correctness. This proof proceeds in two steps. First, we show that an adversary cannot detect that we replace the correct v component in the challenge ciphertext with a random element from $\mathbb{G}$, provided the DDH assumption in $\mathbb{G}$ holds. Second, we show that we can also replace the correct c component with a random element from $\mathbb{G}$ without the adversary detecting this, again, provided the DDH assumption in $\mathbb{G}$ holds. Clearly in the latter case the ciphertext is essentially random, so the adversary does not have an advantage. This proves CCA security. We now give the details.

We first prove that we can replace v in the challenge ciphertext with a random element from $\mathbb{G}$. Assume that an adversary exist that can detect whether v has been replaced by a random element. We show how we can use such an adversary to solve a DDH instance. We simulate the entire game honestly, except for the challenge query. Let $(g, X, Y, Z) = (g, g^x, g^y, g^z)$ be a DDH instance in $\mathbb{G}$. Our goal is to decide whether $z = xy$ or not. To do so, we will encode this problem into v. If $z = xy$ then v is correctly formed, otherwise it is random. Thus, any algorithm that can decide on the well-formedness of v can be used to solve the DDH-problem.

We setup the system as in the TDH.KeyGen$(n, k, \mathbb{G}, g, q)$ algorithm, with one exception. Instead of generating $\bar{g}$ randomly we pick $\beta \in_R \mathbb{Z}_q$ and set $\bar{g} = Y^\beta$. Clearly, $\bar{g}$ is a random generator from $\mathbb{G}$ as required. Let κ be the private key. Obviously, we can answer all decryption queries honestly, as the challenger knows all the required keys.

Now we show how to answer the challenge query. Pick a random element $\alpha \in_R \mathbb{Z}_q$ and set $u = X^\alpha$, so $r = x\alpha$. Then, $c = m \cdot u^\kappa$ is correctly formed. We let $v = Z^{\alpha\beta}$. Now, if $z = xy$ then $v = Z^{\alpha\beta} = (g^{y\beta})^{x\alpha} = \bar{g}^{x\alpha} = \bar{g}^r$ as required. Otherwise, v is a random element in $\mathbb{G}$. The proof of knowledge we simulate as per Lemma 1. So the simulation is perfect, and hence any adversary that can distinguish between a correctly formed v and a random element can be used to solve the DDH-problem.

Now that we have seen that we can replace the v element with a random element under the DDH assumption, we proceed by replacing the c component with a random element. Again, we assume that there exist an adversary that can detect whether c is correctly formed, given that v has already been replaced by a random element.

This time, the setup is more complicated. Again, we will use a DDH instance (g, X, Y, Z) in $\mathbb{G}$. Let g be the generator. This time, we let the public key $w = X$, so $\kappa = x$ (but, we do not know κ). Assume, w.l.o.g., that the corrupted servers are numbered $1, \ldots, k-1$. Choose their key-shares randomly $\kappa_1, \ldots, \kappa_{k-1} \in_R \mathbb{Z}_q$, and let $w_i = g^{\kappa_i}$ be their corresponding verification keys. Let $\mathcal{I} = \{0, 1, \ldots, k-1\}$, then for $k \le j \le n$ we have

$$w_j = w^{\lambda_0^{\mathcal{I}}(j)} \prod_{i=1}^{k-1} w_i^{\lambda_i^{\mathcal{I}}(j)},$$

by Lagrange interpolation. Furthermore, we pick a random element $\alpha \in_R \mathbb{Z}_q$ and set $\bar{g} = w^\alpha$. This allows us to answer decryption queries, because the proof of knowledge essentially forces the adversary to give us $\bar{g}^r = (w^r)^\alpha$. This completes the setup.

We will now show how to answer decryption queries for server j, with $k \le j \le n$. Let $\psi = (c, L, u, v, e, d)$ be the ciphertext. If the ciphertext is not valid, simply return $(j, \perp)$. If the ciphertext is valid, then with overwhelming probability $v = \bar{g}^r$, and hence $v^{1/\alpha} = w^r = u^\kappa$. We now use Lagrange interpolation on the set $\mathcal{I} = \{0, 1, \ldots, k-1\}$ to find u_j:

$$u_j = v^{\lambda_0^{\mathcal{I}}(j)/\alpha} \prod_{i=1}^{k-1} u^{\kappa_i \lambda_i^{\mathcal{I}}(j)}.$$

Finally, we show how to deal with a challenge query of two messages $m_0, m_1 \in \mathbb{G}$ and label L. First, pick a bit $b \in_R \{0, 1\}$ and set $u = Y$, and $v \in_R \mathbb{G}$ (by our first step, this is as the adversary now expects). Finally, set $c = m_b \cdot Z$. Now, if $z = xy$ then c is correctly formed as before, otherwise c is random. Any adversary that distinguishes between a well-formed c and a random c breaks the DDH assumption.

We have seen how, in two steps, we can modify the protocol in such a way that the ciphertext gives no information about the plaintext. Hence, the adversary cannot win. □

B. ANONYMITY GAMES

In this section we formally define user anonymity and moderator anonymity using the following two security games.

Game 2 (User anonymity). *The following anonymity game is between an adversary $\mathcal{A}$ and a challenger. The game proceeds in five phases. Each user in the system is identified by a user identification number* `uid` *of the adversary's choosing. The challenger keeps track of a set of honest users $\mathcal{U}_H$ and a set of users $\mathcal{U}_C$ that are under the adversary's control.*

Setup phase *At the start of the game the adversary informs the challenger about the number of moderators n and the threshold k it wants to use. In addition, the adversary indicates a set $\mathcal{C} \subset \{1, \ldots, n\}$ of cardinality $k-1$ of moderators it wants to corrupt. The challenger runs SetupGM to setup the group manager and SetupModerators to setup the moderators, where the challenger controls the uncorrupted moderators, while the adversary controls the corrupted moderators in $\mathcal{C}$. Finally, the challenger sets $\mathcal{U}_H = \emptyset$ and $\mathcal{U}_C = \emptyset$.*

Query phase *In the query phase the adversary can make the following queries:*

AddU (`uid`) *The adversary can make an AddU(*`uid`*) query to request that a user with identifier* `uid` *is* *added to the system. The challenger creates this user and runs UserJoin on behalf of this user with the GM. The challenger stores the user's private information and the credential. It adds* $\texttt{uid}$ *to* $\mathcal{U}_H$.

JoinU (uid) *The adversary makes a JoinU(*$\texttt{uid}$*) query to request that a user it constructed, i.e., the adversary chooses the keys, joins the system. To this end, the adversary runs the UserJoin protocol—on behalf of the user—with the GM—controlled by the challenger. The new user will have identifier* $\texttt{uid}$ *and the challenger adds* $\texttt{uid}$ *to* $\mathcal{U}_C$.

CorruptU (uid) *The adversary can request to corrupt user with id* $\texttt{uid} \in \mathcal{U}_H$. *The challenger looks up the user's private information and credential and gives them to the adversary. It also adds* $\texttt{uid}$ *to* $\mathcal{U}_C$ *and removes* $\texttt{uid}$ *from* $\mathcal{U}_H$.

TransactSP $(\texttt{uid}, \epsilon, \tau)$ *To request that a user with* $\texttt{uid} \in \mathcal{U}_H$ *(the adversary can simulate this query for corrupted users) runs the PerformTransaction(*τ*) protocol for epoch* ϵ *where the adversary acts as SP,*[13] *the adversary can make a TransactSP(*$\texttt{uid}, \epsilon, \tau$*) query. The adversary receives all the information that the SP would normally receive, including the transaction record* t.

VoteToLink (j, t) *The adversary makes a VoteToLink(*j*,* t*) query to request the decryption share* ψ_j *from moderator* $j \notin \mathcal{C}$. *In response, the challenger runs VoteToLink(*δ_j, t*) on behalf of moderator* j *and returns the result to the adversary.*

Challenge phase *Eventually the adversary will select two users with identifiers* $\texttt{uid}_0, \texttt{uid}_1 \in \mathcal{U}_H$, *a transaction* τ *and an epoch* ϵ. *The challenger picks a bit* $b \in_R \{0,1\}$ *and acts as if the adversary called TransactSP(*$\texttt{uid}_b$*,* ϵ, τ*). Let* t *be the corresponding transaction record.*

Restricted query phase *After the challenge query the adversary can continue to make queries as before, with the following restrictions. It is not allowed to call CorruptU on the users* $\texttt{uid}_0, \texttt{uid}_1$ *nor is it allowed to make VoteToLink queries involving* t *(it can already create* $k-1$ *ballots because of the corrupted moderators it controls).*

Output phase *Eventually the adversary will output a guess* b' *of bit* b. *The adversary wins if* $b = b'$.

At any point in time the adversary can run Link (because anyone can run this algorithm).

GAME 3 (MODERATOR ANONYMITY). *The moderator anonymity game is a modification of the user anonymity game (see Game 2), again between an adversary and a challenger. It proceeds in four phases:*

Setup phase *At the start of the game the adversary informs the challenger about the number of moderators* n *and the threshold* k *it wants to use. The challenger runs SetupGM to setup the group manager and SetupModerators to setup the moderators. Finally, the challenger sets the set of corrupted moderators* $\mathcal{C} = \emptyset$.

Query phase *The adversary controls all users. It can make JoinU, TransactSP and VoteToLink queries as in the user anonymity game (note that the VoteToLink queries are restricted to uncorrupted moderators). Additionally, it can make the following query:*

CorruptM (i) *The adversary request the corruption of a moderator* i. *The challenger gives all keys of moderator* i *to the adversary and adds* i *to* $\mathcal{C}$.

Challenge phase *Eventually, the adversary requests votes on a valid transaction record* t *of its choosing. To do so, it sends* t *and two sets of moderators* M_0 *and* M_1 *to the challenger. The challenger verifies that the transaction record* t *is valid and new, that the sets are of equal size, i.e.,* $|M_0| = |M_1|$, *and that there are no corrupted moderators in the query sets, i.e.,* $(M_0 \cup M_1) \cap \mathcal{C} = \emptyset$; *the challenger aborts otherwise. Finally, the challenger picks a bit* $b \in \{0,1\}$ *and then returns VoteToLink(*i, t*) for each moderator* $i \in M_b$.

Restricted query phase *The adversary can make JoinU and TransactSP queries as before. The VoteToLink(*i, t*) query can only be made on non-challenge transaction records.*

Finally the adversary outputs a guess b' *for bit* b. *The adversary wins if* $b' = b$. *We say the voting scheme has* full moderator anonymity *if no adversary can win this game.*

C. SHUFFLING RANDOMIZED KEYS

In the first step of the fully anonymous vote-to-link protocol, the SP shuffles randomized moderator keys. Just as for the user's proof of shuffling, the SP uses Groth's verifiable shuffle protocol [16] to construct proof π_{SP}, however, the randomization of the moderators' public keys using Randomize complicates this proof slightly.

The trick to seeing why we can apply Groth's protocol is to reinterpret the Randomize function in a special ElGamal encryption scheme. In particular, let $\hat{g} \in_R \mathbb{G}_1$ be a random generator, and $g \in \mathbb{G}_1$ the corresponding public key. Then the encryption of $m \in \mathbb{G}_1$ is $\mathsf{Enc}'(m, g) = (m \cdot g^{\beta}, \hat{g}^{\beta})$, where $\beta \in_R \mathbb{Z}_q$. Note that when $\hat{g}$ is truly random the ciphertexts cannot be decrypted.

Now, we reinterpret the original public keys as ciphertexts in this scheme, i.e., h_i becomes $H_i = \mathsf{Enc}'(h_i, g) = (h_i, 1)$ (i.e., we use $\beta = 0$). Similarly, we can reinterpret $\hat{h}_i = \mathsf{Randomize}(h_{\sigma_{SP}(i)}, \alpha_i)$ as

$$\hat{H}_i = H_{\sigma_{SP}(i)} \cdot \mathsf{Enc}'(1, g) = (h_{\sigma_{SP}(i)} \cdot g^{\alpha_i}, \hat{g}^{\alpha_i}),$$

where encryption $\mathsf{Enc}'(1, g)$ uses ephemeral key α_i. Hence, $\hat{H}_1, \ldots, \hat{H}_n$ are simply shuffled and randomized versions of the original ciphertexts $H_1, \ldots, H_n$. And this is exactly what we can prove using Groth's verifiable shuffle protocol.

[13]Allowing the adversary to select the ϵ gives the adversary slightly more power, in an actual system time does not run backwards.