

The AEOLUS SybilGuard Service

Stefan Schiffner

February 17, 2010

SybilGuard in a Nutshell

SybilGuard assumes the existence of a **friendship network**. Every user exchanges with his friends a symmetric key. In terms of SybilGuard friendship means a pair of friends believes each other that they will not start a Sybil attack. The design of SybilGuard depends on the fact that the establishment of friendship relations is hard particularly for the adversary. This assumption can be ensured by real life interactions. Furthermore every node has a global, permanent and unique node ID.

Friendship network. After the key exchange a friendship graph is established. User are represented as nodes (vertexes) in N and exchanged keys are links (edges) in L . Since the keys are symmetric the graph is undirected. For the sake of notation $L_n \subseteq L$ with $n \in N$ denotes all links of n .

Random routes. Every honest user $n \in N$ choose a random permutation $r \subset L_n \times L_n$ of its links. This permutations creates random routes: A message arriving at link $l_1 \in L_n$ will be send to l_2 if and only if the incoming link is mapped to the outgoing link in the node's permutation (i.e. $(l_1, l_2) \in r$). As we will see these random routes are used to distribute witness and registry tables during the initialization phase of SybilGuard.

Witness and registry tables. Every node holds two tables for every one of its links. The witness table contains the random route's track of length ω that starts at the node in direction of the link. The registry table contains the route's track of the length ω that ends at the node. The route's track is a list of the node's identifier in order of appearance while following the route.

The protocol itself. Consider a verifier $v \in N$ wants to test whether a suspect node $s \in N$ is in the Sybil free group. The verifier requests s to send him his witness tables. Now v compares his witness tables with the ones from s to find a witness x . Therefore, v starts with one of his routes and looks for an intersecting node $x \in N$ on the suspects routes. That means v checks whether the chosen link's witness table have at least one common entry with one of the suspects witness tables.

After v found an x , v requests x for its registry tables in order to check if the suspect can reach x on one of its routes within ω steps. In particular that means v checks if s 's ID is in one of x 's registry tables. Is s registered at x , that

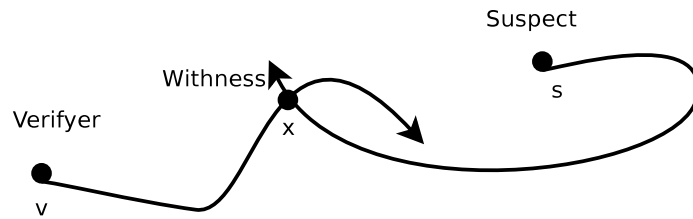


Figure 1: Single verification step

means s 's ID is in one of x 's registry tables, then v accepts s as an honest node with respect to the link that's witness table was used to find x .

The verifier reruns the protocol with the other links and finally accepts the node if most of the runs accept the node. In Figure is a single verification step shown.

The Prototype

The prototype is implemented in Java using JXTA technology. It runs in the AEOLUS testbed environment and offers its service to other JXTA services. However the prototype is meant to demonstrate the principal mechanisms of a social network based Sybil protection protocol and in that way it is very limited in its functionality. In particular the functionalities that are needed for making friends are not implemented. Friendship relations are randomly generated and distributed by a super node. Furthermore, it is assumed that communication lines are secure and thus all communication is not encrypted. Last, it is assumed that all nodes are honest about the content of their tables, that is, that the double checking with the witness is not needed and thus done.

These all seams to be very limeted, but it gives us already a wide playground to understand this type of Sybil protection protocols. The super node gives us the possibility to inject different shaped friendship networks, and the network implementation gives us an idea about network traffic and calculation overhead of single nodes. However there is much space for extentions of the prototype.

The general structure of the prototype is shown in Figure . On the left hand side of the diagram, the functionalities of a SybilGuard node are outlined, while on the right hand side the functionalities of the super node are sketched. Furthermore there is a vertical split. While the upper part of the figure outlines the functionalities for the actual protocol, the management parts are sketched in lower part. Note there are no centralized entities in the actual protocol part.

Protocol Part A node that runs a SybilGuarded application needs to run a `SybilGuardService`. This service has three tasks. (1) it builds up the `randomRoutes`, (2) it verifies other nodes (suspects) and (3) it responds to verification requests. A node updates its `randomRoutes` by sending its routes to its neighbors until the routs are getting stable. To verify a suspect the verifier sends a verification request, which is answered by the suspect by sending its current `randomRoutes`.

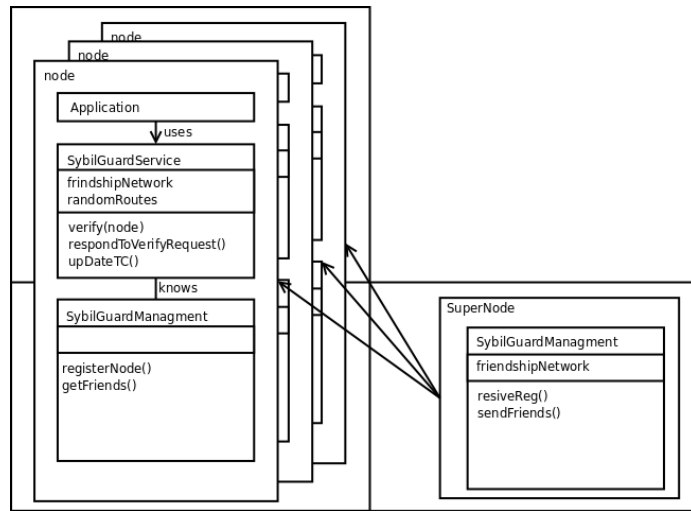


Figure 2: Structure of the prototype

Management Part The **SybilGuradManagement** is not part of the protocol as it would be used in the wild, since the protocol assumes the existence of friendship relations. However, it is an open research question how friendship relations are established. Every node registers itself at the **SuperNode** with its `nodeID`, by this the **SuperNode** gets all participants to know. This knowledge is used to generate and distribute the friendship relations. As argued before, in a real world application, friendship relations would be established differently, but due to the design of the prototype we are able to inject different network topologies.