# PKI with a Little Help from My Friends: A Secure Decentralized Social Network

Jacky Chang `kyoki`
Christian J. Ternus `ternus`
Geoffrey Thomas `geofft`
Ian Ynda-Hummel `ianyh`

## Abstract

We present a design for a social network meeting the following three requirements: (1) maliciously mining data from the network is difficult, if not impossible, (2) users delegate independent and uncorrelatable sub-identities to compartmentalize different friends or sets of friends, and (3) interactions between friends do not reveal significant portions of the network graph to an untrusted party. We use a distributed system to prevent any central authority from being able to collect all of the data contained in the network, and broadcast all requests for information to a constantly changing "council" of nodes who monitor the traffic for large numbers of requests coming from a small number of clients. To balance the need for anonymous, unlinkable sub-identities while preventing vulnerability to so-called "Sybil attacks" we propose a dark web of trust using "trust tokens" distributed by users of the system to quantify trustworthiness of other parties during all interactions while minimizing the leakage of information concerning the shape of the social network. Though a practical and large-scale implementation would need to solve some inefficiencies, our design demonstrates that a secure system meeting these criteria is definitely realizable.

## 1 Introduction

With the advent of ubiquitous social networking, the problem of controlling one's personal information has become more important than ever. Many users of social networks would like to be confident of who has access to their personal information and data. The most popular social networks — Facebook, Twitter, MySpace, and so forth — rely on a single company running the network that users may not consider a *trusted* third party. Switching to a distributed model can address this problem, but it may not limit who can view a user's data, or address the issue of unintentional information leakage.

Straightforward cryptographic techniques are not quite sufficient, here. Encrypting data would certainly keep malicious eavesdroppers from accessing information, but would do nothing to address one of the major sources of data leakage in social networks: the shape of the networks themselves. If an attacker knows which nodes are connected to each other, he can infer data about one node from what he knows about the others. In fact, when given a large of amount of anonymized data, it is possible to reconstruct information about specific people, even when those people present no publically viewable information [5]. This is known as *reidentification* [7].

In order to avoid this, users need to have some way of connecting to other users without revealing their own network connections to each other. This concern must be balanced with the fact that some amount of information needs to be exchanged to prove identity (e.g., "I am a friend of a friend") in order to ensure that a malicious user is not presenting himself as legitimate one in order to gain access to privileged information.

In Section 2 we present the specification for this system, including our basic assumptions. Subsequent sections discuss our design and how it meets those specifications. Section 3 discusses how our system prevents the malicious mining of data, Section 4 discusses the manner in which identities are treated, and Section 5 outlines the manner in which our system prevents information leakage from user interaction.

## 2 Specification

The essential problem with current, popular social networks is the centralization of authority. A single entity has purview over a user's private and potentially sensitive information. Security policies can change in ways that leak your information in ways a user may not want and has no control over.

To solve this problem, we propose a distributed social networking protocol that meets the following three requirements.

- Although the system needs to be distributed to avoid a single third party, it should be sufficiently **hard to data mine** the social network; for instance, requests for an unusual amount of data should be obvious to the people providing this data.
- Users should be able to present **multiple sub-identities** that cannot be identified as related unless the user specifically authorizes this.
- Since we lack a single central authority, we need a trust metric similar to PGP's web of trust [6], but the entire goal of this network is to avoid publicizing friend relationships at the outset – we need a **dark web of trust** that requires revealing as little information about the friend graph as possible.

### 2.1 Assumptions

Our design relies on a few key assumptions.

- We assume the security of public-key cryptography primitives: signing, encryption, decryption, and verification.
- We leverage several protocols from the Tor project [2], and thereby assume the existence of the following secure primitives:
  - Decentralized directory protocols (for public key distribution)
  - Anonymous routing, as with `.onion` URLs
- We treat each user as an always-online node, and thereby assume that in practice users have access to trustworthy, always-running computers that will keep their private key secure.

The reasons for these assumptions will hopefully be made clear below; furthermore, they allow us to focus our design efforts on the interesting cryptographic problems presented by decentralizing a social network.

## 3 Preventing Data Mining

In order to prevent some central authority from having access to all of the aggregate data in the network, and therefore preventing a single point of corruption or attack, we use a distributed system, with clients keeping track of their own data and the data of friends who are a hop away from them on the network. Without a centralized authority, however, we cannot rely on a single source that meters requests made by a user.

Requests for data are broadcast anonymously to a set of $n$ nodes which we refer to as the "council." If, for example, Geoff requests data on Ian from Jacky, Jacky would tell the council "I won't say who I am, but Geoff was requesting data from me." She would not identify herself, as to do so would reveal a friend relationship between Geoff and Jacky to the council. If the council sees a large number of requests from Geoff during a small time period, they would be able to tell all of the other nodes in the network, "Stop accepting and responding to requests from Geoff."

After some set amount of time, the old council would be discarded and a new council selected using a majority of votes from the nodes in the network. If, during a council's tenure, one of the nodes drops, the council would select a new member. Decisions of the council, including elections and announcements, are made through the Paxos distributed consensus algorithm [**?**].

When first entering a network, users begin their own councils, of which they are the only member. When they are told of another council, usually upon joining an active network that already has a council, they drop their own and begin to report to the existing council. If the council does not yet contain $n$ members, likely because the network contains fewer than $n$ users, the user would become a member of the council. The presumption here is that although a single member of the council may be malicious, but the majority is unlikely to be, and this can be maintained through Paxos.

This council method of tracking information requests in a distributed system is very vulnerable to so-called "Sybil" attacks, in which a malicious user creates many anonymous sub-identities which he then uses to request information from the different nodes [3]. Since the identities cannot be connected to each other or the attacker's main identity, the council has no way of knowing that they are all the same person attempting to data mine. This gives rise to the need for a way to validate the trustworthiness of an account while still maintaining its anonymity, our solution to which is discussed in Section 5.

## 4 Anonymous Sub-Identities

We require that the system allow a user to create multiple, anonymous sub-identities. For example, a user might want separate personal and work profiles, and a particularly cautious user, one who believes in keeping his work and home life separate, may want to ensure that these identities cannot be artbitrarily connected. We note that in the system we propose, this indistinguishability is computational in that it is a quality of proper use of the system. If a user decides to put his social security number on both of his sub-identities' profiles, a human being would likely be able to connect the two.

Our solution design for this feature is simple, but both motivates and provides an essential piece of the solution to the dark web of trust. A sub-identity is simply a public/private key-pair. When a user wishes to create a new identity, he generates a new key-pair and associates a name with that keypair. Assuming that these key-pairs are, in fact, randomly generated, then determining a connection between two sub-identities is impossible without further information, assuming that the names are sufficiently distinct.

We must, of course, have a method for associating an identity with a given key-pair, which is complicated by the anonymity of these identities. In terms of a PGP-style web of trust, we need a way to show the existence of a path in the graph without revealing the path itself.

## 5 Dark Web of Trust

The standard model for calculating trust in a decentralized network is the "web of trust" [4] whose most well known implementation is for the PGP system. In the PGP web of trust, each user generates a public/private keypair using an encryption algorithm such as RSA or DSA and uploads their public key to a keyserver [1]. Users then meet up in person and verify each others' credentials (this is colloquially known as a "keysigning party") and, with their own private keys, sign certifications that the key does in fact belong to the name associated with the key. These signatures are then uploaded to the public keyservers.

Given that the signatures are all on the public servers, and the list of signatures on any given key can be accessed easily, a client can generate the "trust path" between one person's private key and another's public: e.g., Alice has verified Bob's identity, who has verified Celine's, who has verified Dedric's, and so forth. Given the existence of enough trust paths, Alice can be reasonably convinced, even in the event of a failure in judgment (or of malice) invalidating one of the trust paths,

$$\boxed{\text{T, 1, 84f2e014bd992...}}_{\text{G}}$$

Figure 1: A trust token.

that the key she has for Dedric belongs to the real Dedric.

The problem with this design is that is inherently not anonymous. Finding a trust path requires being able to search the entire graph to find a specific path that one can be reasonably sure is a trusted one. This is directly at odds with our desire that personal information, such as data on who is connected to whom, not be exposed without reason. Fortunately, what we really want is to be sure that a path exists and how strongly trusted such a path is. The path itself is is not necessary knowledge for a user to have.

## 5.1 Trust Tokens

The manner in which we establish the existence of a path in the web of trust is through **trust tokens**. Trust tokens can be thought of as units of trust: a user provides some number of trust tokens to each of his friends depending on the extent to which he trusts each of them. Those friends can then relay those trust tokens to their friends as they choose, and so forth. Trust tokens will then propagate through the web of friend relationships.

As a user of the system, if another user presents you with trust tokens that originated from you, you can be assured that there exists some path from you to the user in the web of trust. Furthermore, we design the distribution such that you know the *length* of the path, and thus its strength. Therefore, the amount of trust that you distribute becomes a meaningful statement.

A trust token is simply a randomly generated, unique value kept by a user. To distribute this token, the user distributes a signature of the form shown in Figure 1. Distributing trust tokens requires sending them to a specific user. If user $G$ wishes to give initial trust tokens to user $T$, he will generate sign$(T, 1, token_G)$ and send that value to $T$, which states that user $G$ has given user $T$ 1 unit of $G$'s trust.

Now consider that $T$ has 1 trust token of $G$'s. He can then relay up to half of that value to *each* of his friends. So if $T$ is friends with $I$ and $J$, he could distribute up to 0.5 units of $G$'s trust to each of them. They could, in turn, distribute up to 0.25 units of $G$'s trust to *their* friends, and so on. Then, when some user wishes to initiate a friend relationship with $G$, they can ensure $G$ that their name is valid by presenting his trust tokens. Relaying trust tokens does not cause the relayer to lose her tokens; it is simply a way to note that longer paths in an anonymous network are less trustworthy.

The reason we use a geometric decay, incidentally, is so that malicious users cannot amplify a trust token arbitrarily by bouncing it between the two of them (or between subidentities of the same user, even): at worst, a user could misrepresent $1 + \frac{1/2}{+}\frac{1/4}{+}\cdots = 2$ times as trust as they were legitimately given. We consider this mostly acceptable, but as described below these trust tokens will appear suspicious because of their length anyway.

To permit a web-of-trust path between any two arbitrary users, each user who joins the network will generate trust tokens and distribute them to their immediate contacts, who will relay them to each of theirs. Thus the system is not particularly efficient — each receiving user must keep trust tokens for every user they may possibly want to converse with, and each new friend link requires relaying trust tokens for each of these originating users — but our immediate goal is on corectness and security, not efficiency., provided that the computational resources remain far less than the computational resources required to break any of the cryptographic primitives. Future work may incorporate means to reduce the amount of local storage needed, for instance by dropping unimportant trust tokens and providing a mechanism to re-request them from the source (relayed along all the possible paths), although any such method would need to be careful to guard againsts side-channel attacks.

This system presents a potential failure mode. The distribution of trust tokens potentially reveals the shape of the network as the distribution must be verifiable in some way. We must specify the particular manner in which these trust tokens are redistributed, but this problem is solved using anonymous sub-identities.

## 5.2 Signing Chains

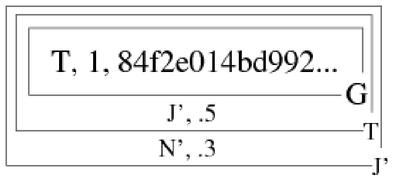

Figure 2: A forwarded trust token.

Let's say that user $T$ has 1 trust token from $G$. As described above, he has this in the form of a signature $\sigma = \text{sign}(T, 1, token_G)$. If $T$ has friend $J$ he wishes to trust with $G$'s tokens, he sends the signature $\text{sign}(J, 0.5, \sigma)$ to $J$. $J$ can then distribute those tokens to her friend $N$, and so on (Figure 2. But an identity is just a public/private keypair, so what exactly is being used to sign? If users were to use some universal identity to distribute these tokens, then any user with a trust token could reconstruct the entire path that token took.

That is where the anonymous sub-identities come in. Each user is known to his friends by a different name. So $T$ may be friends with $I$ and $J$, but $I$ knows $T$ by some identity $T_I$, whereas $J$ knows $T$ by some identity $T_J$, and these identities being just random keypairs are not directly connectable to each other.

The chain of signatures shown in Figure 2, then, is seen as a flow of information as shown in Figure 3. Information does not go directly from node to node, but is rather routed through the sub-identities of a given node. So when $G$ distributes a key to $T$, he sends $\sigma = \text{sign}_{G_T}(T_G, 1, token_G)$ to $T_G$, the name by which he knows $T$, where $\text{sign}_{G_T}$ is the signature using identity $G_T$, the name by which $T$ knows $G$. When $T$ redistributes these tokens to $J$, he looks up $J_T$, the name by which he knows $J$, and sends $\sigma' = \text{sign}_{T_J}(J_T, 0.5, \sigma)$ to $J_T$.

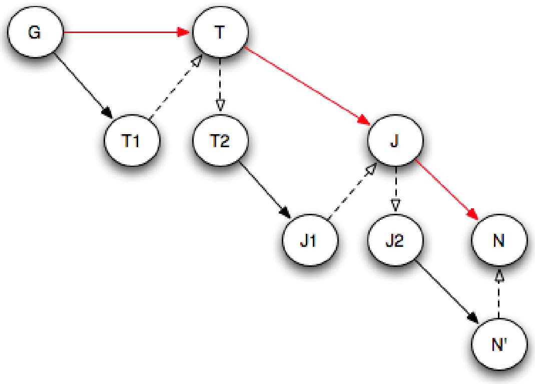

Figure 3: Compare the actual path the token takes in red, and the path $N$ sees in black. (Dotted lines display the flow of information, but are not recoverable by $N$.)

## 5.3 Information Leakage

While we designed the system to leak as little information as possible, certain pieces of information are leaked by the token distribution process. This is necessary on a fundemental level from the fact that while sub-identities should not be connectable to the primary identity, trust tokens do need to be connectable to a particular user.

The first piece of information leaked is the edge in the graph of friends between the token generator and the immediate friend to whom that token is distributed. In the example above, when $J$ receives the $G$ tokens from $T$, she sees the first transfer between $G_T$ and $T_G$, thus discovering the name by which $T$ knows $G$, and thus connecting $G_T$ to $G$.

Secondly, the last edge in the sign chain is leaked. In the example above, if $N$ receives the $G$ tokens from $J$ and wishes to initiate a friend request with $G$, he must present the tokens in his possession to $G$, verifying the existence of a path between them. In doing so, $G$ sees the name $N_J$, by which $J$ knows $N$, and he can therefore connect $N_J$ and whatever $N_G$ is presented to him. We note, however, that $G$ is unable to determine the identity of $J$.

One interesting thing to note here is that if if $J$ were to initiate a friend relationship with $G$, $G$ *would* be able to determine the identity of $T$, and thus discover the friend relationship between $T$ and $J$. However, the subgraph created by $G$, $T$ and $J$ becomes complete if the friend request is accepted, and we consider a complete friend graph to be a strong enough connection that the data leaked is not necessarily unknown.

Finally, at each link in the chain, a holder of a token learns some name of the person who distributed the token to him. For example, user $B$ has some tokens for user $X$ that he received from user $A$. If user $B$ redistributes these tokens to his friend user $C$, $C$ discovers the name by which $A$, whose identity is unknown to $C$, knows $B$.

The important thing to note here is that while information is leaked in the signature chain, the actual shape of the graph is not determinable, which also hints at a mechanism by which we can obscure this leaked information. The reason the path is obscured is because it is routed through anonymous sub-identities. We can, then, arbitrarily route through randomly generated sub-identities to obscure the leak of information.

In the first case, if $G$ wants to give $n$ trust tokens to $T$, he can start with $2^c n$ trust tokens, and route them through $c$ random sub-identities. By doing so, $J$ would not be able to determine the connection between $G_T$ and $G$ as there are $c$ hops between $G$ and $T$, and such hops could be legitimate token distributions.

Similarly, at each link in the chain, any user can route through its own arbitrary sub-identities to keep the user he forwards tokens to from discovering any of his legitimate names. In the example case, $N$ could route through some random number of sub-identities. There is, however, a trade off here that does not exist for the token generator. Routing through such identities increases anonymity, but reduces the weight of the resulting path.

## 6 Conclusion

In this paper, we have described a secure, decentralized social networking system that allows interaction and information exchange between individuals while obfuscating the overall structure of the graph. We have described how this system effectively propagates trust relationships between individuals while keeping information leakage to a minimum.

## 7 Acknowledgments

Thanks to Victor Costan for timely assistance and understanding.

## References

[1] ABDUL-RAHMAN, A. The pgp trust model. In *EDI-Forum: the Journal of Electronic Commerce* (1997), vol. 10, pp. 27–31.

[2] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13* (2004), USENIX Association, p. 21.

[3] DOUCEUR, J. The sybil attack. *Peer-to-Peer Systems* (2002), 251–260.

[4] GOLBECK, J., PARSIA, B., AND HENDLER, J. Trust networks on the semantic web. *Cooperative Information Agents VII*, 238–249.

[5] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy* (2008), vol. 8, Citeseer, pp. 111–125.

[6] STALLINGS, W. PGP web of trust. *Byte 20*, 2 (1995), 161–162.

[7] SWEENEY, L. Weaving technology and policy together to maintain confidentiality. *JL Med. & Ethics 25* (1997), 98.

## Appendix

To the tune of the Beatles' *With A Little Help (From My Friends)*:

*What would you do if I made friends with you?*
*Would you give your trust tokens to me?*
*Lend me your trust and I'll build you a graph*
*and I'll try not to leak your ID.*
*Oh, PKI with a little help from my friends...*

*Chorus:*
*Do you need any-body?*
*I need somebody to trust!*
*Could it be (spoken)* anybody?
*I want somebody to trust!*

*How can I link to a Council of Twelve*
*if I start in a graph all alone?*
*Maybe you'll offer to let me join yours*
*or I'll try building one of my own.*
*Oh, PKI with a little help from my friends...*

*[Chorus]*

*Why should I send your request to my friend*
*if you won't even tell me your name?*
*As long as he gave you some tokens of his*
*then he only has himself to blame...*
*Oh, PKI with a little help from my friends...*

*[Chorus]*