TRANS
Internet-Draft
Intended status: Experimental
Expires: January 8, 2016

L. Nordberg
NORDUnet
D. Gillmor
ACLU
T. Ritter

July 07, 2015

Gossiping in CT
draft-linus-trans-gossip-ct-02

Abstract

This document describes three gossiping mechanisms for Certificate Transparency (CT) [RFC6962]: SCT Feedback, STH Pollination and Trusted Auditor Relationship.

SCT Feedback enables HTTPS clients to share Signed Certificate Timestamps (SCTs) (Section 3.2 of [RFC6962]) with CT auditors in a privacy-preserving manner by sending SCTs to originating HTTPS servers which in turn share them with CT auditors.

In STH Pollination, HTTPS clients use HTTPS servers as pools sharing Signed Tree Heads (STHs) (Section 3.5 of [RFC6962]) with other connecting clients in the hope that STHs will find their way to auditors and monitors.

HTTPS clients in a Trusted Auditor Relationship share SCTs and STHs with trusted auditors or monitors directly, with expectations of privacy sensitive data being handled according to whatever privacy policy is agreed on between client and trusted party.

This Internet-Draft will expire on January 8, 2016.

Table of Contents

1. Introduction

The purpose of the protocols in this document is to detect misbehavior by CT logs. In particular, CT logs can misbehave either by rewriting history or by presenting a "split view" of their operations, also known as a partitioning attack [THREAT-ANALYSIS]. CT provides mechanisms for detection of these misbehaviors, but only if the community dependent on the log knows what to do with them. In order for the community to effectively detect log misbehavior, it needs a well-defined way to "gossip" about the activity of the logs that makes use of the available mechanisms.

One of the major challenges of any gossip protocol is limiting damage to user privacy. The goal of CT gossip is to publish and distribute information about the logs and their operations, but not to leak any additional information about the operation of any of the other particpants. Privacy of consumers of log information (in particular, of web browsers and other TLS clients) should not be damaged by gossip.

This document presents three different, complementary mechanisms for non-log players in the CT ecosystem to exchange information about logs in a manner that preserves the privacy of the non-log players involved. They should provide protective benefits for the system as a whole even if their adoption is not universal.

2. Overview

Public append-only untrusted logs have to be monitored for consistency, i.e., that they should never rewrite history. Additionally, monitors and other log clients need to exchange information about monitored logs in order to be able to detect a partitioning attack.

A partitioning attack is when a log serves different views of the log to different clients. Each client would be able to verify the append-only nature of the log while in the extreme case being the only client seeing this particular view.

Gossiping about what's known about logs helps solve the problem of detecting malicious or compromised logs mounting such a partitioning attack. We want some side of the partitioned tree, and ideally both sides, to see the other side.

Disseminating known information about a log poses a potential threat to the privacy of end users. Some data of interest (e.g. SCTs) are linkable to specific log entries and thereby to specific sites, which makes them privacy-sensitive. Gossip about this data has to take privacy considerations into account in order not to leak associations between users of the log (e.g., web browsers) and certificate holders (e.g., web sites). Even sharing STHs (which do not link to specific log entries) can be problematic - user tracking by fingerprinting through rare STHs is one potential attack.

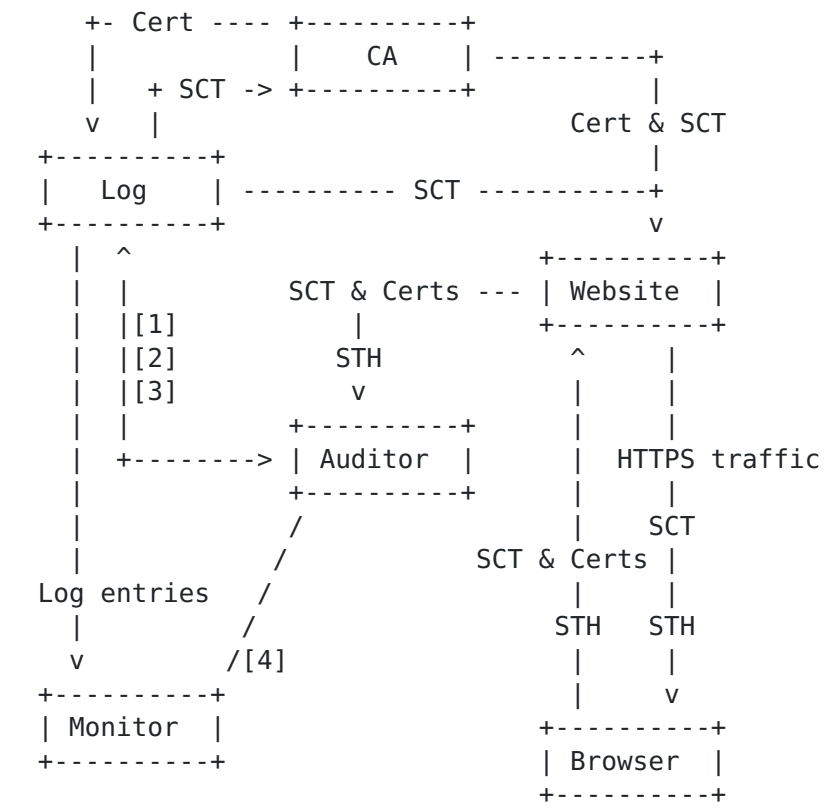
However, there is no loss in privacy if a client sends SCTs for a given site to the site corresponding to the SCT, because the site's access logs would already indicate that the client is accessing that site. In this way a site can accumulate records of SCTs that have been issued by various logs for that site, providing a consolidated repository of SCTs which can be queried by auditors.

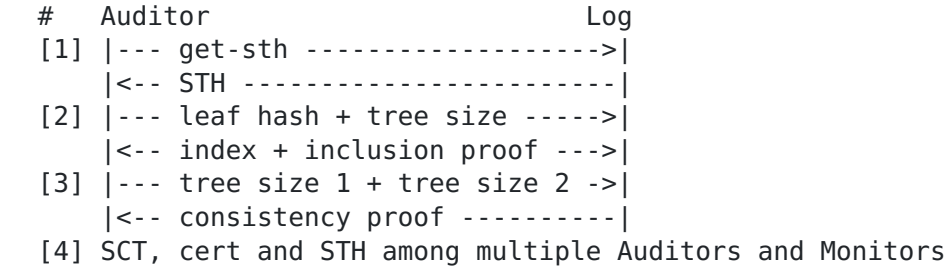
Sharing an STH is considered reasonably safe from a privacy perspective as long as the same STH is shared by a large number of other clients. This "safety in numbers" is achieved by requiring gossip only for STHs of a certain "freshness" and limiting the frequency by which logs can issue STHs.

3. Terminology and data flow

This document relies on terminology and data structures defined in [RFC6962], including STH, SCT, Version, LogID, SCT timestamp, CtExtensions, SCT signature, Merkle Tree Hash.

The following picture shows how certificates, SCTs and STHs flow through a CT system with SCT Feedback and STH Pollination. It does not show what goes in the Trusted Auditor Relationship stream.





4. Who gossips

- HTTPS clients and servers (SCT Feedback and STH Pollination)

- HTTPS servers and CT auditors (SCT Feedback)

- CT auditors and monitors (Trusted Auditor Relationship)

Additionally, some HTTPS clients may engage with an auditor who they trust with their privacy:

- HTTPS clients and CT auditors (Trusted Auditor Relationship)

5. What to gossip about and how

There are three separate gossip streams:

- SCT Feedback, transporting SCTs and certificate chains from HTTPS clients to CT auditors/monitors via HTTPS servers.
- STH Pollination, HTTPS clients and CT auditors/monitors using HTTPS servers as STH pools for exchanging STHs.
- Trusted Auditor Stream, HTTPS clients communicating directly with trusted CT auditors/monitors sharing SCTs, certificate chains and STHs.

5.1. SCT Feedback

The goal of SCT Feedback is for clients to share SCTs and certificate chains with CT auditors and monitors in a privacy-preserving manner.

HTTPS clients store SCTs and certificate chains they see and later send them to the originating HTTPS server by posting them to a .well-known URL. This is described in Section 5.1.1. Note that clients send the same SCTs and chains to servers multiple times with the assumption that a potential man-in-the-middle attack eventually will cease so that an honest server will receive collected malicious SCTs and certificate chains.

HTTPS servers store SCTs and certificate chains received from clients and later share them with CT auditors by either posting them or making them available on a .well-known URL. This is described in Section 5.1.2.

5.1.1. HTTPS client to server

An HTTPS client connects to an HTTPS server for a particular domain. The client receives a set of SCTs as part of the TLS handshake. The client MUST discard SCTs that are not signed by a known log and SHOULD store the remaining SCTs together with the corresponding certificate chain for later use in feedback.

When the client later reconnects to any HTTPS server for the same domain it again receives a set of SCTs. The client MUST add new SCTs from known logs to its store of SCTs for the server. The client MUST send to the server the ones in the store that are for that server and were not received from that server.

Note that the SCT store also contains SCTs received in certificates.

The client MUST NOT send the same set of SCTs to the same server more often than TBD. [benl: "sent to the server" only really counts if the server presented a valid SCT in the handshake and the certificate is known to be unrevoked (which will be hard for a MitM to sustain)] [TODO: expand on rate/resource limiting motivation]

An SCT MUST NOT be sent to any other HTTPS server than one serving the domain that the certificate signed by the SCT refers to. This would lead to two types of privacy leaks. First, the server recieving the SCT would learn about other sites visited by the HTTPS client. Secondly, auditors or monitors recieving SCTs from the HTTPS server would learn information about the other HTTPS servers visited by its clients.

If the HTTPS client has configuration options for not sending cookies to third parties, SCTs MUST be treated as cookies with respect to this setting.

SCTs and corresponding certificates are POSTed to the originating HTTPS server at the well-known URL:

https://<domain>/.well-known/ct/v1/sct-feedback

The data sent in the POST is defined in Section 5.1.3.

HTTPS servers perform a number of sanity checks on SCTs from clients before storing them:

1. if a bit-wise compare of an SCT plus chain matches a pair already in the store, this SCT and chain pair MAY be discarded

2. if the SCT can't be verified to be a valid SCT for the accompanying leaf cert, issued by a known log, the SCT SHOULD be discarded

3. if the leaf cert is not for a domain that the server is authoritative for, the SCT MUST be discarded

Check number 1 is for detecting duplicates. It's important to note that the check must be on pairs of SCT and chain in order to catch different chains accompanied by the same SCT. [XXX why is this important?]

Check number 2 is to prevent spamming attacks where an adversary can fill up the store prior to attacking a client, or a denial of service attack on the server's storage space.

Check number 3 is to help malfunctioning clients from leaking what sites they visit and additionally to prevent spamming attacks.

Note that an HTTPS server MAY perform a certificate chain validation on a submitted certificate chain, and if it matches a trust root configured on the server (but is otherwise unknown to the server), the HTTPS server MAY store the certificate chain and MAY choose to store any submitted SCTs even if they are unable to be verified. The risk of spamming and denial of service can be mitigated by configuring the server with all known acceptable certificates (or certificate hashes).

5.1.2. HTTPS server to auditors

HTTPS servers receiving SCTs from clients SHOULD share SCTs and certificate chains with CT auditors by either providing the well-known URL:

https://<domain>/.well-known/ct/v1/collected-sct-feedback

or by HTTPS POSTing them to a number of preconfigured auditors. This allows an HTTPS server to choose between an active push model or a passive pull model.

The data received in a GET of the well-known URL or sent in the POST is defined in Section 5.1.3.

HTTPS servers SHOULD share all SCTs and accompanying certificate chains they see that pass the checks in Section 5.1.1.

HTTPS servers MUST NOT share any other data that they may learn from the submission of SCT Feedback by HTTPS clients.

Auditors SHOULD provide the following URL accepting HTTPS POSTing of SCT feedback data:

https://<auditor>/ct/v1/sct-feedback

Auditors SHOULD regularly poll HTTPS servers at the well-known collected-sct-feedback URL. The frequency of the polling and how to determine which domains to poll is outside the scope of this document. However, the selection MUST NOT be influenced by potential HTTPS clients connecting directly to the auditor, as it would reveal private information provided by the clients.

5.1.3. SCT Feedback data format

The data shared between HTTPS clients and servers as well as between HTTPS servers and CT auditors/monitors is a JSON object [RFC7159] with the following content:

o sct_feedback: An array of objects consisting of

  * x509_chain: An array of base64-encoded X.509 certificates. The first element is the end-entity certificate, the second chains to the first and so on.

  * sct_data: An array of objects consisting of the base64 representation of the binary SCT data as defined in [RFC6962] Section 3.2.

The 'x509_chain' element MUST contain at the leaf certificate and the full chain to a known root.

5.2. STH pollination

The goal of sharing Signed Tree Heads (STHs) through pollination is to share STHs between HTTPS clients and CT auditors and monitors in a privacy-preserving manner.

HTTPS servers supporting the protocol act as STH pools. HTTPS clients and others in the possesion of STHs should pollinate STH pools by sending STHs to them, and retrieving new STHs to send to new servers. CT auditors and monitors should retrieve STHs from pools by downloading STHs from them.

STH Pollination is carried out by sending STHs to HTTPS servers supporting the protocol, and retrieving new STHs. In the case of HTTPS clients, STHs are sent in an already established TLS session. This makes it hard for an attacker to disrupt STH gossiping without also disturbing ordinary secure browsing (https://).

STHs are sent by POSTing them at the .well-known URL:

https://<domain>/.well-known/ct/v1/sth-pollination

The data sent in the POST is defined in Section 5.2.3.

The response contains zero or more STHs in the same format, described in Section 5.2.3.

An HTTPS client may acquire STHs by several methods:

- in replies to pollination POSTs;
- asking its supported logs for the current STH directly or indirectly;
- via some other (currently unspecified) mechanism.

HTTPS clients (who have STHs), CT auditors and monitors SHOULD pollinate STH pools with STHs. Which STHs to send and how often pollination should happen is regarded as policy and out of scope for this document with exception of certain privacy concerns.

An HTTPS client could be tracked by giving it a unique or rare STH. To address this concern, we place restrictions on different components of the system to ensure an STH will not be rare.

- Logs cannot issue STHs too frequently. This is restricted to 1 per hour.
- HTTPS clients silently ignore STHs which are not fresh.

An STH is considered fresh iff its timestamp is less than 14 days in the past. Given a maximum STH issuance rate of one per hour, an attacker has 336 unique STHs per log for tracking.

When multiplied by the number of logs that a client accepts STHs for, this number of unique STHs grow and the negative privacy implications grow with it. It's important that this is taken into account when logs are chosen for default settings in HTTPS clients.

[TBD urge HTTPS clients to store STHs retrieved in responses?]

[TBD share inclusion proofs and consistency proofs too?]

5.2.1. HTTPS client STH Fetching

An HTTPS client retrieves SCTs from an HTTPS server, and must obtain an inclusion proof to an STH in order to verify the promise made by the SCT. This retrieval mechanism reveals the client's browsing habits when the client requests the proof diretly from the log. To mitigate this risk, an HTTPS client MUST retrieve the proof in a manner that disguises the client from the log.

Additionally, for this inclusion proof to be acceptable to the client, the inclusion proof MUST reference a STH that is within the acceptable freshness interval.

Depending on the client's DNS provider, DNS may provide an appropriate intermediate layer that obfuscates the linkability between the user of the client and the request for inclusion (while at the same time providing a caching layer for oft-requested inclusion proofs.)

Also Tor.

5.2.2. Auditor and Monitor Action

Auditors and Monitors participate in STH pollination by retrieving STHs from HTTPS servers. They verify that the STH is valid by checking the signature, and requesting a consistency proof from the STH to the most recent STH.

After retrieving the consistency proof to the most recent STH, they SHOULD pollinate this new STH among participating HTTPS Servers. In this way, as STHs "age out" and are no longer fresh, their "lineage" continues to be tracked in the system.

5.2.3. STH Pollination data format

The data sent from HTTPS clients and CT monitors and auditors to HTTPS servers is a JSON object [RFC7159] with the following content:

- sths - an array of 0 or more fresh STH objects [XXX recently collected] from the log associated with log_id. Each of these objects consists of
  - sth_version: Version as defined in [RFC6962] Section 3.2, as a number. The version of the protocol to which the sth_gossip object conforms.
  - tree_size: The size of the tree, in entries, as a number.
  - timestamp: The timestamp of the STH as defined in [RFC6962] Section 3.2, as a number.
  - sha256_root_hash: The Merkle Tree Hash of the tree as defined in [RFC6962] Section 2.1, as a base64 encoded string.
  - tree_head_signature: A TreeHeadSignature as defined in [RFC6962] Section 3.5 for the above data, as a base64 encoded string.
  - log_id: LogID as defined in [RFC6962] Section 3.2, as a base64 encoded string.

[XXX An STH is considered recently collected iff TBD.]

5.3. Trusted Auditor Stream

HTTPS clients MAY send SCTs and cert chains, as well as STHs, directly to auditors. Note that there are privacy implications of doing so, outlined in Section 6.1.1 and Section 6.1.5.

The most natural trusted auditor arrangement arguably is a web browser that is "logged in to" a provider of various internet services. Another equivalent arrangement is a trusted party like a corporation which an employer is connected to through a VPN or by other similar means. A third might be individuals or smaller groups of people running their own services. In such a setting, retrieving STHs and inclusion proofs from that third party in order to validate SCTs could be considered reasonable from a privacy perspective. The HTTPS client does its own auditing and might additionally share SCTs and STHs with the trusted party to contribute to herd immunity. Here, the ordinary [RFC6962] protocol is sufficient for the client to do the auditing while SCT Feedback and STH Pollination can be used in whole or in parts for the gossip part.

Another well established trusted party arrangement on the internet today is the relation between internet users and their providers of DNS resolver services. DNS resolvers are typically provided by the internet service provider (ISP) used, which by the nature of name resolving already know a great deal about what sites their users visit. As mentioned in Section XXX, in order for HTTPS clients to be able to retrieve inclusion proofs for certificates in a privacy preserving manner, logs could expose a DNS interface in addition to the ordinary HTTPS interface. An informal writeup of such a protocol can be found at XXX.

5.3.1. Trusted Auditor data format

[TBD specify something here or leave this for others?]

6. Security considerations

6.1. Privacy considerations

The most sensitive relationships in the CT ecosystem are the relationships between HTTPS clients and HTTPS servers. Client-server relationships can be aggregated into a network graph with potentially serious implications for correlative de-anonymisation of clients and relationship-mapping or clustering of servers or of clients.

6.1.1. Privacy and SCTs

An SCT contains information that links it to a particular web site. Because the client-server relationship is sensitive, gossip between clients and servers about unrelated SCTs is risky. Therefore, a client with an SCT for a given server should transmit that information in only two channels: to a server associated with the SCT itself; and to a trusted CT auditor, if one exists.

6.1.2. Privacy in SCT Feedback

SCTs introduce yet another mechanism for HTTPS servers to store state on an HTTPS client, and potentially track users. HTTPS clients which allow users to clear history or cookies associated with an origin MUST clear stored SCTs associated with the origin as well.

Auditors should treat all SCTs as sensitive data. SCTs received directly from an HTTPS client are especially sensitive, because the auditor is a trusted by the client to not reveal their associations with servers. Auditors MUST NOT share such SCTs in any way, including sending them to an external log, without first mixing them with multiple other SCTs learned through submissions from multiple other clients. The details of mixing SCTs are TBD.

There is a possible fingerprinting attack where a log issues a unique SCT for targeted log client(s). A colluding log and HTTPS server operator could therefore be a threat to the privacy of an HTTPS client. Given all the other opportunities for HTTPS servers to fingerprint clients - TLS session tickets, HPKP and HSTS headers, HTTP Cookies, etc. - this is acceptable.

The fingerprinting attack described above could be avoided by requiring that logs i) MUST return the same SCT for a given cert chain ([RFC6962] Section 3) and ii) use a deterministic signature scheme when signing the SCT ([RFC6962] Section 2.1.4).

There is another similar fingerprinting attack where an HTTPS server tracks a client by using a variation of cert chains. The risk for this attack is accepted on the same grounds as the unique SCT attack described above. [XXX any mitigations possible here?]

6.1.3. Privacy for HTTPS clients requesting STHs

An HTTPS client that does not act as an auditor should only request an STH from a CT log that it accepts SCTs from. An HTTPS client should regularly request an STH from all logs it is willing to accept, even if it has seen no SCTs from that log.

6.1.4. Privacy in STH Pollination

An STH linked to an HTTPS client may indicate the following about that client:

- that the client gossips;
- that the client been using CT at least until the time that the timestamp and the tree size indicate;
- that the client is talking, possibly indirectly, to the log indicated by the tree hash;
- which software and software version is being used.

There is a possible fingerprinting attack where a log issues a unique STH for a targeted log auditor or HTTPS client. This is similar to the fingerprinting attack described in Section 6.1.2, but it is mitigated by the following factors:

- the relationship between auditors and logs is not sensitive in the way that the relationship between HTTPS clients and HTTPS servers is;
- because auditors regularly exchange STHs with each other, the re-appearance of a targeted STH from some auditor does not imply that the auditor was the original one targeted by the log;
- an HTTPS client's relationship to a log is not sensitive in the way that its relationship to an HTTPS server is. As long as the client does not query the log for more than individual STHs, the client should not leak anything else to the log itself. However, a log and an HTTPS server which are collaborating could use this technique to fingerprint a targeted HTTPS client.

Note that an HTTPS client in the configuration described in this document doesn't make direct use of the STH itself. Its fetching of the STH and reporting via STH Pollination provides a benefit to the CT ecosystem as a whole by providing oversight on logs, but the HTTPS client itself will not necessarily derive direct benefit.

6.1.5. Trusted Auditors for HTTPS Clients

Some HTTPS clients may choose to use a trusted auditor. This trust relationship leaks a certain amount of information from the client to the auditor. In particular, it is likely to identify the web sites that the client has visited to the auditor. Some clients may already share this information to a third party, for example, when using a

server to synchronize browser history across devices in a server-visible way, or when doing DNS lookups through a trusted DNS resolver. For clients with such a relationship already established, sending SCT Feedback to the same organization does not appear to leak any additional information to the trusted third party.

Clients who wish to contact an auditor without associating their identities with their SCT Feedback may wish to use an anonymizing network like Tor to submit SCT Feedback to the auditor. Auditors SHOULD accept SCT Feedback that arrives over such anonymizing networks.

Clients sending feedback to an auditor may prefer to reduce the temporal granularity of the history leakage to the auditor by caching and delaying their SCT Feedback reports. This strategy is only as effective as the granularity of the timestamps embedded in the SCTs and STHs.

6.1.6. HTTPS Clients as Auditors

Some HTTPS Clients may choose to act as Auditors themselves. A Client taking on this role needs to consider the following:

- an Auditing HTTPS Client potentially leaks their history to the logs that they query. Querying the log through a cache or a proxy with many other users may avoid this leakage, but may leak information to the cache or proxy, in the same way that an non-Auditing HTTPS Client leaks information to a trusted Auditor.

- an effective Auditor needs a strategy about what to do in the event that it discovers misbehavior from a log. Misbehavior from a log involves the log being unable to provide either (a) a consistency proof between two valid STHs or (b) an inclusion proof for a certificate to an STH any time after the log's MMD has elapsed from the issuance of the SCT. The log's inability to provide either proof will not be externally cryptographically-verifiable, as it may be indistinguishable from a network error.

7. IANA considerations

TBD

8. Contributors

The authors would like to thank the following contributors for valuable suggestions: Al Cutter, Ben Laurie, Benjamin Kaduk, Karen Seo, Magnus Ahltorp, Yan Zhu.

9. ChangeLog

9.1. Changes between -01 and -02

- STH Pollination defined.
- Trusted Auditor Relationship defined.
- Overview section rewritten.
- Data flow picture added.
- Section on privacy considerations expanded.

9.2. Changes between -00 and -01

- Add the SCT feedback mechanism: Clients send SCTs to originating web server which shares them with auditors.
- Stop assuming that clients see STHs.
- Don't use HTTP headers but instead .well-known URL's - avoid that battle.
- Stop referring to trans-gossip and trans-gossip-transport-https - too complicated.
- Remove all protocols but HTTPS in order to simplify - let's come back and add more later.
- Add more reasoning about privacy.
- Do specify data formats.

10. References

10.1. Normative References

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, June 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

10.2. Informative References

[THREAT-ANALYSIS]
Kent, S., "Threat Analysis for Certificate Transparency", 2015, <https://datatracker.ietf.org/doc/draft-ietf-trans-threat-analysis/>.

Authors' Addresses

Linus Nordberg
NORDUnet

Email: linus@nordu.net

Daniel Kahn Gillmor
ACLU

Email: dkg@fifthhorseman.net

Tom Ritter

Email: tom@ritter.vg