Quantitative Verification of Certificate Transparency Gossip Protocols

Michael Oxford, David Parker, Mark Ryan
School of Computer Science, University of Birmingham, UK
{mco120, d.a.parker, m.d.ryan} [at] cs.bham.ac.uk

Abstract—**Certificate transparency is a promising log-based system designed to audit internet certificates publicly and is currently supported by Google Chrome. However, it is potentially vulnerable to split-world attacks, where certain users are directed to a fake version of the log. So, to ensure that users are seeing the same version of a log, gossip protocols have been designed in which users share data sourced from the log. In this paper, we propose a new way of evaluating these protocols using probabilistic model checking, a technique for formally verifying quantitative properties of computer systems. We describe our approach to modelling and verifying the protocols, including a novel approach to determine worst-case model parameters. We analyse several aspects of the protocols, including the success rate of detecting inconsistencies in gossiped data and the efficiency in terms of bandwidth, comparing different protocol variants and also our own proposals to improve protocol performance.**

I. Introduction

Public key infrastructures are a crucial ingredient of secure communication and are of growing importance as we see ever increasing numbers of network-enabled devices and cloud-based services. In the past, there have been many security incidents as a result of weaknesses in public key infrastructures [22], [23] and this has motivated security researchers to find alternatives that do not rely on the trust of certificate authorities to distribute certificates correctly. Out of all previously proposed solutions [6], [19], [8], *certificate transparency* (CT) has been the most promising [15].

CT uses an append-only log, based a Merkle hash tree data structure, that stores certificates in such a way that it can produce two types of proofs, in the form of a list of hashes, upon request: an audit proof is used to show that a certificate has been properly added to the log, and a consistency proof shows how a previous snapshot of the log can be extended to the current one. CT is currently part of the Google Chrome web browser and is in active development.

An open problem is how to ensure that the view of a log used for CT is kept consistent for each internet user. A powerful adversary with an abundance of skill and resources at its disposal, such as a totalitarian government, could issue fake certificates to perform man-in-the-middle attacks on a large population and have them contact a forked version of the log that they control. This is known as a *split-world attack*. Without a way for the targeted users to share their view of the log with everyone else, they will never know whether they are the victims of such attacks.

In an attempt to prevent this, *gossip protocols* have been designed [4], [18] which make users distribute data in the form of signed tree heads, all sourced from a public CT log. However, as yet, there is insufficient analysis of their effectiveness to justify their deployment. For example, while large-scale realistic simulations have been used to measure the connection overhead and data spread produced by some of these protocols (see [4]), there is no rigorous analysis of how reliable they are in detecting split-world attacks; this is a hard problem since the detection rate depends on a number of factors such as how users browse the internet and the information they have cached from a log.

To overcome these issues, in this paper, we apply formal verification to CT gossiping. In particular, we use *probabilistic model checking*, a verification technique designed for analysing systems that exhibit stochastic behaviour. In the context of CT gossip protocols, stochastic modelling is needed to capture uncertainty about the way that traffic flows through a network, which has a significant effect on the effectiveness of the protocols. By exhaustively exploring the possible ways that gossiping nodes can behave and the possible configurations of the network that can arise, and then numerically solving the resulting model, model checking can provide precise results for quantitative system properties such as the probability of an attack being detected or the expected time taken for information to propagate across the network. We use the PRISM tool [14] to model CT gossip protocols as discrete time Markov chains, and then verify the protocols and study the trade-offs between such quantitative properties.

Our main objective is to demonstrate how we can evaluate the effectiveness of CT gossip protocols via model checking, using the Chuat et al. [4] protocols as a case study. We describe how to model the protocols and capture the quantitative properties of interest in the temporal logic used by PRISM. We also describe a novel combination of probabilistic model checking with sequential model-based global optimisation, which determines appropriate worst-case parameters for our models. By studying a variety of quantitative properties over several different protocol scenarios, we show the benefits and drawbacks of deploying these protocols, comparing them in regards to the security they provide for users and their efficiency. Additionally, we suggest how the original protocol designs could be improved by having servers gossip with each other and justify our claims using the techniques presented in this paper.

978-1-7281-4760-4/20/$31.00 ©2020 IEEE

II. Related Work

CT gossip protocols other than the ones by Chuat et al. [4] have been considered, such as those by Nordberg et al. [18]. These differ in that they rely on a dedicated auditor to collect information from across the internet and contact the log for proofs on a regular basis.

There is also significant interest in gossip being used in transparency systems to verify consistency. As part of CONIKs [17], which is designed to publicly audit encryption keys used for peer-to-peer communication systems, Melara et al. mention how gossip protocols can be used by auditors (the identity providers that run CONIKs servers) to detect non-equivocation of a log by sharing snapshots of it. Building upon CONIKs, Etemad and Küpçü introduce KAS (key authentication service) which they show to be more efficient in regards to storage and computation while still using gossip [5].

Formal verification of protocols in general is a broad field, but our focus here is specifically on quantitative or probabilistic approaches. Probabilistic model checking has been used in the past to analyse the effectiveness of gossip protocols in non-security settings. Fehnker and Gao [7] analysed the performance of gossiping and flooding protocols in a small network using model checking while considering cases where network collisions and data loss occurs, combining this with large-scale simulations in order to see if the results were consistent for more realistic network models. Kwiatkowska et al [13] provided quantitative analysis of a gossip protocol that uses random peer sampling in very small networks. It included a scheduler which determined the order in which node executes the protocol, also keeping track of the nodes which have already exchanged data. Unlike the models found in previous work, the models used for this paper are based on a client-server network model where clients are unable to communicate with each other and must gossip information through servers.

More recently, Webster et al. [21] applied verification via PRISM to analyse the Firefly-Gossip (FiGo) algorithm used in IoT devices to communicate with nearby devices and synchronise their internal clocks. They discussed the abstraction process of the algorithm and design assumptions to significantly reduce the size of the PRISM model e.g. the speed the clock 'ticks' are kept constant for each node due to the homogeneity of the hardware. By taking account into the 'clock drift' phenomena caused by the temperature of the node's hardware, they were able to find the steady-state probability of the clocks being synchronised at varying levels of temperature. However, at most four sensor nodes are present in the models and modelling for larger networks still presents a challenge due to the state space explosion problem.

III. Background

A. Gossiping for Certificate Transparency

The protocols of Chuat et al. [4] make clients and servers exchange log-sourced data which we call gossip messages, with servers acting as proxies for clients to talk to each other across different areas of the internet. Signed tree heads (STHs), the main pieces of data that are gossiped in the protocol, contain the unique object identifier of the log they are generated from, the root hash of the log's Merkle tree, the corresponding tree size when the hash was generated, a timestamp from when it was created and a public signature.

We briefly explain how the protocol works. Firstly, after a client connects to an HTTPS-enabled server and finishes the negotiation phase of the TLS handshake, it calls `getClientMessage()` to generate the gossip message m_1 which is piggy-backed on an HTTPS request. The server receives m_1 and checks to see if it is valid, meaning that all the STHs included in the message are correctly signed by a known log and contain the root hash of a Merkle tree with at least one certificate appended. Then, the server replies in kind by calling `getServerMessage()` to generate the gossip message m_2 which is sent back to the client using an HTTPS response. With data from both parties now successfully exchanged, after performing the necessary consistency checks, both the client and server must refresh their knowledge of the log if the messages they retrieved have newer information. Since the gossiping was done via HTTPS, no-one can know whether any gossip happened without breaking the encryption used.

The two variants of this protocol that were given by Chuat et al. are called *STH-Only* gossiping, where both m_1 and m_2 consist of only one STH, and *STH and consistency proof* gossiping, where messages contain a pair of STHs with a consistency proof between their respective tree sizes; for the rest of the paper, we will refer to the latter version as *STH-and-proof*. The idea behind gossiping proofs as well as messages is to reduce the amount of connections an entity needs to make to a public log when requesting proofs as part of the protocol execution. The updating procedure for the client is very similar in both versions: using the server's message, it requests a consistency proof from the log whenever the tree size in m_2 is distinct from what the client already knows, updating itself when necessary using the contents of m_2. Next, if the server's signed certificate timestamp (SCT), a 'promise' from the log to append the corresponding certificate sent to the client beforehand through the TLS handshake, has not been audited yet, then the client will request both inclusion and consistency proofs from the log to check if the certificate exists in the log and updates its local state by retrieving the latest STH from the log.

Whenever someone receives an STH either through gossiping or by directly contacting the log, the protocol uses the `checkSTH()` method to check for consistency between multiple STHs when they are passed as parameters. To explain how this method works, suppose that we have two STHs s_a and s_b sourced from the same log, which are both associated with tree sizes $t_a, t_b \in \mathbb{N}$ respectively. `checkSTH(`s_a, s_b`)` will verify two things: (i) If $t_a = t_b$, check the root hashes of s_a and s_b are the same; (ii) If the timestamp of s_a is older then the timestamp of s_b, check that $t_a \leq t_b$. If either of these conditions are not met, then the protocol treats this as a security incident and the log cannot be trusted. Otherwise, the protocol can be executed normally.

When a node does detect log inconsistency, either by `checkSTH()` failing or the log providing nothing when given a proof request, the node will start to propagate a warning message that encapsulates the issue that was discovered instead of STHs or proofs. A recipient of the warning message will pass it onto a log monitor for them to investigate further and determine if the log is misbehaving. The updating procedures for the server in the two versions work very differently. In STH-Only gossiping, the server is required to only store one STH, whereas in STH-and-proof the server will record the messages it receives using a 'map' and will gossip them depending on the message it receives from clients.

B. Probabilistic Model Checking

Probabilistic model checking is a formal verification technique that has been used to analyse computer and control systems and many other phenomena that exhibit random behaviour [12]. By mathematically describing every possible type of behaviour in an exhaustive manner, model checking gives us a way to capture the possible states a system can be in and how it may evolve between states. In this paper we use discrete time Markov chains (DTMCs), where a transition between abstract states occur with a prescribed probability.

In order to analyse or verify system properties, we specify them formally using temporal logic; for our purposes we use probabilistic computational tree logic (PCTL) [9], extended with operators to reason about costs and rewards [12]. For example, the PCTL formula $\mathbf{P}_{=?}[\mathbf{F}^{\leq t}$ "attack successful" $]$ represents the probability of a system being attacked successfully within $t \in \mathbb{N}$ time steps. We also augment DTMCs with reward structures that assign numerical values to states or transitions. These are useful, for example, to measure how many times a client needs to contact a log to request a proof or how many nodes have the latest STH. PRISM provides various reward-based properties. Examples are:

- $\mathbf{R}^{\textit{queue}}_{=?}[\mathbf{I}^{=t}]$ - the expected size of a queue after exactly t steps (instantaneous reward).
- $\mathbf{R}^{\textit{power}}_{=?}[\mathbf{C}^{\leq t}]$ - the expected cumulative amount of power used within t steps (cumulative reward).
- $\mathbf{R}^{\textit{requests}}_{=?}[\mathbf{F}\ \texttt{end}]$ - the expected number of requests before a protocol terminates (reachability reward).

For precise details of the semantics of these, and the algorithms required to model check them, see for example [12].

C. Sequential Model-based Global Optimisation

Finding the maxima or minima of complex, multivariate functions can be a costly process. Furthermore, the analytic form of the function itself may not be known, and so it is necessary to optimise it using only a series of inputs and outputs. Given an infinite set $\mathcal{X}$ and a real-valued objective function $F : \mathcal{X} \to \mathbb{R}$, *black-box optimisation* finds a point $x^* \in \mathcal{X}$ that best minimises (or maximises) F. The function can only be called a finite number of times using an input x to obtain $F(x)$.

Sequential model-based global optimization (SMBO) [10] is a form of Bayesian optimisation which uses a cheaper surrogate function in place of F. It continuously tunes the surrogate with each received output of $F(x^*)$, with x^* being the input suggested by the surrogate model proposes, until we reach the maximum number of trials permitted SMBO is a popular method in the field of machine learning when neural networks but can be equally applied to similar problems where a parameter space needs to be explored in order to optimise a particular function [10].

SMBO algorithms can be generally described as follows: given an objective function $F : \mathcal{X} \to \mathbb{R}$, we replace it with a surrogate model $\mathcal{M}$ and repeatedly perform two tasks: optimise a criterion function S, which measures the 'interest' of asking F which points to evaluate, and reconfigure $\mathcal{M}$ after expanding our dataset with a new data point $(x^*, F(x^*))$. In this paper, we use the tree-structured Parzen estimator (TPE) algorithm implemented by the Hyperopt library (see subsection V) which creates two non-parametric probability density functions formed by previous observations when the output of F is above or below a threshold chosen by the algorithm. In other words, it creates two sets where one of them consists of all the "good" points that sufficiently minimises F. TPE chooses the next point by drawing from the density formed by these "good" points and evaluating the ratio between these two densities [1].

IV. Modelling and Verification

A. Network Abstraction

Recall that clients gossip STHs using servers as staging posts that will distribute them to other clients. For the purposes of modelling and verification, we define an abstraction of the network comprising the clients and the servers they regularly connect with. We distinguish between several types of client and servers, based on their behaviour, and aggregate the behaviour of clients or servers of the same type.

In our model, network traffic occurs in a sequence of 'rounds', during which clients make connections with servers. We assume that all clients are capable of gossiping. The average rate at which these clients gossip with servers, which we call the *gossiping rate*, is defined as the average proportion of outgoing connections to these servers where gossip is being used. The average rates at which at which these clients connect to each domain given that it will gossip with them, called the *client profile*, is the average proportion of outgoing connections to each server where gossip is being used out of the total number of outgoing connections to these servers where gossip is used. We discuss how we derive suitable values for these parameters in Section V.

Formally, we define a *network topology* to be a triple $(\mathcal{C}, \mathcal{S}, \mathcal{P}, \mathcal{G})$, where:

i) $\mathcal{C}$ is the set of clients types,
ii) $\mathcal{S}$ is the set of server types,
iii) $\mathcal{P} : \mathcal{C} \times \mathcal{S} \to [0, 1]$ is the client profile, where, for each $c \in \mathcal{C}$, we have $\sum_{s\in\mathcal{S}} \mathcal{P}(c, s) = 1$,

iv) $\mathcal{G} : \mathcal{C} \rightarrow [0,1]$ is the gossiping rate function.

We assume that there exists a log server that everyone in the network can contact which is operated by a maintainer. The maintainer can either be honest or malicious depending on the scenario; we describe this in more detail later.

B. Modelling the Protocol

The basic structure of our models can be described as follows: first, clients randomly decide whether to participate in a round of gossiping depending on their gossip rate. If they decide not to, they do nothing until the next round of gossiping commences. Otherwise, they choose which server to connect with using their client profile and 'gossip' their STH. Afterwards, everyone in the round updates their local state by comparing against the states of the entity they are connected with and changes their own internal state accordingly. After the round is complete the clients reset themselves by disconnecting but remember their stored messages before a new round begins.

We can initialise the states of our clients and servers in many ways to have different STHs and SCTs stored, bringing more complexity to our model. Therefore, to avoid these problems we impose the following design restrictions:

i) Every client and server in the network will contact only one log; clients that are victims of a split-world attack are redirected to a forked version of the log.
ii) Clients and servers already have valid and non-empty data (i.e., the tree sizes of stored STHs are non-zero) before gossiping begins.
iii) All clients and servers are gossip-enabled.
iv) All the clients have previously audited the SCTs for the servers they can contact.

The state of each client in the model comprises several variables. First, c_g indicates which stage they are at in the protocol execution. A connection state c_s records the server they are currently connected with ($c_s = 0$ means that the client is not connecting with anyone) and c_{skip} denotes whether or not a client decides to skip a round. Note that c_s is different from c_{skip} because if the latter is true then nothing gets altered in a round apart from c_g. A variable c_{sth} keeps track of the STH the client has stored, where its usage depends on the scenario we are looking at. Each server has one variable s_{sth} that has an equivalent purpose to c_{sth}.

Normal and Split-World Scenarios. We want to analyse how well the protocols adapt under normal conditions and during a split-world attack; the log maintainer is honest in the former situation, while in the latter it attempts to fork the log for malicious purposes.

For the normal scenario, the nodes in the network start with an old STH s_l that was generated before the rounds of gossiping begin. We let the one server and one client have the newest STH s_m to start with, where $t_l < t_m$. The reason for this is that, for corporations who provides service to millions of users, it will want to have the latest information of the log for the purposes of security and a client may, for example, act as a self-designated CT auditor who contacts the log server outside of gossiping.

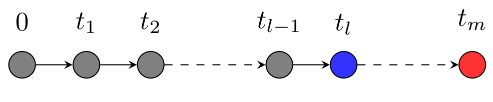
Our second scenario looks at the case when the log maintainer decides to target a single client by forking the log and using fake certificates to steal sensitive information regardless if someone discovers what happened afterwards (a 'smash-and-grab' attack). We describe the threat model as follows. Before gossiping in the model begins, the log commences a split world attack and by maintaining two separate ledgers forked from the original log at tree size t_l (the current size everybody in the network knows) - the genuine log and a 'rogue' log. This rogue log has appended to it fake certificates which are used for servers hosting spoof websites and one of the websites being mimicked includes one present in our network topology. The new STHs corresponding to the genuine and rogue logs are s_m and s'_n respectively, where $t_m, t'_n \in \mathbb{N}$ are the respective tree sizes, $t_m < t'_n$ and timestamp(s_m) < timestamp(s'_n) - that way, the `checkSTH()` function used in the protocols that checks for inconsistency can be bypassed. Up to tree size t_l, both logs have the same list of certificates, implying that consistency between t_l, t_m and t_l, t'_n can be proven.

The rogue log is in collusion with the spoof server previously mentioned using a fake certificate, their goal being to steal information from a particular user when they connect to the server. If somehow a client can be targeted by having their internet connections controlled so that any proof/STH requests they make is redirected to the rogue log, then after gossiping with the spoof server the clients update themselves using the STH s'_n. Suppose that this does happen, meaning that at the beginning we have one updated server with data s_m and a client has s'_n. Everyone else in the network has the old STH s_l such that $t_l < t_m < t'_n$. The spoof server is never contacted again (or the probability of being contacted in the future is so small it is negligible).

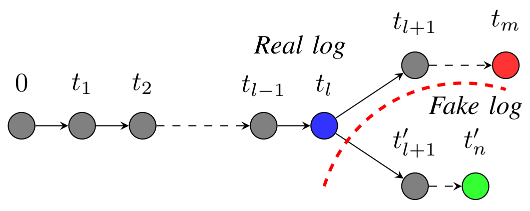
In our model, detection occurs when either a client retrieves both s_m and s'_n through gossiping and receives no response to its request for an extension proof (the log will not be able to provide a valid proof between the real and fake data) or receives a warning message from a server that has already discovered that an inconsistency was found.

The variables c_{sth} and s_{sth} are integers that represent the data type each entity possesses, and are summarised in Table I. The rounds of gossiping end when at least one client detects that something is wrong for the reasons mentioned previously. In addition, we use Boolean variables c_d and s_d as 'detection flags' which are true when a client or server have already detected something, respectively.

Protocol Variations. As previously mentioned in Section III, the format of the gossip messages is different between the *STH-Only* and *STH-and-Proof* versions of the Chuat et al. [4] protocols. Each client stores only one message in both versions, but in *STH-and-Proof* a server stores multiple messages and gossips them depending on what it receives. However, by having everyone in the network possess a baseline knowledge of the log, we make servers gossip only the messages represented by s_{sth} in the model. Table I shows which messages correspond to the values that c_{sth}/s_{sth} can take in both our normal and



(a) Normal



(b) Split-world

Fig. 1: An abstract representation of the growth of the log in (a) a normal scenario and (b) during a split-world attack.

Normal case		Gossip message format	
c_{sth}/s_{sth}	Data Type	*STH-Only*	*STH-and-Proof*
`False`	Old data	s_l	$(s_a, s_l, p_{a,l})$
`True`	New data	s_m	$(s_l, s_m, p_{l,m})$

Split world case		Gossip message format	
c_{sth}/s_{sth}	Data Type	*STH-Only*	*STH-and-Proof*
0	Old data	s_l	$(s_a, s_l, p_{a,l})$
1	Real data	s_m	$(s_l, s_m, p_{l,m})$
2	Fake data	s'_n	$(s_l, s'_n, p_{l,n})$

TABLE I: Summary of what the c_{sth}/s_{sth} variables represent in the models for each value for both the normal and split-world cases.

split-world models.

We are interested in how often the protocols make clients contact the log during their execution. To measure this in our models we use reward structures to count how many times a consistency proof is requested from clients. Each variation of the protocol has different requirements for the clients to call the log depending what gossip messages they receive:

- *STH-Only* - the tree size of the received message is different from what the client already knows, regardless of how old it is. In regards to our models, this means that between a connected client/server pair the log is 'called' when the value of c_{sth} is not equal to s_{sth}.
- *STH-and-Proof* - if we let the client's message be $(s_a, s_b, p_{a,b})$ and the obtained message $(s_c, s_d, p_{c,d})$, where $t_a, t_b, t_c, t_d \in \mathbb{N}$ are tree sizes, it is required that $t_b \neq t_c$ and $t_b \neq t_d$. Due to our model design, this condition can occur if a client has already updated but the server it is gossiping with in a round has not. The client will also request a proof when it obtains both the real and fake data which will trigger a detection.

C. Protocol Properties

We want to evaluate certain quantitative properties that will help us determine if the protocols provide good security and do not place a burden on internet bandwidth. To be more precise, we will look at three desired properties:

i) *Data dissemination*: How effectively do the protocols spread the latest data to each client in the network?

ii) *Protocol efficiency*: How much demand does the protocol put on the log?

iii) *Rate of detection*: How quickly can the protocols detect a split-world attack occurring in the network?

To evaluate property i) using our models, we will measure the expected proportion of clients to possess the new STH per round, computed using reward structures that track the current proportion after round $r \in \mathbb{N}$. In PRISM, this is written:

$$\mathbf{R}^{\mathit{client\_proportion}}_{=?}[\mathbf{I}^{=r}]. \tag{1}$$

Alternatively, we can find the expected amount of gossiping rounds it takes until every client in the network has the latest data, i.e., c_{sth} is set to `true` for every client:

$$\mathbf{R}^{\mathit{rounds}}_{=?}[\mathbf{F}\ \texttt{clients\_all\_updated}].$$

Likewise, we can measure ii) using reward structures, distinguishing between the conditions for each variation to request a proof, and find the cumulative amount of log connections made from clients per round. We use the property:

$$\mathbf{R}^{\mathit{log\_connections}}_{=?}[\mathbf{C}^{\leq r}]$$

Property iii) is exclusive to the split-world model and can be written using the $\mathbf{P}$ operator which is used to compute the likelihood of the occurrence of a specified event:

$$\mathbf{P}_{=?}[\mathbf{F}^{\leq r}\ \texttt{detect}]. \tag{2}$$

Finally, to find the expected number of rounds that will take place before the `detect` event occurs, then we can use the property $\mathbf{R}^{\mathit{rounds}}_{=?}[\mathbf{F}\ \texttt{detect}]$.

V. Modelling Parameters

Our models include a number of parameters that need to be defined, primarily those relating to the network abstraction described in Section IV-A. We now summarise those parameters and our approach to choosing suitable values for them. Our approach is based on defining potential ranges for these parameters and then using sequential model-based global optimisation (SMBO; see Section III-C) to determine *worst-case* values in terms of the reliability or security of the protocol. This means the results we generate for quantitative aspects of the protocol's behaviour can represent guarantees on its performance over a range of network scenarios.

In this paper, we demonstrate our methodology using artificial network data, starting with the presumption that our clients are grouped into three abstract types with differing behaviours, which we will call C_1, C_2 and C_3; we also use five server types $S_1, \ldots, S_5$. Each (honest or split-world) model is then characterised by the following parameters:

i) `type_freq` - The number of clients of each type;

ii) `init_states` - The initial states of the clients/servers, in particular which have the latest log data;

iii) `gsp_rates` - The gossip rates for each client type;

iv) `intervals` - The collection of real-valued intervals for each client type profile, specifying the range of values each probability can take.

Server \ Client type	C_1	C_2	C_3
S_1	$[0.01, 0.1]$	$[0.01, 0.1]$	$[0.01, 0.1]$
S_2	$[0.2, 0.4]$	$[0.2, 0.4]$	$[0.2, 0.4]$
S_3	$[0.2, 0.3]$	$[0.4, 0.5]$	$[0.15, 0.25]$
S_4	$[0.3, 0.4]$	$[0.2, 0.3]$	$[0.15, 0.25]$
S_5	$[0, 0.29]$	$[0, 0.19]$	$[0, 0.49]$

TABLE II: Probability intervals used for each of our three client types, denoted as C_1, C_2 and C_3. We assume that they can connect with five distinct services in the network.

For our experiments, we fix a representative set of intervals for the client profile probabilities, shown in Table II. Here, for simplicity, we will also fix gossip rates at 0.5. For other model parameters, we aim to determine (using SMBO) values that produce the worst-case result for properties we identify as important; for the normal case this is the data dissemination rate and for split-world this is the detection rate (see subsection IV-C). The main idea here is to show evidence of good results for server gossip in the worst-case and then take it as evidence of good results in all cases.

We developed a Python application which uses the Hyperopt library [2], [3] to search over our parameter space and suggest the parameters to use after a number of evaluations. The code and the full set of results produced for this paper are available from [24]. The objective function we optimise first constructs a PRISM model description according to the parameter inputs given and afterwards calls PRISM to analyse the model. A number of options are given to the user to customise how the objection function is executed, for example the PRISM property to use during verification. The objective function outputs a real-valued result related to the property we want to investigate.

VI. Server-to-server Gossip

We also define a variant of the gossip protocols in which servers gossip directly with each other instead of behaving as static entities. Server-to-server gossiping can help spread STHs to different regions of the internet quickly. This mitigates the need for users to venture outside their typical activity to obtain fake data from potentially malicious sources, which we rely on in the case of client-server-only gossip.

Some design decisions arise when deciding how to make the servers gossip. We need to find the correct balance between data sharing and servicing clients and decide how many servers a server should contact in each session; having millions of servers gossip with each other at the same time will place a strain on bandwidth and will result in dissatisfied customers.

We suggest some ways in which servers can discover peers that they can gossip with, inspired by existing mechanisms used by the Tor anonymity network and BitTorrent:

- **Directory authority** - Tor clients contact a directory authority in order to discover any relay points that can be used to create secure channels [20]. To apply this to our extended protocols, we can establish similar authorities that keep track of a list of selected servers for gossiping
- **Distributed Hash Table (DHT)** - BitTorrent uses a DHT mechanism which removes the need for a point of authority (a 'tracker') that keeps track of all the clients in the network. Clients store a DHT, which in practice acts as a sort of routing table, for a small number of 'good' nodes that can respond to requests successfully [16].

a)

Parameter	Best suggestion
`type_freq`	C_1: 1, C_2: 1, C_3: 3
`init_states`	Client of type C_1 and server S_1 has new data
C_1 distribution	$[0.087, 0.205, 0.283, 0.391, 0.034]$
C_2 distribution	$[0.012, 0.377, 0.408, 0.202, 0.001]$
C_3 distribution	$[0.02, 0.223, 0.152, 0.24, 0.365]$
Worst result	9.146

b)

Parameter	Best suggestion
`type_freq`	C_1: 1, C_2: 1, C_3: 3
`init_states`	Client of type C_3 has fake data; server S_1 has real data
C_1 distribution	$[0.012, 0.201, 0.287, 0.382, 0.118]$
C_2 distribution	$[0.014, 0.359, 0.406, 0.206, 0.014]$
C_3 distribution	$[0.01, 0.202, 0.154, 0.164, 0.47]$
Worst result	36.007

TABLE III: The worst-case parameters discovered for a) the data dissemination rate for the honest case and b) the detection rate in the split-world case. Values rounded to 3 d.p

We extend our models with server-server gossip using a simple abstraction: when a server has the latest log data it gossips this outside the update phase, i.e. before clients connect with them. To make sure that a server rarely shares data with everyone and not strain its performance, we fix a probability for it choosing another server to gossip with (including itself) at 0.2. To reduce the complexity in our models, gossiping is conducted using unidirectional channels, i.e. servers send messages but do not receive replies.

VII. Results

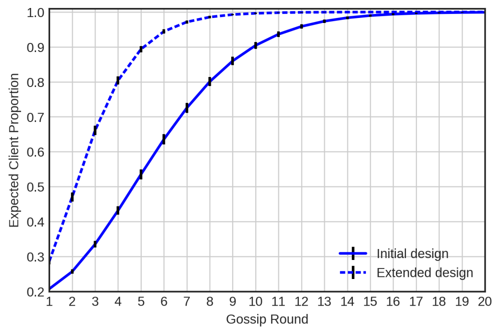
In total, we studied four protocol variants:

- STH-Only protocol without servers gossiping,
- STH-Only protocol with servers gossiping,
- STH-and-proof protocol without servers gossiping,
- STH-and-proof protocol with servers gossiping.

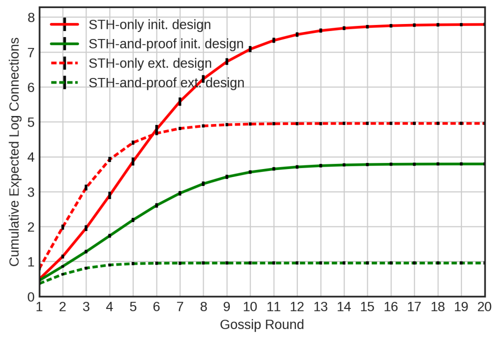
The first and third are the original designs presented in [4], which we will refer to as the *initial* designs. The others are our modified versions, which we call the *extended* designs. We model both types of design under normal conditions and when a split-world attack is taking place, both described previously in Section IV.

We used PRISM's symmetry reduction functionality [11] to improve scalability where clients exhibited the same behaviour. We also use (simulation-based) statistical model checking to generate approximate results. In the following graphs we give the 99% confidence interval using error bars. We will also use statistical model checking to find estimates for larger models where there are many clients in the network while still fixing our number of servers to five.

Using our Python application (see Section V), we performed 200 trials each on our normal and split-world models to find parameters that produce the worst-case result for our properties.

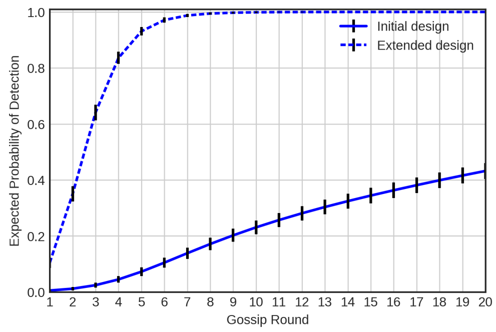


(a) Client proportion

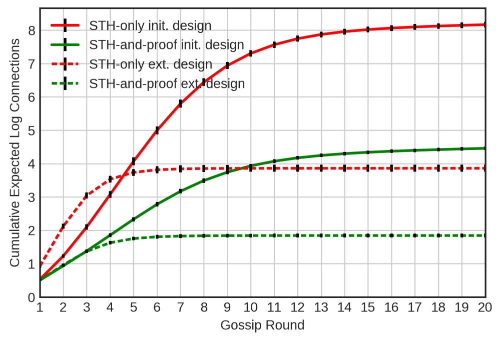


(b) Log connections

Fig. 2: Plots of the model checking results for the normal scenario.



(a) Detection rate



(b) Log connections

Fig. 3: Model checking results for the split-world scenario.

The properties we used for the normal and split-world models are the expected number of rounds it takes to completely spread the latest data to all clients, respectively, so we want to try and maximise them. Table III provides information on the resulting parameter values for each model type.

A. Normal Scenario

Next, using the parameters described above, we analysed several properties of our models, looking at the first twenty rounds of gossiping.

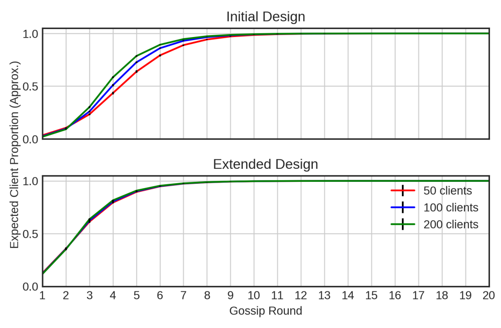
Data Dissemination. We measure the expected proportion of clients that have the latest log data (see Fig. 2(a)). It is clear that having servers gossip ameliorates the data spread; as there are more chances of a server being updated per round, this will impact the client's chance of obtaining the latest data too. For both versions of the Chuat et al. [4] protocol, we expect the dissemination rate to remain the same as the updating procedure for both of them are very similar.

Client Log Connections. We measure the number of connections made to the log to determine the inefficiency of the protocol. Fig. 2(b) shows that *STH-and-Proof* requires less log connections from clients. This is because in *STH-Only* the events where a server's gossip message has a different tree size to what the connected client has is very frequent. However, for *STH-and-proof* the conditions to request a proof are very specific so the chances of a client calling the log is very small. Server-to-server gossip slightly improves efficiency for both versions in the long term, most likely because servers will update themselves sooner than clients.

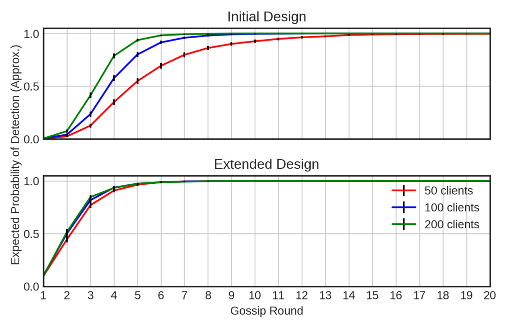
B. Split-World Scenario

Detection Rate of Attack. We expect that the chances of detection per gossiping round to be the same for both versions of the protocol; in the context of our PRISM model design, the conditions for detection in *STH-Only* and *STH-and-Proof* are exactly the same. From Fig. 3(a) we see that using the initial design of the protocol to detect attacks takes a while, where after twenty rounds there will only be about a 42% chance for a client to notice that something is wrong. However, making servers gossip will significantly improve the detection rate as the legitimate data will be spread through the network very quickly and result in an inconsistency being found when fake data is gossiped. We should make aware that the results we obtained does largely depend on the initial conditions in our network; if we let a client with a high connectivity rate be targeted, then it would be more likely for someone to detect an attack in only a few rounds.

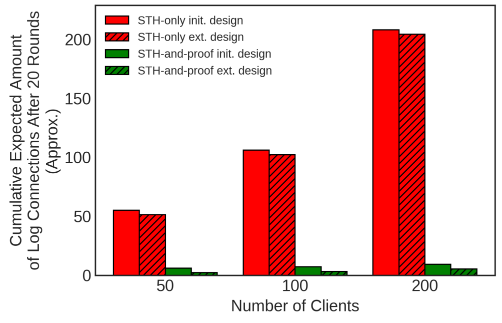
Client Log Connections. The patterns in the results from Fig. 3(b) are very similar to what we have observed previously in the normal scenario but we expected that slightly more connections will be made on average due to the addition of more types of data being passed around in the network.



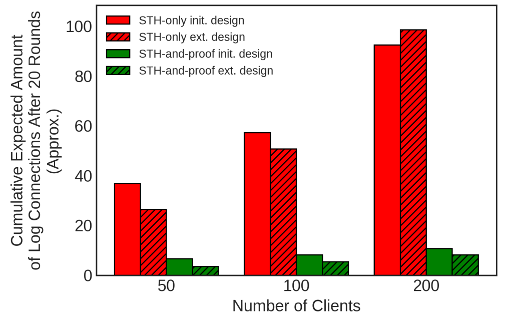
(a) Data dissemination



(a) Detection rate



(b) Log connections



(b) Log connections

Fig. 4: Statistical model checking results for the normal scenario.

Fig. 5: Statistical model checking results for the split-world scenario.

C. Estimating Quantitative Properties for Large Networks

We investigate the gossip protocols when deployed in larger networks; this is very hard to achieve through standard model checking so we rely on PRISM's statistical model checking methods (which are based on discrete-event simulation) to find approximate values. Taking our original models with five clients and servers each, we scaled them to include 50, 100 and 200 clients while still fixing the number of of servers at five and preserving the proportions of each type of client. For all our results, we use the statistical methods to find our estimations with their respective 99% confidence interval by generating 2000 path samples. The amount of time it took to output results did often fluctuate but in general we found that it took longer to find results for our split-world models.

Normal Scenario. For both the initial and extended designs, we approximate the data dissemination for the first 20 rounds of gossiping. In Fig. 4(a), we also provide the confidence intervals at each round represented by a black bar. We can clearly see that by having our servers gossip the clients will get updated quickly. When measuring efficiency, in Fig. 4(b) we see a similar pattern emerging from when we looked at a smaller network, with the STH-and-Proof version looking more scalable as the number of clients increase. However, server gossip does not appear to affect efficiency significantly.

Split-world. The approximate detection rate for the first 20 rounds is shown in Fig. 5(a), suggesting that server gossip improves this property. One interesting observation is that the detection rate for the extended design appears invariant of the number of clients there are in the network. This is likely because, as more servers go into the detection phase early, computing the probability of detection reduces to finding the chances of at least one client connecting to a server that has already started to gossip warning messages. Maintaining the ratios between the types of client may have also had an impact on the results too. In Fig. 5(b) we see similar outcomes in the expected cumulative total of log connections as in the normal scenario but *STH-Only* produces slightly more connections than before with server gossip as the number of clients increase to 200.

VIII. Discussion

From our analysis, *STH-and-Proof* appears to be more adaptable to large networks than *STH-Only* as it requires fewer client-side connections to check for data consistency. Even at a low rate of gossiping, our simple server-to-server gossip mechanism improved the data spread and as a result clients could detect inconsistencies earlier. In practice, servers will not always be able to gossip and administrators may object to their machines talking with other servers they have no control over. Despite this, we recommend that future gossip protocols for CT make servers gossip in the same fashion as clients instead of being static objects in the network.

One of the main challenges in designing our models was overcoming the state space explosion. We have used several approaches to improve scalability, notably our grouping of clients into clusters in our network abstraction, and the combination of this with symmetry reduction. However, modelling for networks with much larger networks of clients with diverse behaviours is still a challenge.

Another limitation to our models is the design compromises we had to make. In large networks, many devices will have different levels of knowledge of the log, connecting with many servers concurrently and will be auditing SCTs as part of the protocol logic, requiring extra log connections. Apart from including more nodes in the network and more diverse client profiles, a further detail we could include in the models are Boolean flags for each client indicating which SCTs they have successfully audited or make c_{sth}/s_{sth} take a range of integers indicating the largest tree size the client/server knows. On the other hand, this will make model checking much more expensive and, when comparing the SCT audit process in *STH-only* and *STH-and-Proof*, the logic for both of them is identical so adding this feature will not add a new layer of meaning to our results.

With the growing prevalence of cloud-based services on the internet, gossip protocols such as the ones discussed in this paper will become vital in the future to ensure the protection of these services against non-equivocation attacks that can severely affect the users and businesses that rely on them. By managing a diverse range of connections from across the globe, cloud services are well suited to act as auditors that pool large amounts of STH data and perform consistency checks with them. As soon as a log is deemed untrustworthy, warning messages can easily be propagated to clients (or other third-party auditors) that are in constant contact with the cloud which will quickly gain the attention of a log monitor. Finally, as we have previously seen, the relatively little overhead some of these protocols produce indicates that they're unlikely to negatively impact bandwidth and thus maintain the availability of these services. Whilst we used a client-to-server network topology in this paper, there should be very little change in our methodology when analysing a "client-to-cloud" network topology where clients randomly connect with distinct cloud services.

In the future, we would like to investigate how we can reliably collect internet traffic data so we can establish more unique client types for our models. It would also be interesting to see if the techniques presented in this paper can be applied to other gossip protocols proposed for other log-based systems [17], [5].

IX. Conclusion

We have presented a new methodology for formally evaluating quantitative aspects of the security of gossip protocols for certificate transparency, using probabilistic model checking. We explained our abstraction of the network and the properties we measured. We applied Bayesian optimisation to find the parameters for our models that give the worst-case scenarios for our properties. This means the results we generate for the protocol's behaviour can represent guarantees on its performance over a range of network scenarios.

From our results, it appears that gossiping consistency proofs with STHs improves the security of users at minimal cost. We have also proposed a variant of the protocol in which servers gossip directly with each other (instead of just via clients). Our verification methods show that this variant improves the security and efficiency aspects of the protocols.

References

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[2] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox. Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1), 2015.

[3] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proc. ICML'13*, pages 115–123. JMLR, 2013.

[4] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *Proc. CNS'15*, pages 415–423. IEEE, 2015.

[5] M. Etemad and A. Küpçü. Efficient key authentication service for secure end-to-end communications. In *ProvSec'15*, pages 183–197. Springer, 2015.

[6] C. Evans, C. Palmer, and R. Sleevi. Public Key Pinning Extension for HTTP. RFC 7469, Apr. 2015.

[7] A. Fehnker and P. Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *ADHOC-NOW*, volume 6, pages 128–141. Springer, 2006.

[8] P. Hallam-Baker and R. Stradling. DNS certification authority authorization (CAA) resource record. RFC 6844, Jan. 2013.

[9] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[10] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. LION'11*, pages 507–523. Springer, 2011.

[11] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proc. CAV'06*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.

[12] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *SFM'07*, volume 4486 of *LNCS*. Springer, 2007.

[13] M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in PRISM. *ACM SIGMETRICS Performance Evaluation Review*, 36(3):17–22, 2008.

[14] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV'11*, pages 585–591. Springer, 2011.

[15] B. Laurie, A. Langley, E. Kasper, E. Messeri, and R. Stradling. Certificate Transparency Version 2.0. Internet-Draft draft-ietf-trans-rfc6962-bis-27, Internet Engineering Task Force, 2017. Work in Progress.

[16] A. Loewenstern and A. Norberg. The BitTorrent DHT protocol. http://www.bittorrent.org/beps/bep_0005.html. Accessed 2020-05-05.

[17] M. Melara, A. Blankstein, J. Bonneau, E. Felten, and M. Freedman. CONIKs: Bringing key transparency to end users. In *Proc. USENIX'15*, pages 383–398, 2015.

[18] L. Nordberg, D. Gillmor, and T. Ritter. Gossiping in CT. Internet-Draft draft-ietf-trans-gossip-05, Internet Engineering Task Force, 2018. Work in Progress.

[19] J. Schlyter and P. Hoffman. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC 6698, 2012.

[20] P. Syverson, R. Dingledine, and N. Mathewson. Tor: The second-generation onion router. In *Proc. USENIX'04*, pages 303–319, 2004.

[21] M. Webster, M. Breza, C. Dixon, M. Fisher, and J. McCann. Formal verification of synchronisation, gossip and environmental effects for critical iot systems. In *Proc. AVoCS'18*, 2018.

[22] C. Williams. Globalsign screw-up cancels top websites' HTTPS certificates. http://www.theregister.co.uk/2016/10/13/globalsigned_off/. Accessed 2020-05-05.

[23] K. Zetter. Diginotar files for bankruptcy in wake of devastating hack. https://www.wired.com/2011/09/diginotar-bankruptcy/. Accessed 2020-05-05.

[24] Supporting material. http://www.prismmodelchecker.org/files/spc20/.