# Gossip Based Certificate Transparency

Victor Nguyen
University of Minnesota - Twin Cities
Minneapolis, MN United States of America
nguy2875@umn.edu

Thomas Bischof
University of Minnesota - Twin Cities
Minneapolis, MN United States of America
bisch112@umn.edu

Joel Nielsen
University of Minnesota - Twin Cities
Minneapolis, MN United States of America
niels641@umn.edu

Abraham Narvaez
University of Minnesota - Twin Cities
Minneapolis, MN United States of America
narva027@umn.edu

Raymond He
University of Minnesota - Twin Cities
Minneapolis, MN United States of America
hexxx863@umn.edu

## ABSTRACT

**Certificate Transparency is an additional mechanism that can be added to the current SSL certificate system to improve the reliability and user trust of certificates. Certificate Transparency has been implemented in some fashion in Google Chrome, Firefox, and Safari web browsers. Additionally, gossip protocols can be used with Certificate Transparency to help make the checking of the consistency between certificate logs more efficient. Using a combination of information about Certificate Transparency and gossip protocols, we are able to replicate a certificate log, certificate authority, domain address, and client in order to demonstrate an attack. The attack will exploit a potential weakness in the Certificate Transparency and gossip protocol combination to identify if a given user is gossiping. The gossiping user could then be fed a false certificate or blocked from sharing its gossip with other certificate logs.**

## Ⅰ. Introduction

Since the inception of TLS, public key infrastructure (PKI) and its cryptographic protocols has been an important factor in allowing users to exchange data over a secure connection. However, the digital certificate system has been prone to numerous attacks. These attacks not only target the TLS protocol, but also the trust model that the protocol depends on. This trust model relies on the trustworthiness of certificate authorities (CAs), who issue certificates to verified domains. These certificates are used by browsers to determine the authenticity of the domains before completing the connection. Because of the trust that browsers place in certificate authorities to provide valid certificates, fraudulent certificates issued by CAs can go undetected and lead to users connecting to malicious domains. This can lead to man-in-the-middle attacks, in which an entity, working with a CA, can use a certificate to intercept a user's traffic. An attempt to solve this vulnerability led to the creation and adoption of Certificate Transparency (CT). This provided additional measures to keep CAs accountable and make the certificate issuance process transparent by introducing certificate logs, monitors, and auditors. Working together, these provide users with viewable certificate logs that contain information about the issued certificates. Adhering to a very specific set of rules, these certificate logs are designed to allow users to check the validity of certificates. However, a novel attack against CT has been discovered. In the current implementation of CT, users request a view of the log. An attacker in control of the certificate log can control what the log looks like to users, so when requested, a log containing a fraudulent certificate can be sent to a user. This results in a split-world attack, in which the victim believes that the domain they are trying to connect to is safe because the certificate log includes the certificate that the domain provided. To combat this, gossip protocols were introduced. In this approach, peers in a network select partners at random and compare their views of the certificate log, using standard HTTPS packets to exchange data. The detection of inconsistencies in their views can lead to detection of a split-world attack. This approach is designed to prevent split-world attacks while also preserving the privacy of gossiping peers.

In this paper, we make the following contributions. First, we simulate the gossip message format of the protocol described in *Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs*. Then, we design and

construct a side channel attack to detect the presence of gossip messages and isolate gossiping and non-gossiping peers, eliminating some of the privacy-protecting benefits of the implemented gossip protocol and opening up the avenue to further attacks.

## ＩＩ. Certificate Authorities

Certificate authorities are the backbone of the digital certificate system that allows clients to connect securely to websites. As the third party that domains depend on to request certificates from and clients depend on for certificate validation, the importance of having a trustworthy certificate authority cannot be understated. Secure connections over the internet rely on a website presenting a certificate that contains a key, and the client verifying the validity of the certificate before connecting. To acquire a certificate, the website owner makes a request to a certificate authority, who verifies the server's identity, issues a certificate, and signs the certificate with its private key. When connecting to the domain, the client or browser will look at the certificate and verify that the certificate is verified by a trusted certificate authority and is not expired.

To determine which certificate authorities can be trusted, browsers such as Google Chrome operate programs that authorized certificate authorities join. To join, each CA must meet specific criteria in order to be considered, and once accepted, these CAs can issue certificates that the browsers trust. This trust is maintained by browsers using databases of approved CAs, which is a root store that contains a CA's root certificate. Major browsers such as Google Chrome and Firefox contain thousands of trusted CAs.

At the highest security level, certificate authorities have a root certificate that is the basis for all issued certificates. This root certificate is used to create intermediate CA (ICA) root certificates, and continue to branch down to end digital certificates. This is called a trust hierarchy. Typically, CAs will use these ICAs to issue certificates rather than the root certificates in order to protect the security of the root certificate. In addition, most CAs delegate the task of verifying a certificate requester's authenticity to a registration authority (RA).

While there is no technical requirement to become a certificate authority, publicly trusted certificate authorities typically participate in a CA/browser forum, which governs how CAs work with browsers and provides guidelines for handling certificate distribution. Violation of these guidelines or other rules can reduce the trust in a certificate authority and lead to revocation of their authority.

The issue with certificate authorities in the current SSL certificate system is that compromised certificate authorities can issue fraudulent certificates. In one instance, fraudulent certificates were issued by trusted ICAs in Turkey and France [1]. These trust vulnerabilities are especially useful for man-in-the-middle attacks in which an entity such as a government agency, working with a CA, can impersonate a website and intercept traffic.

## ＩＩＩ. Certificate Transparency

Certificate Transparency provides three additional components to the current SSL certificate system in order to increase the public's trust in the current system. The three components that Certificate Transparency adds are:

1. Certificate logs
2. Certificate monitors
3. Certificate authorities.

### Certificate Logs

Certificate logs are a service that record SSL certificates, and these logs maintain three important invariants. The first invariant is that the logs are append only, additional log entries can only be added and none can be deleted. The second invariant is that the logs use a special hashing technique called Merkle Tree Hashing to keep the logs safe from potential attackers. The final invariant is that the logs are available to the public to view and to check for the proper behaviors. These invariants make it very difficult for attackers to attempt to corrupt the logs and use them for evil.

### Certificate Monitors and Auditors

Certificate monitors are servers set up to check all certificate log servers in order to check for certificates that are illegitimate. Certificate monitors can be started by individuals, companies, or other institutions that want to participate in the maintaining of certificate logs. For example, Google runs a certificate monitor that issues certificates for all websites that Google operates. Monitors have the ability to tell if a certificate in a given log has qualities that make it invalid or dangerous to people it may issued too. The monitors are then able to alert the entity

running the certificate log about these potentially harmful certificates. A more detailed explanation for reporting certificate problems, “These error reports contain more detailed information than can be collected via the usage metrics … the reports contain the hostname of the website on which the error occurred, the full certificate chain, and whether the user chose to bypass the warning” [9]. Certificate auditors also provide checking mechanisms for certificate logs, though large logs such as Google’s logs combine the roles of auditing and monitoring.

Certificate Transparency can be implemented in multiple variations of the current SSL certificate system. Two of the potential implementations of Certificate Transparency are X.509v3 extension and the TLS extension.

### X.509v3 extension

In the case of the X.509v3 extension, a certificate authority will need to contact a certificate log server before issuing an SSL certificate. The certificate authority will validate the SSL certificate and return a signed certificate timestamp (SCT). A better explanation for a SCT is “SCTs are offered to clients with a certificate to indicate that the certificate has been or will shortly be publicly logged” [9]. Both of these objects will be then passed to the domain that requested them and then to the client of the domain. This process can be seen in Figure 1.

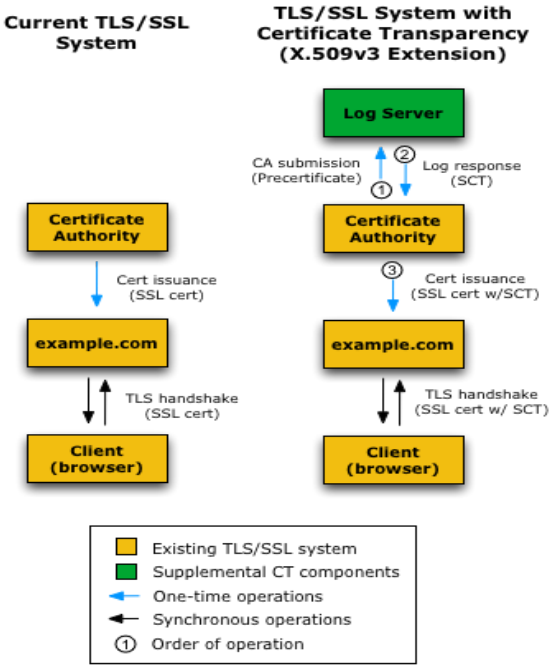


*Figure 1:* illustrates the differences between the current SSL System and then the X.509v3 extension added to it [9].

### TLS extension

In the case of the TLS extension, a domain receives an SSL certificate in the same way that it would in the normal SSL certificate system. However, instead of sending it to the client, it gets sent to the certificate log for validation. The certificate log checks the SSL certificate and sends a signed certificate timestamp (SCT) back to the domain if the SSL certificate was valid. The domain then sends both the objects to the client. This process can be seen in Figure 2.

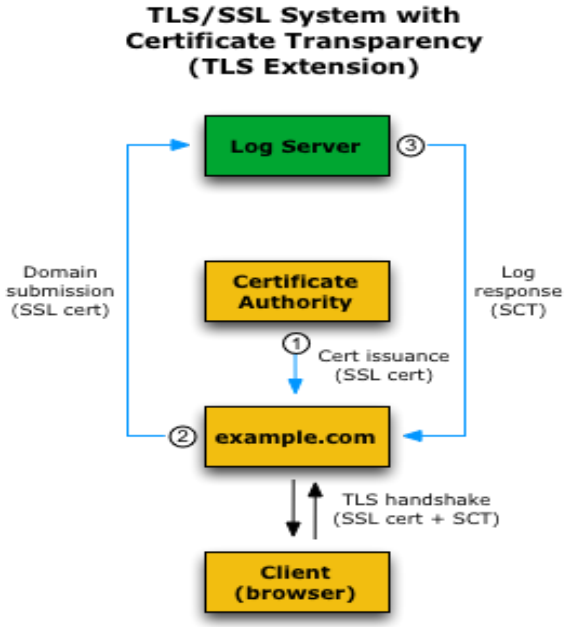


Figure 2: illustrates the TLS extension combined with the SSL certificate system [9].

## IV. Complications and Issues with Certificate Transparency

### Privacy

Certificate Transparency faces a number of complications. For example, the information in CT logs gives away information that users may want to remain private, such as listings of subdomains that reveal enterprise structure details and gives attackers a list of potential windows for attacking. Websites of large organizations with security-sensitive domain names that wish to redact

information from log entries for privacy reasons have had averse attitudes against the openness of Certificate Transparency logs, and solutions have been proposed, but none have yet been agreed upon and implemented [7], [2]. Because of these privacy concerns, some certificate authorities and certificate owners choose not to log their certificates [2].

Relatedly, Scheitl et al. found a correlation between DNS clients and internet scanners to which they concluded that Certificate Transparency logs are commonly used to find vulnerable targets for malicious connections and subdomain enumeration [4]. Their analysis consisted of leaking their own subdomains, analyzing the queries and the tendencies of the sources of the queries, and observing that the queries were done in batch jobs that scanned an unusually high number of ports on their machines [4].

In addition to the privacy of enterprise organizations being of concern, the privacy of users is also an issue in the current Certificate Transparency implementation. When a log misbehaves and the certificate in question is not publicly accessible, then the signed certificate timestamp must be shared with a verifier to report the misbehavior so that the issue can be investigated. The verifying entity is often a browser vendor, so the vendor that the signed certificate timestamp is shared with acquires the user's browsing activity [3]. Eskandarian et al. also note that this is, in fact, the principle mechanism used in auditing non-public sites visited by users, which makes this an especially exigent privacy concern [3].

#### Usability

The usability of Certificate Transparency is also a concern. Certificate Transparency relies on widespread adoption and cooperation between the large technology ecosystems. Currently, users are able to click through warnings or switch to browsers that do not implement Certificate Transparency, and studies by Google show that users typically bypass warnings in the aforementioned ways when they encounter Certificate Transparency errors [2]. This removes the incentive for site owners to comply with Certificate Transparency policies.

#### Auditing

The Audit function of the Certificate Transparency system does not fully detect certificate mis-issuance. Log misbehavior includes presenting different views of logs to different clients, editing or removing information from a log, not delivering a certificate by the maximum merge delay, or producing STHs more often than the STH Frequency Count log parameter. As of now, there is no standard method of detecting the first of the four misbehaviors, and this is a subject which our paper addresses [8].

#### Trust

Certificate Transparency also centralizes trust to certificate authorities. For instance, a malicious certificate authority may obtain multiple CA certificates from multiple parent CAs. The information submitted to the parent CAs will be different, except the public key and subject name. If malicious activity is detected by the malicious CA and its certificates are revoked, it can still have the certificate validated by the other SCTs provided by other CAs. This is illustrated in figure three. The fix appears to be relatively easy, but is not standardized [8].

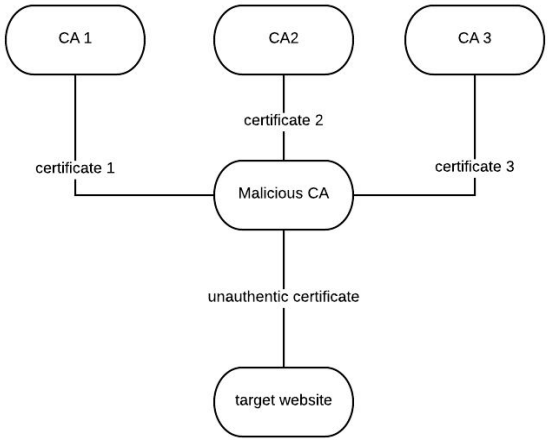


*Figure 3:* Illustrates multiple validation chain vulnerability

## Ⅴ. Gossip Protocols

Gossip protocols provide an efficient manner of transmitting information from one node within a network to another node. The use of gossip protocols can be seen in a variety of different areas such as detection of inconsistencies, distributed search, and autonomic self-management. A basic implementation of a gossip protocol generally consists of three different stages:

1. Peer-selection
2. Data Exchange/Comparison
3. Data Processing

*Figure 4:* Illustrates the process of random peer-selection.

**Peer-selection**

Peer-selection is the process in which a node picks a peer to exchange information with. The success of a gossip protocol often relies on an efficient method of biasing peer communication to control congestion within the network. To ensure that all nodes within the system hold the same information it generally takes O(log(N)) steps, where N is the number of nodes within the network. As shown in *figure 4* peer-selection attempts to have each node communicate with every other node within the network to ensure that information between the nodes is consistent. The second process that is performed is the exchange of information between the two peer nodes. This step compares data from both nodes to determine whether a node has had its data tampered with. Our paper specifically analyzes this step as a way to verify that certificate logs are legitimate and determine whether a log has been tampered with. The third process is determined by what the protocol wants to do with the data that it has been given.

**STH-Only Gossiping**

While the general idea of a gossip protocol is fairly basic, gossip protocols can vary depending on the rules that are built into the algorithm. In the research paper *Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs* there are references to two specific gossip protocol algorithms relating to CT; the first protocol is the STH-Only Gossiping and the second protocol is the STH-and-Consistency-Proof Gossiping. The STH-Only Gossiping protocol focuses on the main idea that [1] “any inconsistency that can be detected with a STH of a certain tree size can also be detected with a STH of larger tree size--provided that the consistency between these two STHs has been proven”. This theory allows clients and servers to only keep the STH with the largest tree size and still be able to detect an inconsistency between the logs. The way this protocol is described to be implemented within [1] is that when a client receives a STH from a log, they store it and send it in all their gossip messages. By sending the STH with all their gossip messages the peer will be able to compare the sent STH with the current STH that is stored, checking to see if any inconsistencies are present. The peer will only contact the log for a consistency proof if the STH that it receives is has a larger tree size than the current STH tree size.

**STH-and-Consistency-Proof Gossiping**

The STH-and-Consistency-Proof Gossiping has the same security properties as the STH-Only Gossiping protocol while also gossiping consistency proofs together with the STHs. By gossiping the consistency proofs with the STH it minimizes the ability of an adversary to infer who is gossiping and who is not [1]. It is difficult for an adversary to infer who is gossiping and who is not, because gossip messages are sent through HTTPS headers that are encrypted. However, it is still possible for an adversary to determine whether a client is gossiping or not by listening to the clients network traffic. Once an adversary is able to determine whether a client is gossiping there is a possibility for an attack to happen.

**SCT Feedback Gossiping**

In addition to the other gossiping protocols mentioned, [10] SCT Feedback is another protocol in which clients share signed certificate timestamps (SCTs) and certificate chains from HTTPS clients to CT auditors via HTTPS servers. Whenever a client connects to a server, the client receives a set of SCTs as part of the TLS handshake and checks SCTs against a log known to the client. The client then verifies the set of SCTs and [10] "stores the remaining SCTs together with a locally constructed certificate chain which is trusted in an sct_feedback object or equivalent data structure for later use in SCT feedback". This data is then sent back to the server in a POST request. [10] The server should reply with HTTP 200 response code and an empty body if it processed the request.

## Ⅵ. Methodology of Replication

**CT Ecosystem**

In order to replicate the Certificate Transparency ecosystem, we had to implement a few key components. Firstly, we needed a Certificate Authority so that we can

create certificates and experiment with issuing fraudulent certificates. To do this, we followed a guide that describes how to implement a CA using openssl commands[1].

The next step was to create a CT log that we could use to get SCTs and STHs in order to properly gossip. Implementing our own log was a daunting task, and even using open-source code to get a log running was more difficult than anticipated. The code implementing logs and auditing is extremely complex and robust, so we were only able to use Google's Certificate Transparency tools in Python for monitoring logs. Therefore, we looked into using existing logs and eventually got our newly created CA's root certificate onto one of Google's test logs.

### Client and server

After this, we needed a server and client that simulate a normal user browsing a website. We used our CA to sign a certificate for a server and received a certificate and SCT from the Testtube log. We set up a simple NodeJS server on a local machine that emulates a pollination server described by Nordberg et al. [10]. The server contains various pages that are made to try to simulate what a normal browsing experience would be for a user. There are pages of various sizes, and some of them contain asynchronous AJAX requests to load more content. The server contains a web-page that accepts POST requests that contain SCT feedback in the format specified by Nordeberg et al. This includes all of the protocols described such as discarding repeat SCTs, only accepting SCTs from clients browsing that specific domain, and, most importantly, the sending of a 200 response code with an empty body when the server successfully receives an SCT feedback request (see attack section). The client is a simple Python program that uses the requests library to connect to the various web-pages in a random order. According to protocol, it will gossip the SCT at a random point in the already set-up TLS session with the server.

## ⅤⅠⅠ. Side Channel Attacks

### Description

A system that is vulnerable to side channel attacks is a system that leaks information via some weakness in the implementation of the system rather than the implementation of the algorithm itself. An example of this would be to analyze the time it takes for a login page to validate a password. The timing of how long this takes can leak information about how close the input password is to the real password. The types of side channel attacks focused on in this paper are size and timing side channel attacks. More specifically, we focus on side channel attacks which can be used to infer information about HTTPS/TLS traffic without breaking the encryption.

### Relevant Attacks

Although HTTPS strongly encrypts traffic, certain details such as the source and destination IP addresses, size of messages, and the timing of requests can be observed by anyone. Chen et al. discusses how many pieces of sensitive information can be inferred by analyzing size and timing of HTTPS traffic [11]. For example, if a user is typing something in a search box that gives search suggestions, an attacker can look at the size of the suggested searches that are sent by the server in order to infer what letter the user typed.

There are a few facts about how the internet is structured that makes these types of attacks possible. Firstly, HTTPS does not encrypt the destination of requests, so an attacker can gain this information trivially. In addition, an attacker can request a web-page themselves and see the size and timing of the components loaded by that page. This information can be compared to intercepted traffic to verify that someone is visiting a certain page. For example, an attacker could see how many additional components were requested by the web-page after the page is loaded, then use this information to identify which page was being accessed by the user. Finally, there is no way to keep the size of packets a secret, so attackers can gain even more information about the web-page itself and which components it is loading. These factors contribute to an environment in which carefully planned side-channel attacks are very possible.

## ⅤⅠⅠⅠ. Details of Attack

### Rationale

Our attack is based off of a vulnerability of the protocol described by Nordberg et al[10]. Specifically, the paper says that "[t]he HTTPS server SHOULD respond with an HTTP 200 response code and an empty body if it was able to process the request" where the request refers to the client

[1] The online guide we followed to implement our own CA: https://deliciousbrains.com/ssl-certificate-authority-for-local-https-development/

sending SCT feedback to the server's dedicated POST page. The rationale behind the attack is that this gossip feedback into empty server body should be a very unique set of packet exchanges. In other words, it will be rare for a client to send a medium sized packet that immediately results in a very small server response.

**Implementation**

The first challenge to implementing the attack is that it will depend on the size of the headers used by the server. If a server includes many fields in its HTTPS headers, then this may make the packet size larger than one that doesn't. HTTPS over TLS encrypts the entire packet including the header, so it isn't possible to look at each packet's header size. However, one can get a good estimate simply by accessing the site themselves and viewing the difference between the length of the content received and the entire packet size. This approximate header size is the first piece of information used in our attack. The next step of our attack is to actually analyze the victim's traffic. We implemented this using a packet analyzer called Wireshark. We tuned Wireshark to only listen to traffic that is coming from our server. In an attack on a normal website, Wireshark would need to be tuned to the specific IP address or addresses associated with the website. In addition, the packets can be filtered by source IP address if an attacker wishes to single out one user's traffic to the website. The result will be a record of all of the requests and responses that are made by the client to the server. In our attack we analyzed all of the responses that the server sent to the client. As discussed above, the client is automated to visit four pages on the website in a random order. Wireshark captured the sizes and timing of all four of those requests and responses (see Figure 5). This figure shows the client hello, server hello, various key exchanges, and

| Protocol | Length | Info |
|---|---|---|
| TCP | 76 | 35228 → 9007 [SYN] Seq=0 Win=43690 Len= |
| TCP | 76 | 9007 → 35228 [SYN, ACK] Seq=0 Ack=1 Win= |
| TCP | 68 | 35228 → 9007 [ACK] Seq=1 Ack=1 Win=4377 |
| TLSv1.2 | 331 | Client Hello |
| TCP | 68 | 9007 → 35228 [ACK] Seq=1 Ack=264 Win=44 |
| TLSv1.2 | 1449 | Server Hello, Certificate, Server Key E |
| TCP | 68 | 35228 → 9007 [ACK] Seq=264 Ack=1382 Win= |
| TLSv1.2 | 194 | Client Key Exchange, Change Cipher Spec, |
| TLSv1.2 | 342 | New Session Ticket, Change Cipher Spec, |
| TLSv1.2 | 305 | Application Data |
| TLSv1.2 | 332 | Application Data, Application Data |
| TCP | 68 | 35228 → 9007 [FIN, ACK] Seq=627 Ack=192 |
| TCP | 68 | 9007 → 35228 [FIN, ACK] Seq=1920 Ack=62 |
| TCP | 68 | 35228 → 9007 [ACK] Seq=628 Ack=1921 Win= |
| TCP | 76 | 35230 → 9007 [SYN] Seq=0 Win=43690 Len= |
| TCP | 76 | 9007 → 35230 [SYN, ACK] Seq=0 Ack=1 Win= |

*Figure 5:* Displays an example of a captured gossip packet

the application data. Everything except the packets labelled "application data" can be ignored since this is where the server response is sent. While it isn't shown in the figure, the first application data packet is sent by the client which results in an immediate response by the server. Importantly, the content length of the second packet is quite small at 332 bytes (this is the gossip packet). In fact, the sizes of the other 3 pages were 431, 986, and 3015 bytes with the 431 byte page being a completely empty web-page. This demonstrates that the smallest possible sized page that can be sent is still bigger than the gossip response from the server meaning that an attacker can look for packets of this size to determine whether a client is gossiping or not.
To test the rarity of these sizes of packets, we analyzed our own web traffic as we visited common sites such as Google, Facebook, Youtube, and Wikipedia for 30 minutes without using any gossiping. We found that contrary to our above claim, packets of sizes as small as the one above are relatively common. However, this is where analysis of the timing and order of packets can be used. In normal browsing, when packets of this smaller size are sent, it can usually be assumed that they are images or other website enhancements rather than actual web-pages. This means they are likely housed on the website's server and don't require a request to obtain. In our attack, the 332 byte packet was sent by the server immediately after the client made a request. We never observed any such similar request/response sequences in our normal browsing. Specifically, we never saw a request made by our web browser that resulted in an immediate response from the server with content between 300 and 400 bytes.

## IX. Results and Analysis

**Consequences**

The above attack simply demonstrates that an attacker can use packet size and timings to determine whether or not a client is gossiping or not. This itself is a breach of protocol that violates the intent of Nordbert et al. that no one should be able to determine whether a particular client is gossiping. However, this can be a launch point for a variety of other attacks not explored in this paper. If the above attack was successful, then the attacker now knows that a client is gossiping, and knows the relative size of the client's gossip packets which can be determined by looking at the request the client made before receiving the small server response. There are a few things the attacker can do with this

information. Firstly, if an attacker can determine that a client is not gossiping, then they know that client is particularly vulnerable to attacks that would be affected by gossiping. On the other hand, if an attacker detects that a certain client is gossiping, they can try to block all future gossip packets based on the information they have about the client's gossip packet length and timing. In either situation, an attacker can isolate a particular client in order to carry out attacks against them that could be mitigated by gossiping.

**Analyzing the attack**

In our implementation of the CT gossiping ecosystem, our attack is successfully able to determine whether or not a particular client is gossiping with a server. The attack is successful due to a vulnerability in the protocol described for gossiping servers and clients, and our attack would be less effective if this protocol was changed. In particular, our attack relies heavily on the fact that the server response to gossiping will be smaller than most other response traffic. If the server were to randomize the response size, or make the response size similar to other web-page sizes on their server, then this attack would be significantly more difficult to carry out. In addition, the attack we implemented only provides the attacker with knowledge that a client is gossiping. Although this is a breach of privacy, it can't harm the client without the attacker performing a more complex attack against the CT ecosystem itself.
Although this attack was performed in the context of the CT ecosystem, similar size-base side-channel attacks could be used against gossip protocols in other areas.

## X. Applications to Current and Future Work

There is still some work to be done in the future to improve CT. As mentioned in section IV above, Certificate Transparency and gossip protocols do have some privacy problems. Privacy in the modern day is more important to users than ever before. Certificate Transparency can cause privacy issues for both individuals and businesses because of the need to send certificates that are misbehaving to a log monitor or auditor for investigation. The quote, "[t]he problem with this approach is that it violates user privacy: the browser must send the offending SCT to the verifier, thereby revealing the user's browsing behavior to the verifier" shows the difficulty of maintaining the user's privacy while keeping the Certificate Transparency system working correctly [3]. A user's browsing habits are not something that should be sacrificed in order to maintain the Certificate Transparency system. Certificate Transparency will also cause problems with businesses, "[b]ecause domain names are publicly available in the CT log, logging certificates will reveal the servers' private domains" [3]. This will cause businesses to need to enable additional security measures to protect their private domains from attackers because otherwise unknown domains are now available to attack. Certificate Transparency's future implementations will need to address these privacy concerns or convince users that the additional privacy loss will be a lower cost than the benefits gained from Certificate Transparency.

Another problem that needs to be addressed is with the public key infrastructure. As described earlier within the paper the digital certificate system relies on trust models formed by certificate authorities. Certificate authorities have the important duty of verifying that an owner of a certificate is who they say they are. The certificate authority also [5] "holds the duty of terminating the validity of a certificate prior to its preconfigured expiration date upon becoming aware of certain circumstances, e.g., mistaken issuance site or CA compromise, affiliation change, a superseding certificate, or cessation of the holder's operations". The revocation process is done by either the issuing CA or online certificate status checking protocol (OCSP) responders. However, according to [5] during practice some CA certificates do not include any revocation information, and when OCSP responders are specified, they are often unresponsive. This allows for a multitude of untrusted certificates being specified as trusted certificates by a trusted CA.

One way in which gossip protocols can be applied is in blockchain technology. Blockchain technology is a decentralized way of keeping track of transactions across a network. Blockchain is composed of a series of blocks that are cryptographically hashed of the previous block, a timestamp, and a transaction date. Many common implementations of blockchain are based on Proof of Work (PoW) which relies on a peer-to-peer network following a protocol to implement node-to-node communication and validation and Byzantine Fault Tolerance (BFT) which focuses on cooperation between participants. A major problem with the PoW approach as stated in the paper [6] is "that the energy per transaction is enormous, the transaction rate is very low, and the latency is very high." A major problem with the BFT approach is that it is a closed membership system. The paper [6] proposes that by incorporating gossip protocols in the implementation of

blockchain that it is possible to “combine low energy with relatively open membership”. Gossip protocols have the potential to lower the transaction cost and improve the security of blockchains by providing a system to validate previous transactions.

## Ⅺ. Conclusion

The current approach to solving CT problems using gossip protocols still suffers from privacy concerns. In this paper, we replicated a gossip protocol implementation and constructed a side channel attack that detected the presence of gossip messages in order to isolate non-gossiping peers from gossiping peers. This attack is very simple to implement. We evaluated this attack using a simulation of the CT ecosystem to show that it is possible to detect gossip in HTTPS packets and carry out further attacks using the gathered information. Our research shows that because it is still relatively easy to carry out attacks with gossip protocols in place, there is still work to be done to improve gossip protocols and protect privacy.

## REFERENCES

[1] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie and E. Messeri, "Efficient gossip protocols for verifying the consistency of Certificate logs," *2015 IEEE Conference on Communications and Network Security (CNS)*, Florence, 2015, pp. 415-423.

[2] E. Stark, R. Sleevi, R. Muminovi´c, D. O'Brien, E. Messeri, A. P. Felt, B. McMillion, and P. Tabriz, “*Does Certificate Transparency break the web? measuring adoption and error rate*,” in *40th IEEE Symposium on Security and Privacy (SP)* Los Alamitos, CA, 2019, pp. 463-478.

[3] S.Eskandarian, E.Messeri, J.Bonneau, and D.Boneh, “Certificate Transparency with privacy,” *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 329–344, 2017.

[4] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch. 2018. *The Rise of Certificate Transparency and Its Implications on the Internet Ecosystem*. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. ACM, New York ,NY, USA,343-349. DOI: https://doi.org/10.1145/3278532.3278562

[5] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements," *2013 IEEE Symposium on Security and Privacy*, Berkeley, CA, 2013, pp. 511-525.

[6] Robbert van Renesse, "A Blockchain Based on Gossip? - a Position Paper", *Cornell University*, 2016, [online] Available: https://www.zurich.ibm.com/dccl/papers/renesse_dccl.pdf.

[7] R. Stradling “Certificate Transparency: Domain Label Redaction,” January 2017, https://tools.ietf.org/html/draft-strad-trans-redaction-01.

[8] S. Kent, “Attack and Threat Model for Certificate Transparency draft-ietf-trans-threat-analysis-16”, *Public Notary Transparency*, 2018, https://tools.ietf.org/html/draft-ietf-trans-threat-analysis-16

[9] Anon. How Certificate Transparency Works. Retrieved May 5, 2019 from http://www.certificate-transparency.org/how-ct-works.

[10] L. Nordberg, D. Gillmor, and T. Ritter, “Gossiping in CT,” January 2018, https://tools.ietf.org/html/draft-ietf-trans-gossip-05.

[11] S. Chen, R. Wang, X. Wang and K. Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," *2010 IEEE Symposium on Security and Privacy*, Berkeley/Oakland, CA, 2010, pp. 191-206. doi: 10.1109/SP.2010.20