Alexandra Dirksen*, David Klein, Robert Michael, Tilman Stehr, Konrad Rieck, and Martin Johns

LogPicker: Strengthening Certificate Transparency Against Covert Adversaries

Abstract: HTTPS is a cornerstone of privacy in the modern Web. The public key infrastructure underlying HTTPS, however, is a frequent target of attacks. In several cases, forged certificates have been issued by compromised Certificate Authorities (CA) and used to spy on users at large scale. While the concept of Certificate Transparency (CT) provides a means for detecting such forgeries, it builds on a distributed system of CT logs whose correctness is still insufficiently protected. By compromising a certificate authority and the corresponding log, a covert adversary can still issue rogue certificates unnoticed.

We introduce LogPicker, a novel protocol for strengthening the public key infrastructure of HTTPS. LogPicker enables a pool of CT logs to collaborate, where a randomly selected log includes the certificate while the rest witness and testify the certificate issuance process. As a result, CT logs become capable of auditing the log in charge independently without the need for a trusted third party. This auditing forces an attacker to control each participating witness, which significantly raises the bar for issuing rogue certificates. LogPicker is efficient and designed to be deployed incrementally, allowing a smooth transition towards a more secure Web.

Keywords: web privacy, web pki, secure communication

DOI 10.2478/popets-2021-0058
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Over the last decades the popularity of web-based applications like online shopping, banking or instant messaging has greatly increased. So has the deployment of HTTPS, which has become the cornerstone of privacy on the Web. In January 2020, 95% of page loads in Google Chrome were served over HTTPS [28]. Consequently the number of attacks on the Web's Public Key Infrastructure (Web PKI), which is in charge of issuing certificates used by HTTPS, is increasing. The trust anchors of the PKI are Certificate Authorities (CA), that issue X.509 certificates [17] used on the Web. The CA/Browser Forum [3, 21, 22] agreed upon guidelines for issuance and management of X.509 certificates.

However, trusting CAs has not always turned out well. Several cases of illicit certificate creations have been reported [33, Sec. 3.3]. The creation of such *rogue certificates* usually goes unseen and can be used for identity theft. Hence it follows, *not all CAs can be trusted* to check the ownership of a domain before issuing a certificate, which is a violation of the guidelines. Although the first documented certificate mississuance happened in 2001 [10], the case of DigiNotar in 2014 was the first to raise public awareness. This security breach resulted in the issuing of more than 500 rogue certificates which were used to intercept the communication of over 300.000 Iranian citizens [23, 56]. Further violations were documented up to this year [39, 40, 46, 49].

The cryptographic properties of the Web PKI provide a high level of security and are assumed to not be forgeable. Soghoian and Stamm [65] introduce an attacker who bypasses the cryptographic mechanisms by compelling CAs and service providers to assist in their act of surveillance. This implies a strong attacker like a government, that is in control of a subset of the DNS structure and uses rogue certificates for Man-in-the-Middle (MitM) attacks in order to intercept user communication. The authors refer to this surveillance attempts as *government compelled certificate creation*.

Some cases indeed demonstrate that states are willing to abuse the Web PKI for surveillance purposes. In 2016 and 2019 the government of Kazakhstan asked Mozilla to include their *national security certificate*, which was intended to intercept citizen's data [35, 48, 57]. After the request had been declined, the government officially instructed all Kazakh citizens to manually install this certificate. Both attempts of surveillance did

***Corresponding Author: Alexandra Dirksen:** TU Braunschweig, E-mail: a.dirksen@tu-bs.de
David Klein: TU Braunschweig, E-mail: david.klein@tu-bs.de
Robert Michael: TU Braunschweig, E-mail: r.michael@tu-bs.de
Konrad Rieck: TU Braunschweig, E-mail: k.rieck@tu-bs.de
Martin Johns: TU Braunschweig, E-mail: m.johns@tu-bs.de

not remain unseen and browser vendors took action to protect their users [48].

In 2014 Ben Laurie proposed Certificate Transparency (CT) [41, 43] in response to these attacks. It was first deployed by Google's browser Chrome and voluntarily used by some CAs. CT uses public append-only logs in which all certificate issuances must be logged. Domain owners can now detect possible miss-issuances by monitoring those logs for their own domains. The impact of CT soon became visible [34, 66]. To speed up adoption Google forced CAs to comply with CT guidelines if they wanted their certificates to be trusted by Chrome by the end of 2017 [62].

Unfortunately in May 2020, DigiCert reported the compromise of one of its CT logs due to a technical vulnerability [4]. To our knowledge, this is the first documented case of a log compromise and it was reported one day after its discovery by the operator itself. This is of high concern, since DigiCert not only is a CA but also operates CT logs and it is difficult to prove that those logs have not been affected by this vulnerability as well. It is evident that combined attacks against CAs and CT logs will increase, once existing strategies for surveillance of internet users are not applicable anymore.

1.1 Motivation

All usable CT logs included in the Chrome browser are still operated by CA vendors only [26]. This raises two major concerns: On the one hand, a compromise of a CA easily leads to a compromise of a CT log of the same vendor. On the other hand, CAs can send the certificates to their preferred CT logs at the outset of its issuance process. It leads to an opportunity for sufficiently strong attackers to perform the aforementioned compelled certificate creation attack. The attack model assuming both CA and CT log being malicious has been proposed by Kent [37]. Soghoian and Stamm described in [65] a realistic scenario that shows how this attack can be of use during diplomatic negotiations or for industrial espionage. This kind of attack is known as a "low probability, high impact" event and has been introduced by Bussiere and Fratzscher [14]. For political, commercial or socially motivated attacks, as it is mostly the case with surveillance and espionage, Aumann and Lindell [8] introduced the *covert adversary* model. With these assumptions in mind we ask: How can we ensure the security of the Web's PKI if no single entity in the system can be trusted?

Strengthening the security of this PKI, however, is a challenging task. Several attempts have been made to improve the idea of CT, yet they suffer from issues of deployability, usability, and security. We believe that the current Web PKI is already complex enough and thus we strive for security mechanisms that do not add further single points of failure.

1.2 Contribution

We introduce LogPicker, a protocol that strengthens the Web PKI in multiple ways:

First, attacks resulting from the collaboration of malicious CAs and CT log are mitigated. Second, the protocol enables the automatic and targeted auditing of CT log entries. Third, the new certificate issuance process allows the inclusion of CT Monitoring into the process, in order to make rogue certificates detectable.

LogPicker rests on a collaborative logging scheme, where a randomly selected log includes the certificate while the remaining witness and testify the certificate issuance process. For this the roles of existing components of the Web PKI are extended, opening up new opportunities to address other problems like CT Monitoring. We evaluate LogPicker's contribution to the overall correctness probability of the Web PKI using a probabilistic analysis and compare the results to CT and its extension Gossip. In addition, we present a prototype of LogPicker implemented in a simulated Web PKI and analyse our simulations.

Section 2 highlights some background and related work. Our threat model and goals are described in Section 3. Section 4 introduces the preliminaries and a detailed presentation of the LogPicker protocol. Then, LogPicker's achievements are analysed in Section 5 and the prototype as well as its measurements are presented in Section 6. Finally, Sections 7–8 discuss some limitations and future work and conclude the paper.

2 Certificate Transparency

Throughout this work we denote the basic PKI by *CA-based PKI* while other PKIs including additional mitigations are prefixed with their name, e.g., CT-based PKI. In this section we discuss the concept of the CT-based PKI, its extension Gossip as well as other proposals to the CA-based PKI, related to our work. In addition, we discuss the impact of these on the user's privacy.

2.1 Concept

The concept of CT enforces CAs to log all certificate issuances to *public logs*. This allows domain owners to continually check whether all issued certificates for their domain are legitimate. Those logs are operated by independent entities, who currently have no financial benefit from running these logs. Log operators cite their commitment to improving the overall security of the Web PKI as their motivation [68]. Logs must accept all certificates submitted by trusted CAs, append them to their history and make them available to the public.

After a certificate is submitted to a log it must respond with a *Signed Certificate Timestamp* (SCT) [43, Sec. 3] for this certificate, containing the log's signature, among others. An SCT can be seen as a cryptographic promise that the certificate will be included within a *Minimum Merge Delay* (MMD), usually 24 hours. CAs can then serve the SCT along with the newly created certificate to domain owners. CT-enforcing browsers display websites as secure only if they serve a valid SCT with the corresponding certificate. Both, the CT log, issuing the SCT and the CA, issuing the certificate must be trusted by the browser. There are three ways for domain owners to deliver SCTs to web clients [2]:

- SCTs can be *embedded into certificates* using an X.509 extension. However, the SCT cannot be changed once the certificate is created.
- SCTs can be sent using a *TLS extension* during connection establishment. Using this, domain owners can still profit from CT even if their CA does not support it. However, in order to support TLS extensions web servers require an update, which is a major drawback, as discussed in Sec. 3.4.
- SCTs can be delivered via the *Online Certificate Status Protocol* (OCSP) [60]. This requires support from all PKI participants and raises the question of how to deal with unresponsive OCSP responders, which is the reason why it still lacks sufficient deployment.

Since the deployment of CT the user's trust is shifted from former CAs to CT logs. This leads to the question why CT logs deserve more trust than CAs. To this date (June 11, 2021), all logs in Chrome's trust store are operated by CAs, spread over five companies. Therefore it is reasonable to assume that the difficulty of attacking a log is similar to attacking a CA. During the initial design of CT Laurie already considered this problem: "*fixing one set of trusted third parties by introducing another doesn't seem like a step forward.*" [41]. Due to the design of CT it is hard to ensure that the selected log is not controlled by an attacker. In order to mitigate this problem, CT introduces additional roles to the PKI:

Monitor: They check CT logs for rogue certificates issued for (usually their own) domains.

Auditor: They check whether a log's behavior complies with the CT specification.

In the current state, only the issuing CA and the corresponding log are aware of the creation of a certificate. This creates an attack window for the aforementioned compelled certificate creation and split-view attacks, described in Sec. 2.2. The adoption of LogPicker changes the issuance protocol in a way that the choice of the log in charge is shifted from the issuing CA to a set of other logs, resulting in a large number of witnesses of the certificate creation. As a result, an adversary is required to compromise a complete set of CT logs to successfully launch a surveillance attack.

However, consideration must be given to how protocol changes on the CT-based PKI would affect all current participants that must adopt it:

CAs/ CT logs: It is their business to provide services to the Web PKI. Their customers expect them to deal with all changes to the infrastructure and issue certificates which are accepted by all common web browsers. Thus they are motivated to adopt all changes that are enforced by browsers.

Web Clients: They are mostly web browsers, whose vendors are motivated to increase their user's security. Due to the fast release cycles and auto-updates they can deploy new protocol changes quickly.

Web Servers: A significant number of them run outdated software [71], this indicates that usually no changes to a server configuration are made as long as it continues to function.

Policies of the CT-based PKI

Since 2018 two of the major browsers enforce CT: Google's Chrome [51] and Apple's Safari [5]. Mozilla has announced to follow suit with Firefox [15]. Browsers do this by requiring each website's certificate being accompanied by a valid SCT in order to be accepted. The requirements on the CT-based PKI are defined in two policies:

CT Policy: It requires each CT log to comply with the common CT specification [32]. These include e.g., inclusion of a certificate within the MMD, an

availability of at least 99 % and never creating a split view attack.

Browser's CT Policy: Since browsers trust log operators only partially they place additional constrains on their own individual policy [5, 64]. Browsers maintain a list of trusted CT logs which is updated with the browser's regular update mechanism. If a log applies for inclusion in such a list it is first audited for a certain duration to ensure it complies with this policy. A browser accepts only SCTs from logs appearing in it's trusted list.

If a webserver wishes it's certificates to be accepted by a browser, it must comply with the browser's CT policy.

2.2 Auditing

Technically, logs are implemented using a Merkle Tree structure. The tree's root hash, along with the logs signature, is called Signed Tree Hash (STH) and the leaves are hashes of certificates to include. According to the specification in RFC6962 [43] a CT log must:

L1 be available for public queries

L2 include certificates for which it issued an SCT within MMD

L3 maintain an append-only history

To audit whether logs comply with the RFC, voluntary auditors are employed. They make use of the Merkle Tree structure of the logs to request the following proofs for subjected certificates:

Inclusion In order to prove the inclusion of a certificate (*L2*) of a given SCT, an auditor computes a root hash and compares it to the latest root hash provided by the log.

Consistency STHs are used for proving a log's consistency. Auditors collect them from time to time and check whether newer STHs contain the older ones, as required by *L3*. According to specifications, a log must always respond to STH requests with the newest STH no older than the logs MMD.

To allow auditors to check for *L1* and *L2*, compliance with *L1* is required. If a log violates the specification, it will be documented via the proofs gathered by auditors. Logs cannot deny violations since they have to sign all SCTs/STHs. Usually, auditing is meant to be performed by third parties or clients.

Third Party Auditing

Third party auditors must collect SCTs, e.g., by crawling the Web, to perform their auditing task. However, crawling cannot guarantee that all SCTs used on the Web are found. More importantly, third party auditors cannot see SCTs used for targeted MitM attacks, described in Sec. 3.1. In those attacks rogue SCTs and certificates are presented to the victim only. Thus they are not included in public logs, as discussed in Sec. 3.2.

Auditing is an essential task needed to ensure the correct execution of CT. As we stated in Sec. 1.1, in the current state the CT-based PKI lacks an auditing procedure that deserves the client's trust.

Client Auditing

Another approach proposes that clients audit the logs on their own. This can be done in two different ways:

Synchronous during the connection establishment. Since this results in an additional request it slows down the connection and raises the question of how clients deal with unresponsive logs.

Asynchronous by caching SCTs and auditing them in a batch later. Firstly, solutions based on caches are prone to cache flushing attacks [9]. Secondly, it is questionable whether resource-limited devices can perform this task.

More important, if clients reveal an SCT, they disclose a portion of their browsing history. In many cases even a subset of this history is sufficient to identify a user [53]. Thus client auditing poses additional privacy breaches [45] like user tracking [50, Sec. 10.5.2]. Until today no browser implements client based auditing or plans to do so in the future.

Split-View Attacks

CT, by design, allows rogue certificates to be detected only after they have already been issued, and possibly misused. In addition, the MMD provides an attack window where rogue certificates are accepted if accompanied by an SCT but without ever appearing in the log. To implement such an attack a malicous log has to create and maintain two diverging Merkle Trees, which is called a split view attack [16][50, Sec. 10.1]. One of the trees contains a rogue certificate and is served to the victim only. The other tree without the rogue certificate is presented to the public. In order to carry out its attack successfully, the attacker further must identify the victim's request anytime during the malicious log's lifetime. This is especially difficult with moving victims or those with an unknown network infrastructure.

The split-view attack was already considered an open problem in CT's threat model [37, Sec. 3.3.2]. In 2017 the IETF published a draft for a protection mechanism against split view attacks in CT-based PKI [31]. This mitigation, named STH-Cross Logging, proposes to log STHs in one additional *witnessing* log. However, as stated in [31, Sec. 7], if the additional witness is malicious as well, this procedure is useless.

2.3 Extensions

This section first introduces Gossip, an extension to CT that tackles the problem of split view attacks, proposed by Chuat et. al [16]. In addition, we briefly recap other proposals to the Web PKI or CT, as analysed and categorized by Jiangshan and Mark in 2017 [75]

Certificate Transparency - Gossip

The Gossip protocol was adopted as an IETF draft [50]. It's goal is to have clients share their view of a website at different times. On first page load, clients cache SCTs and STHs received by the website, which is referred to as *gossip*. At a later reconnect, the clients return the gossip collected earlier back to the website. In addition, gossip is autonomously collected by third party auditors directly from the website.

If an attacker uses a rogue certificate for a MitM attack on a website, they must hide it from a public view and deceive the client by serving the client an SCT and STH of the malicious view. This allows auditors to detect the attack post mortem, since the malicious log never included the rogue certificate and thus cannot provide the inclusion proof.

The efficacy of the Gossip protocol is based on "the hope that [the] MitM attack will eventually cease, and the client will eventually communicate with the real web server" [1, Sec. 3.3]. In this case diverging SCTs from different sides of the split view would "hopefully find their way to CT auditors", who are in charge of verifying inclusion proofs from the logs [50, Sec. 3].

For users CT Gossip comes with the privacy issues resulting from e.g., fingerprinting attacks [50, Chp.10.5.2]. In addition, each web server must support Gossip since "a web server not deploying SCT Feedback may never learn that it was attacked by a malicious log" [50, Chp.9.1]. Requiring Gossip support by all web servers makes the web-wide adoption nearly impossible. In [58] Ritter, one of the original authors of CT further discusses the challenges of deploying CT Gossip.

Difference Observation

The idea is to have clients share the certificates they observe for a certain domain with each other. This technique was first implemented as a Firefox extension [73]. Another approach was TACK [47] which waives CAs completely. Web servers use their private key for signing self-issued certificates. TACK uses the *trust on first use* (TOFU) assumption, where clients establish a trust relationship to the server on first contact by adding its public key to their trust store. If another key is send upon a later connection, they consider the server untrusted. The usage of TOFU poses the danger of a protocol lock-out due to the fact that a legitimate key change is indistinguishable to one issued by an attacker. Due to these concerns we refrain from the TOFU assumption in LogPicker.

Scope Restrictions

Another way of dealing with the problem of misbehaving CAs is by limiting their abilities to create certificates for domains that will be accepted by clients. In the setting of *HPKP* [19] a server pins its public key for the use of future TLS connections and announces this via an HTTP header. Similar to TACK, HPKP is based on TOFU and meanwhile deprecated due to problems like HPKP suicide [76]. DANE is based on DNS records [32] and allows web servers to pin their certificate in a DNS record. Another approach is the use of CAA records [30]. A domain owner can specify which CAs are allowed to issue certificates for its domain.

Management Transparency

This approach forces CAs to make their certificate issuance transparent, but generally only allows the detection of miss-issuance afterwards. CT, which was already discussed in Sec. 2.1, is probably the most prominent technique.

Using *Sovereign Keys* [18] domain owners cross-sign their CA-issued certificates and publish their related public keys to append-only logs. This solution is similar to TACK. *ARPKI* [11] introduces ARCert, which is composed of two certificates issued by different CAs and published on integrity logs, similar to CT. The logs must be synchronized to prevent split view attacks. *DTKI* [74] allows key revocation for web server keys with the use of *Sovereign Keys*. However, it requires clients to audit synchronously which is hardly feasible as described in Sec. 2.2. In context of CT *CoSi* [70] allows a group of witnesses to co-sign a log's SHTs and append their resulting commit as aggregated signatures to it. This makes split view attacks more difficult but

requires the clients to perform inclusion proof checks. CoSi shares some similarities with LogPicker. However, due to the aforementioned issues we strictly refrain from involving clients into the process of auditing.

3 Threat Model and Goals

A broad threat analysis on CT has been proposed by Kent and published by the IETF [37]. We focus on the specific case where the corresponding CA and CT log are compromised and collaborate in the attack. Kent admits to this attack scenario being out of scope for CT and states that a "distributed audit mechanism" must be employed to detect it.

This work provides such a mechanism: With the integration of LogPicker, the selected log no longer needs to be trusted, such that one single log is sufficient to include the certificate. Thus throughout this work we assume that the browser's policy requires one log (Sec 5.1.1). As described in Sec. 1.1 we assume the *covert adversary* [8] model for this attacker.
In addition, each individual entity of the Web PKI that behaves arbitrarily, e.g. malicious or faulty, is assumed *byzantine*, as defined by Lamport et al. [38].

3.1 Attack Scenario

For our model we assume a targeted attack where the attacker wants to violate a user's privacy by either spying on or modifying the data sent between the user and a server. The attacker wants to perform a MitM attack on the web traffic between them. We assume TLS communication operates as expected and the attacker refrains from attacking its cryptographic primitives. Instead the attacker aims to obtain a certificate for the domain they wish to attack. They control a CA and a CT log, which is even easier if the CA already operates a CT log. The attack is performed in two steps:

1. The attacker compels the CA to omit the domain ownership check in order to create a certificate for a domain, which they have no access to. Thus the CA must omit the domain ownership check and submit the certificate to a log that is also controlled by the attacker.
2. This log creates an SCT for the corresponding certificate. It serves the victim a view containing the rogue certificate while omitting it from the public view.

We are aware that reaching this level of control is not easy. CA and CT log operators know that they are high-value targets and thus it can be assumed they use good security practices. However, as highlighted by the multitude of security issues in Sec. 1.1 we consider this a realistic scenario. In the following we introduce the attacker's capabilities and the security and privacy goals which LogPicker must achieve to protect the Web PKI against them.

3.2 Attacker Capabilities

We consider a *malicious-but-cautious* adversary, who is "malicious if they can get away with it" and "cautious in not leaving any verifiable evidence of its misbehavior" described by Ryan et al. in [59]. This refers to the *covert adversary* model [8]. To successfully carry out the attack described in Sec. 3.1 the attacker must:

AC1 *Compel a CA* to omit the domain ownership check and create a rogue certificate.

AC2 *Control a CT log* that hides the rogue certificate from the domain owner's monitor and present a malicious view to the auditors. This control can be in a direct manner or delegated, e.g., by instructing the compelled CA to use a CT log of the same malicious vendor.

AC3 *Control the connection* between victim and server. They are either able to directly redirect the victim's traffic or they use known attacks on TCP like session hijacking [24] or DNS attacks like DNS cache poisoning [67] just to name a few.

3.3 Privacy & Security Goals

Our goal is to strengthen the Web PKI against the presented attacker in order to preserve the user's privacy. Our defense has to fulfill the following security goals:

SG1 The attacker should not be able to create a rogue certificate *without being detected eventually*. This goal is crucial since the covert adversary will not take any risk of being detected.

SG2 The solution should *not require trusting single entities*. Vice versa, it has to work under the assumption that participating CAs and a high number of logs are malicious.

SG3 To avoid a privacy breach, the *client or any of the user's private data must not participate neither in protocol execution nor in auditing.*

SG4 To avoid lockouts, the solution should *not rely on additional trust assumption* like TOFU.

3.4 Design Goals

We consider the following design goals to create a solution acceptable to all entities in the Web PKI:

DG1 *Performance*: A significant slow down of page loads decreases the user's web experience [12, 72]. A viable solution *must not slow down the connection*, else it runs the risk of not being accepted by browser vendors.

DG2 *Independence*: We want CT logs to *remain independent* in the sense that they may contain different certificates. We do not see the proposals for synchronized CT logs, e.g., ARPKI [11] as practical, as it would require additional coordination between all log providers.

DG3 *Scalability*: The protocol should be *scalable to application in the whole Web*. It must be able to deal with the volume of certificate issuance today and expected in the future. It should enable operators to distribute the computational load on single logs.

DG4 *Incremental Deployment*: We believe that one of the main reasons for the success of CT was its *incremental deployability*. In contrast to browsers optional backwards-compatible changes on the protocols are unlikely to be adopted by servers as seen with HTTP/2 [77]. Thus changes to the Web PKI should not affect web servers.

4 Log Picker

In this section we first give a high-level overview of the LogPicker protocol. Then we introduce the building blocks and notation used in the protocol's description, followed by a detailed description of the protocol's different phases and an additional discussion.

4.1 High-Level Overview

From a high-level point of view LogPicker achieves the goals described in Sec. 3 in three steps:

Step 1 *Convening the witnesses* Depending on the browser policy, a list of logs is selected for the protocol. A CA complying with LogPicker has to obey this selection. The CA chooses one log from this list as a leader and submits the certificate to it for further processing. Note that even if the CA deliberately chooses a malicious log, this has no influence on the security of LogPicker (Sec. 4.3.4). The remaining logs form the *log pool*.

Step 2 *Random log selection* Under the leader's coordination, the pool has the task to collaboratively select one log among them, which will be in charge of including the certificate in its history.

Step 3 *Generating an SCT and LPP.* The leader collects a cryptographic receipt from each log that witnesses and agrees on the selection's outcome. In addition, the selected log creates the SCT and sends it to the leader. From this data it creates a compact LogPicker Proof (LPP) documenting the outcome and successful execution.

Clients who enforce LogPicker accept only certificates that are presented with a valid LPP. Each correct log that witnessed the execution will audit the winning log for inclusion and consistency after the MMD has passed.

4.2 Preliminaries

To formally describe the inner working of the protocol, we first introduce some basic notation and the building blocks utilized to realize the different steps described in Sec. 4.1. Our notation resembles what the reader will find in standard literature on cryptography [36]. The main tools used in our work, namely a digital (aggregate) signature, a commitment scheme and distributed randomness are described best in this notational framework. Throughout this section we assume that the Web PKI's key distribution is reused for LogPicker.

4.2.1 Notation

In the LP-based PKI CT logs are bound to an identity $u \in \mathcal{I}$ with an associated key triple $(\mathsf{sk}_u, \mathsf{pk}_u, \mathsf{k}[ck]_u)$, where the first denotes the secret key, the second the public key and the third the public commitment key. By $\mathcal{L} \subset \mathcal{I}$ we denote the identities of all logs.

Let a and b be two bit strings. By $a \| b$ we denote the concatenation of these two bit strings. We denote the empty string by ε. If a and b are not bit strings we assume that a and b are implicitly encoded into bit strings via a bijective encoding function before concatenation. By $[n]$ we denote the set $\{1, \ldots, n\}$, with $n \in \mathbb{N}$. By writing $\{x_i\}_{i \in [n]}$ we denote the set $\{x_1, \ldots, x_n\}$. By $X = (x_1, \ldots, x_n)$ we denote a vector with $n = |X|$

elements. Let $\mathcal{S}$ be a finite set. By $s \leftarrow_{\$} \mathcal{S}$ we denote that s was sampled uniformly at random from $\mathcal{S}$. With $y \leftarrow_{\$} \mathsf{A}(x)$ we denote the output y of a probabilistic algorithm A for input x. Whenever we use $\leftarrow$ it denotes a deterministic assignment. The operator $\wedge$ denotes the logical 'and'. Whenever the algorithm Now is executed it outputs the current timestamp. With H we denote a cryptographic hash function.

4.2.2 Digital Signature

A digital signature scheme $\mathsf{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrf})$ consists of three algorithms. Initially the signer generates a key pair via $(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{S.Gen}()$, where sk is kept secret to the signer and pk is the public key. An arbitrary message m is signed via $\sigma \leftarrow_{\$} \mathsf{S.Sign}(\mathsf{sk}, m)$. All verifiers in possession of the public key pk can verify the message signature pair via $\mathsf{S.Vrf}(\mathsf{pk}, \sigma, m) = \mathbf{true}$. Informally a digital signature scheme is secure if for a malicious user it is infeasible to forge a valid signature for any m without being in possession of sk. We refer the interested reader to [36]. Our protocol design will use this property later to ensure integrity and authenticity of messages sent by logs.

With a basic digital signature scheme the bandwidth and space requirement for sending and storing signed messages grows linearly with the number of signed messages. Same holds if multiple entities want to sign the same message. In this work we use Aggregate Signatures [13] to reduce this overhead, denoted by AS. An aggregate signature scheme $\mathsf{AS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Vrf})$ consists of four algorithms. The first two algorithms $\mathsf{AS.Gen}$ and $\mathsf{AS.Sign}$ are functionally the same as $\mathsf{S.Gen}$ and $\mathsf{S.Sign}$. Consider $\Sigma = (\sigma_1, \ldots, \sigma_n)$, $M = (m_1, \ldots, m_n)$, $\mathsf{k}[SK] = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and $\mathsf{k}[PK] = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, where $\sigma_i \leftarrow_{\$} \mathsf{AS.Sign}(\mathsf{sk}_i, m_i)$. Via $\sigma_{agg} \leftarrow_{\$} \mathsf{AS.Agg}(\mathsf{k}[PK], \Sigma, M)$, a user is able to aggregate multiple signatures into one aggregate signature. Everyone in possession of the public keys $\mathsf{k}[PK]$ can verify the aggregate signature via $\mathsf{AS.Vrf}(\mathsf{k}[PK], \Sigma, M) = \mathbf{true}$. Analogue to the basic signature scheme, AS is deemed secure, *iff* for a malicious user not in possession of the entire secret key vector $\mathsf{k}[SK]$, it is infeasible to forge a valid aggregate signature for any message vector M, under public key vector $\mathsf{k}[PK]$. Our main intention for introducing this cryptographic primitive is to enable LogPicker to output a compact proof.

4.2.3 Commitment Scheme

Commitment Schemes are a corner stone of many cryptographic protocols. Discussing all the details is out of scope at this place. For clarity we just discuss the basics here and refer the reader to general literature on cryptography [25, 36]. A commitment scheme $\mathsf{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ consists of three algorithms. Consider two entities: a prover and a verifier. Initially the prover generates a public commitment key via $\mathsf{k}[ck] \leftarrow_{\$} \mathsf{C.Gen}()$. Via $(c, r) \leftarrow_{\$} \mathsf{C.Com}(\mathsf{k}[ck], m)$ a prover can commit to a message m by sending $(c, \mathsf{k}[ck])$ to the verifier and keeping r and m secret. Later on the prover eventually decides to reveal the message by sending (r, m) to the verifier which 'opens' the commitment by verifying the statement $\mathsf{C.Open}(\mathsf{k}[ck], c, r, m) = \mathbf{true}$.

A commitment scheme comes with two important security properties. First it is *binding*. This means, after a commitment to m it is infeasible for a malicious prover to change his mind about m and convince the verifier that for some other message m' $\mathsf{C.Open}(\mathsf{k}[ck], c, r, m') = \mathbf{true}$. Second a commitment scheme is *hiding*, meaning no malicious verifier in possession of $(c, \mathsf{k}[ck])$ but (r, m) is able to determine m. Why LogPicker utilizes a commitment scheme will become clear in the next section.

4.2.4 Distributed Randomness

The third core element of LogPicker is that a set of logs randomly selects one log among them. In order to provide an unbiased and random selection even in the presence of potentially byzantine logs we adopted the concept of distributed randomness (DR) first introduced by Popov [55]. Basic DR protocols would require either to trust the leader or each log directly broadcasting its messages to all other logs. Both approaches are unsuitable for LogPicker, because (a) we do not want to introduce a trusted entity and (b) we want to reduce communication complexity. Therefore we propose a leader-driven Commit-Then-Reveal approach, inspired by Syta et al. [69, 70].

Consider a log pool $U = (u_1, \ldots, u_n)$, where $u_i \in \mathcal{L}$ and a leader $u_0 \in \mathcal{L} \setminus \{u_i\}_{i \in [n]}$. First of all u_0 will act as a 'router' by forwarding messages from a log to all other logs in U. This is intended to keep the communication complexity low.

Commit. The logs in the pool commit to a random value $v_i \leftarrow_{\$} \mathbb{Z}_{|U|}$ *without* revealing it. By sending their commitment to the leader u_0. After the commitment the random values v_i are fixed due to the *binding* property of C and still hidden due to the *hiding* property

of C. This has the advantage that neither the leader nor any participating log is able to bias the outcome by adaptively choosing a v_i. At the end of this phase unresponsive logs are excluded from U and the further protocol procedure. Finally the leader broadcasts the vector $V = (v_1, \ldots, v_{n'})$ to the remaining logs.

Reveal. Due to the eventual exclusion of unresponsive logs during the previous phase the protocol proceeds with the remaining pool $U' = (u'_0, \ldots, u'_{n'})$, where $\{u'_i\}_{i\in[n']} \subseteq \{u_i\}_{i\in[n]}$. The remaining logs reveal their hidden v_i. The leader u_0 will broadcast the reveal messages to all logs. Each log can then calculate the final random value v as shown in Eq. 1. We will use this concept to randomly 'pick' a log u'_v from U'.

$$v = 1 + \left(\sum_{i\in[n']} v_i \mod n' \right) \tag{1}$$

4.3 The LogPicker Protocol

Recall, the goal of one LogPicker execution for an input certificate Γ is to randomly select a log from the pool that is supposed to include the issued certificate Γ. Therefore the pool must collaboratively choose a log at random by agreeing on a value v which represents the index of the chosen log. On successful execution the protocol outputs the SCT with an additional LPP that proves the validity of a protocol run.

LogPicker must be able to deal with possibly unreliable or malicious logs. For this task it employs a leader driven protocol, in which some participants may behave byzantine, i.e.:

Logs by not including certificates due to failure or by cooperating with malicious CAs.

CA by not performing domain ownership checks or compelling a log to not include a rogue certificate.

Leader by not creating a valid protocol output or by excluding logs from the execution.

4.3.1 Sending Messages

All messages m sent between logs participating in a protocol run are prepended by a session id $sid \leftarrow \mathsf{H}(u_0 \| t_0 \| \Gamma)$, the senders public identifier $u_s \in \mathcal{I}$, and a signature $\sigma \leftarrow\!\!\$\, \mathsf{S.Sign}(\mathsf{sk}_{u_s}, sid \| u_s \| m)$, where Γ is the new certificate, t_0 the initial timestamp and u_0 the leaders identity. The algorithm to verify such messages is defined in Fig. 1. By Δt we denote the maximum acceptance timespan for messages with the same *sid*. Message verification is designed to prevent replay attacks or other malicious activities that may compromise the integrity or authenticity of a message. Further the *sid* will aid the logs in separating and keeping track of multiple protocol instances.

$\underline{\mathsf{LP.VrfMsg}(\mathsf{pk}_{u_s}, t_0, sid, u_s, m, \sigma)}$
$r_1 \leftarrow \mathsf{Now}() - t_0 < \Delta t$
$r_2 \leftarrow \mathsf{S.Vrf}(\mathsf{pk}_{u_s}, \sigma, sid \| u_s \| m)$
return $r_1 \wedge r_2$

Fig. 1. Procedure for verifying LogPicker messages

4.3.2 Main Protocol

The protocol between the CA and the leader is presented in Fig. 2 with the subprotocol embedded as a black box.

LogPicker Request (CA). First the CA[1] selects a log pool $U = (u_1, \ldots, u_n)$, where $u_i \in \mathcal{L}$ and a leader $u_0 \in \mathcal{L} \setminus \{u_i\}_{i\in[n]}$. The issuing CA can freely choose the leader, since the leader is assumed byzantine anyway. Afterwards the CA invokes the protocol run by sending a LogPicker request (Γ, U) with the issued certificate Γ to the leader. The leader triggers the subprotocol and runs through all four phases with the log pool, described in Sec. 4.3.3.

LogPicker Reply (Leader). Within a predefined time period the leader returns the resulting SCT and LPP to the CA, which can then deliver them along with the certificate Γ. Unresponsive leaders or protocol executions without a valid result are addressed in Sec. 4.3.4.

4.3.3 Subprotocol

In the following we describe the four phases of LogPicker's subprotocol, shown in Fig. 3. Each log in the pool will keep an transcript τ where it stores relevant messages of a protocol instance.

P1 Commit Request (Leader). The leader generates the $sid \leftarrow \mathsf{H}(u_0 \| t_0 \| \Gamma)$ from the submitted certificate Γ, the timestamp $t_0 \leftarrow \mathsf{Now}()$ and its public identifier u_0. Next it broadcasts the message

1 Note that a certificate may also be submitted by other entities like a website operator.

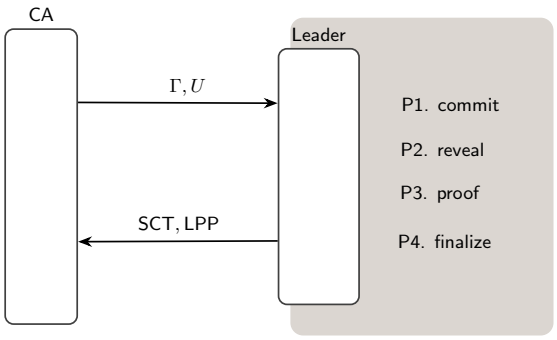

Fig. 2. The LogPicker protocol overview

$(sid, u_0, t_0, \Gamma, U, \sigma_0^1)$ to each log in the pool, where $\sigma_0^1 \leftarrow_\$ \mathsf{S.Sign}(\mathsf{sk}_{u_0}, sid \| u_0 \| t_0 \| \Gamma \| U)$.

Commit Reply (Pool). After computing the $sid \leftarrow \mathsf{H}(u_0 \| t_0 \| \Gamma)$ and verifying that $\mathsf{LP.VrfMsg}(\mathsf{pk}_{u_0}, t_0, sid, u_0, t_0 \| \Gamma \| U, \sigma_0^1) = \textbf{true}$ each log adds (t_0, u_0, Γ, U) to its transcript τ_i. Next it samples a random value $v_i \leftarrow_\$ \mathbb{Z}_{|U|}$ and creates a commitment $c_i \leftarrow_\$ \mathsf{C.Com}(\mathsf{k}[ck]_{u_i}, v_i)$. It sends the commit reply $(sid, u_i, c_i, \sigma_i^1)$ to the leader, where $\sigma_i^1 \leftarrow_\$ \mathsf{S.Sign}(\mathsf{sk}_{u_i}, sid \| u_i \| c_i)$.

P2 Reveal Request (Leader). After excluding unresponsive logs the leader fixes the remaining pool $U' = (u'_0, \ldots, u'_{n'})$, where $\{u'_i\}_{i \in [n']} \subseteq \{u_i\}_{i \in [n]}$. Next the leader verifies that $\mathsf{LP.VrfMsg}(\mathsf{pk}_{u'_i}, t_0, sid, u_i, c_i, \sigma_i^1) = \textbf{true}$. Finally the leader broadcasts the message $(sid, u_0, C, \Sigma^1, U', \sigma_0^2)$ to the remaining pool U', where C is the list of all commits, Σ^1 the corresponding signatures and $\sigma_0^2 \leftarrow_\$ \mathsf{S.Sign}(\mathsf{sk}_{u_0}, sid \| u_0 \| C \| \Sigma^1 \| U')$.

Reveal Reply (Pool). After verifying that $\mathsf{LP.VrfMsg}(\mathsf{pk}_{u_0}, t_0, sid, u_0, C \| \Sigma^1 \| U', \sigma_0^2) = \textbf{true}$ and $\mathsf{LP.VrfMsg}(\mathsf{pk}_{u'_i}, t_0, sid, u'_i, c_i, \sigma_i^1) = \textbf{true}$ each log adds (C, U') to its transcript τ_i and reveals its random value v_i by sending $(sid, u'_i, v_i, r_i, \sigma_i^2)$ to the leader, where (v_i, r_i) is the opening to its previous commitment and $\sigma_i^2 \leftarrow_\$ \mathsf{S.Sign}(\mathsf{sk}_{u'_i}, sid \| u'_i \| v_i \| r_i)$.

P3 Proof Request (Leader). The leader computes $r_1 \leftarrow \mathsf{LP.VrfMsg}(\mathsf{pk}_{u'_i}, t_0, sid, u'_i, v_i \| r_i, \sigma_i^2)$, $r_2 \leftarrow \mathsf{C.Open}(\mathsf{k}[ck]_{u'_i}, c_i, r_i, m)$ and verifies that $r_1 \wedge r_2 = \textbf{true}$. Next the leader broadcasts the message $(sid, u_0, V, R, \Sigma^2, \sigma_0^3)$ to the pool U', where (V, R) is the list of all reveals, Σ^2 the corresponding signatures and $\sigma_0^3 \leftarrow_\$ \mathsf{S.Sign}(\mathsf{sk}_{u_0}, sid \| u_0 \| V \| R \| \Sigma^2)$.

Proof Reply (Pool). Each log computes $r_1 \leftarrow \mathsf{LP.VrfMsg}(\mathsf{pk}_{u_0}, t_0, sid, u_0, V \| R \| \Sigma^2, \sigma_0^3)$, $r_2 \leftarrow \mathsf{LP.VrfMsg}(\mathsf{pk}_{u'_i}, t_0, sid, u'_i, v_i \| r_i, \sigma_i^2)$, $r_3 \leftarrow \mathsf{C.Open}(\mathsf{k}[ck]_{u'_i}, c_i, r_i, m)$ and verifies that $r_1 \wedge r_2 \wedge r_3 = \textbf{true}$. If the verification succeeded each log determines the random index v according to Eq. 1 and adds (V, R) to its transcript τ_i. Now every log creates its part of the LPP by computing $\sigma_i^3 \leftarrow_\$ \mathsf{AS.Sign}(\mathsf{sk}_{u'_i}, \tau_i)$ and thus accepts the protocol pass. Additionally the selected log u'_v creates the SCT. Finally the pool answers the proof request with the message $(sid, u'_i, \mathrm{SCT}_i, \tau_i, \sigma_i^3)$. In case of $u'_i \neq u'_v$ the SCT_i is the empty string ε, in case of $u'_i = u'_v$ the SCT_i is the SCT.

P4 Finalize (Leader). The leader also calculates the resulting index v from V according to Eq. 1. Next the leader verifies that $\forall i, j \in [n'] : \tau_i = \tau_j$. To finalize the protocol run the leader aggregates the signatures Σ^3 into one condensed aggregate signature σ via $\sigma \leftarrow_\$ \mathsf{AS.Sign}(\mathsf{k}[PK], \Sigma^3, \mathrm{T})$, where $\mathsf{k}[PK] = (\mathsf{pk}_{u'_1}, \ldots, \mathsf{pk}_{u'_{n'}})$. The leader fixes the final transcript $\tau = \tau_i$ and finalizes the protocol run by broadcasting the message $(\mathrm{SCT}, \mathrm{LPP})$ to the pool and the CA, where $\mathrm{SCT} = \mathrm{SCT}_v$ and $\mathrm{LPP} = (\tau, \sigma)$. Each log verifies whether the SCT and the LPP match the certificate Γ. They also verify whether the SCT was created by the expected log.

Field Name	Description
`version`	LPP version to allow protocol evolution
`transcript`	Transcript of the protocol run
`extensions`	List of extensions for future use
`signature`	Aggr. signature over the transcript including the certificate

Table 1. Exemplary structure of a LPP

4.3.4 Discussion

In order to proof its consent on the outcome of a LogPicker's execution, each log must sign the transcript. This consent is documented in the LPP. Analogue to an SCT, the LPP will be served to clients accompanied by the certificate in question. The exemplary structure of a LPP is shown in Tab. 1. Clients accept an LPP *iff* it meets the client's policy.

To validate a LPP, the client must verify the aggregate signature contained in the LPP. This requires the public keys of all participating logs. These keys can be distributed with the regular browser update mechanisms, as it is the case by now [26].

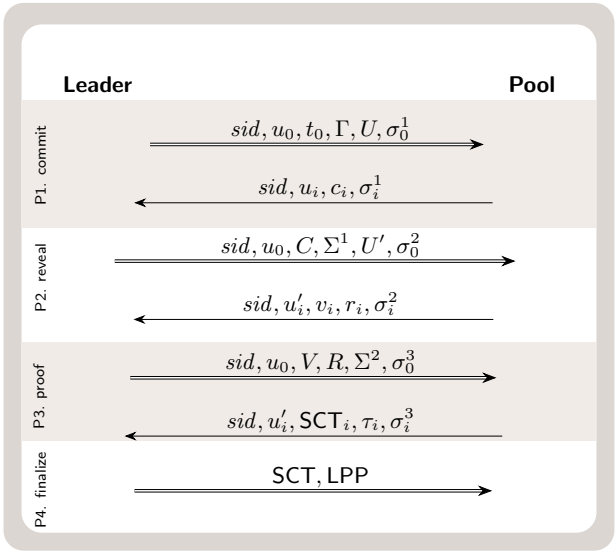

Fig. 3. Overview of LogPicker's subprotocol

Inter-Log Auditing

The fact that multiple logs witness the issuance of a certificate and corresponding SCT makes them suitable to automatically audit this result. This relieves clients and third-parties of the duty to perform auditing, which we consider critical as discussed in Sec. 2.2. The SCT sent by the leader during **P4** is the selected log's promise to include the corresponding certificate into it's history within the MMD. After the MMD has passed, each other log audits whether this promise was kept. If at least one correct log completed the LogPicker run it will discover a malicious log's behavior, as discussed in Sec. 5.1.2.

Handling Protocol Violations

In LogPicker each entity which directly takes part in the execution witnesses the certificate issuance. As soon as someone violates the protocol's specification, there are multiple witnesses who can expose it and trigger a protocol abort, if necessary. Since each message is signed by its sender, the witnesses can use this to prove the sender's misbehavior:

- Log pool and leader can check whether the CA submits a valid certificate.
- CA and log pool can observe the leader creating a valid LPP and not repeatedly exclude logs from the execution.
- CA, leader and each other log in the pool can watch the log in charge creating a valid SCT.
- Leader and log pool can inspect each other's response time, which is part of the CT policy.

However, not each protocol violation necessarily requires an abort. During the execution the leader waits for a certain time for the pool's responses to arrive. If a log's response is missing or the verification of its signatures fail, it must be excluded from the execution. If this happens before **P2** the execution can proceed without this log. If this happens up from **P2** but within the execution, it must be aborted. The leader must notify the waiting CA and other logs with detailed errors about the abort.

Syta et al. described an attacker [69], who is able to influence the result of a DR protocol by repeatedly forcing the protocol to restart until a favorable result appears by chance. As the leader is assumed byzantine in LogPicker, they have several opportunities to trigger protocol aborts. We propose the following mitigation against this attack:

- Based on the approach in [69] logs can keep count of commit requests keyed by domain. After a defined count of commits for the same domain they refuse to participate in further executions for this domain for a certain time. The logs must report the entity responsible (CA or leader).
- The covert adversary used for our threat model wants to conceal any evidence of its misbehavior, as described in Sec. 3. Punishing protocol aborts by logging them would deter an attacker from forcing protocol restarts as well.

As described in Sec. 5.1.2 if one entity repeatedly violates the protocol's specification, it must be reported and browser vendors and CAs must take actions according to their regulations, e.g., excluding the faulty entity from their trusted lists.

Note, that if the CA chooses a malicious leader, the protocols outcome still cannot be influenced: The leader cannot forge the signatures of the witnessing logs, and thus change individual votes. Repeated aborts of the protocol, if the leader is not satisfied with the vote, would be noted by the witnesses and thus be reported. Excluding individual logs from the protocol execution violates fairness, but does not influence the random selection as well.

5 LogPicker Analysis

We proceed to analyse and evaluate the efficacy of LogPicker in different experiments. Our goal is to investigate how LogPicker meets the security and design goals set in Sec. 3. We then investigate how a log's malicious

behavior is detected and how the current CT policies are applied to the LP-based PKI. In addition, we analyse the overall correctness probability of the Web PKI including its different extensions.

5.1 Analysis of LogPicker's Properties

In the following we briefly review the achievements of LogPicker according to goals described in Sec. 3.

5.1.1 Policies of the LP-based PKI

Despite the adoption of LogPicker, the lower-bound constrain on CT logs remains the same such that the general CT policy introduced in Sec. 2.1 can be respected. It only needs to be expanded with the additional constraints that arise on the logs.

Browsers can maintain their own policy (Browser LP Policy) the same way as in the CT-based PKI. Even individual browser constrains, like the logging in two logs [63], can be applied to LogPicker with only minor modifications to the protocol's execution. In addition, browser vendors can set additional requirements in their Browser's LP Policy, i.e., enforcing an LPP to be signed by at least one log from the browser's individual trusted list. This way it ensures that at least one trusted log participates in the LP run which hardens the 1-correct-log assumption. This requirement must be taken into account by the CA during pool selection and does not require any change to the protocol.

However, we encourage browser vendors to join forces and merge their policies in order to make the Web PKI less complex and provide the same security guarantees to all users on the Web. Therefore we support Chrome's decision for dropping a part of their CT policy, which required one of the logs in charge to be operated by Google [52].

5.1.2 Detection of Rogue Certificates

If a CT log wants to support a malicious CA to carry out the attack described in Sec. 3.1 they must hide the corresponding rogue certificate from their public log. Since in the LP-based PKI a user's browser enforces each certificate to be accompanied by a valid LPP, it leads to inter-log auditing by correct logs. This poses a high risk for the attacker to be detected. However, hiding certificates from public logs is a violation of the CT specification, such that the attacker must serve each auditing log that participated in the appropriate LogPicker run a view containing the rogue certificate. If at least *one correct log* participates in the LP run, after MMD it will audit the malicious logs promise to include the rogue certificate. If the rogue certificate is missing in the public view, this violation is detected by at least one correct log. As a consequence the malicious log will be forced to include the rogue certificate in its public view and can therefore be found by monitors.

The infrastructure introduced by LogPicker allows the *interplay of auditing and monitoring*. This gives further options to detect and to mitigate split-view attacks against monitors as well. E.g., logs participating in a LogPicker run that target the domain of a monitor can notify it about the run in progress and even prove their participation by presenting the LPP. This would cause the malicious log to be monitored by the domain owner's monitor and result in the detection of the rogue certificate. The communication between the log and the monitor, however, requires a secure communication channel. Establishing such a channel is a non-trivial task in practice and thus we defer its concrete implementation to future work.

5.1.3 Analysis of Security Goals

LogPicker strengstens the Web PKI against attacks resulting from the collaboration of malicious CAs and CT logs. If this attack is foiled by the random log selection, it will prevent the CA from influencing the log. If the attacker gets lucky and a malicious log is selected by chance, the *interplay of auditing and monitoring* mitigates split view attacks. Both scenarios pose a high risk for the attacker, which they are not willing to take (Sec. 3.2) (**SG1**).

By design LogPicker tolerates byzantine CAs and can handle a high number of byzantine logs (**SG2**). Concretely, it is robust against collaboration of malicious entities among the Web PKI, as introduced in Sec. 3.1.

Client-side auditing is not required using LogPicker since it is performed by the logs. Firstly, this preserves the clients privacy (**SG3**). Secondly, logs are best suited for this task due to their involvement in the certificate issuance process.

Clients can verify the aggregated signature using the LPP served alongside the certificate. There is no need to rely on trust assumptions (**SG4**) since the LPP attests the successful execution of LogPicker.

5.1.4 Analysis of Design Goals

To validate the authenticity of a LPP, the user's browser does only need to verify the aggregate signature contained in the LPP. LogPicker thus does not introduce any additional network requests on loading a website and thus conforms to **DG1**.

LogPicker does not require logs to synchronize with others, such that they remain independent (**DG2**).

Implementations of LogPicker can be scaled efficiently across multiple servers and thus keep up with the expected growth of the Web PKI. Our prototype in Sec. 6 shows that LogPicker can even handle possible future loads of the Web (**DG3**).

The deployment of LogPicker requires only changes to CT logs and browsers (**DG4**), which we argue to be realistically achievable. The LP-based PKI is downwards compatible, such that outdated clients can still be used.

5.2 Probabilistic Analysis

In order to examine the effectiveness of LogPicker we need to analyze the likelihood of a future compromise of this protocol in the context of the Web PKI. For this purpose we use the probabilistic approach as applied to the CA-based PKI by Oppliger [54].
We define an entity of the Web PKI, e.g., a CT log or a CA, as *correct* if it always behaves according to its specification. An entity with arbitrary behavior is assumed *byzantine*. A system of entities like the CA-based PKI is assumed correct, if it behaves correct as a *whole*, even in the presence of some byzantine entities. In this chapter we present only the results of the analysis and refer to Appendix. B and [54] for more details.

5.2.1 CA-Based PKI

Fig. 4 shows the correctness probability of the CA-based PKI taking into account the correctness probability of each individual CA, denoted by p and the number of all existing CAs in the system, denoted by n_{CA}.
Recall that most browsers include less than 200 CAs in their trust list [6, 27] and this number does not even include intermediate CAs, which are able to issue certificates the same way as root CAs. As shown in Fig. 4 the correctness of the CA-based PKI diminishes quickly. Even by assuming a high value of $p = 0.99$ this results in a correctness probability for the whole system of less than 0.2 which we consider a critical value for the trustworthiness of the CA-based PKI.

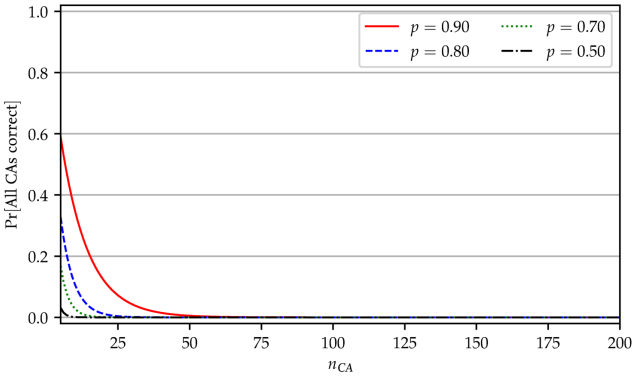

Fig. 4. Probability that all CAs are correct as defined in Eq. 2 over a number of CAs n_{CA}

5.2.2 CT-Based PKI

In the CT-based PKI browsers enforce the logging of certificates to CT logs, whose purpose is to make CA misbehavior public. In addition to n_{CA} the CT-based PKI adds a number of CT logs, denoted by n_{log}, from which the issuing CA is free to choose one. This way incorrect behavior of a CA is detected if the chosen log is correct and includes the CA's rogue certificate into its public history.
Fig. 5 shows the correctness probability of the CT-based PKI by utilizing a ratio r between CAs and CT logs participating in the CT-based PKI with $r = \frac{nca}{n_{log}}$. To approximately reflect the reality at the time of this writing we chose $r = 5.67$, i.e. $n_{CA} = 0.85 \cdot n$ and $n_{log} = 0.15 \cdot n$. An improvement from the CT-based PKI to the previous one is noticeable: For $p = 0.99$ and $n = 200$ the value has increased from 0.20 to 0.80. However, this requires a high level of trust in the individual CAs: with $p = 0.90$ the correctness probability quickly approaches zero. In addition Fig. 9 shows how varying the ratio influence the correctness probability by assuming a fixed $p = 0.99$.

5.2.3 Gossip-Based PKI

The use of CT Gossip adds another layer of fallback by introducing the probability that all websites are gossiping. Even if a CA and the log in charge are compromised, mississued certificates can be detected if the visited website is gossiping. If Gossip is rolled-out partially, only the communication with the corresponding websites is protected, the Web as a whole remains vulnerable. Thus Gossip's contribution to the system's correctness probability depends on each individual website's probability for gossiping.

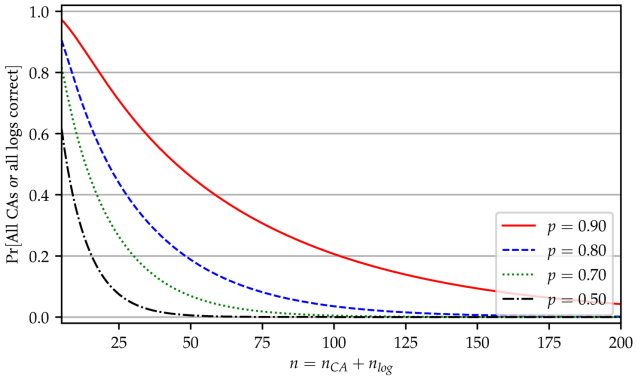

Fig. 5. Probability that all CAs or all logs are correct as defined in Eq. 4 over number of entities n with a fixed ratio $r \approx 5.67$ with $r = \frac{n_{CA}}{n_{log}}$

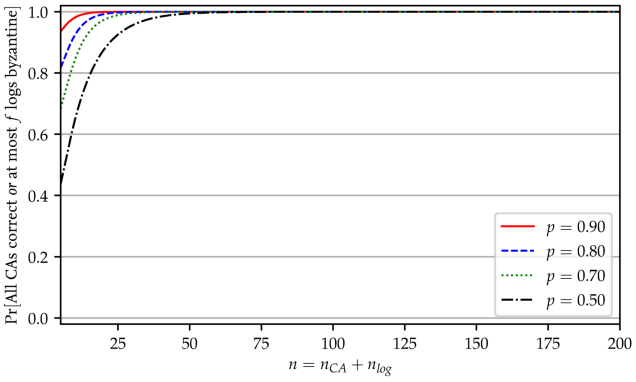

Fig. 6. Probability that all CAs are correct or at most $f = n_{log} - 1$ logs are byzantine as defined in Eq. 7 over number of entities n with a fixed ratio $r \approx 5.67$ with $r = \frac{n_{CA}}{n_{log}}$

Due to the large number of existing websites the probability that *all* websites are gossiping is close to zero such that Gossip's contribution to the correctness of the CT-based PKI is negligible. Thus we assume the correctness probability for the resulting Gossip-based PKI to be the same as for the CT-based PKI.

LP-Based PKI

In order to guarantee an unpredictable outcome at least one correct CT log must contribute to the random log selection. For this analysis we assume all logs to participate in each LogPicker run. The maximum number of byzantine logs f that can be handled by LogPicker is denoted by $f = n_{log} - 1$. For simplicity we use a coin toss to decide for each individual log whether it is correct or not. We consider it an unrealistic scenario to assume such a high number of logs to be byzantine. Nevertheless, we analysed the LP-based PKI using this lower bound in order to show that even in this worst case scenario LogPicker is able to maintain a high correctness probability.

The result shown in Fig. 6 represents the overall correctness probability for the LP-based PKI applied for different probabilities of correctness of the participating CAs. A significant improvement compared to the previously analysed systems can be seen. For any value of $p > 0.25$ the probability approaches 1 for $n > 50$. This results in a correctness of almost 100 % for each successful execution of LogPicker. In addition Fig. 10 shows that varying the ratio has little influence on the correctness probability of the PL-based PKI, even by assuming a low $p = 0.5$.

6 Prototype

To ensure compliance with **DG3** from Sec. 3.4 we demonstrate that LogPicker does not impose unacceptable delays to certificate issuances for the current scale of the Web PKI and beyond. We therefore prototyped LogPicker[2] to show it meeting the aforementioned design goals. Our prototype covers all phases described in Sec. 4.3.2. Its implementation is described in more detail in Apx. A.

We performed the experiment on a cluster consisting of 9 nodes, each equipped with two Intel Xeon E5-2640v4 CPUs and 64 GB of RAM, running CentOS 7. We ran 15 instances of our prototype on each node.

At the time of writing, Chrome trusts ≈ 30 usable CT logs [26]. As we expect the CT ecosystem to grow further, our experiment described in the following highlights that LogPicker does not impose constraints on the future growth of the Web PKI.

Performance Characteristics

According to **DG3** LogPicker must not slow down the certificate creation significantly. As stated by Schoen customers of Let's Encrypt "[...] should be able to get a new cert [...] in one minute" [61]. Fig. 7 shows the runtime of the LogPicker protocol to issue one certificate with an increasing number of logs. In our experiment the client submits 1000 certificates to a leader process for varying numbers of logs. We measured the time each certificate took from reaching the leader process to the completion of the protocol and averaged the results. Under conditions roughly four times the size of the cur-

2 The prototype is available under https://logpicker.github.io/.

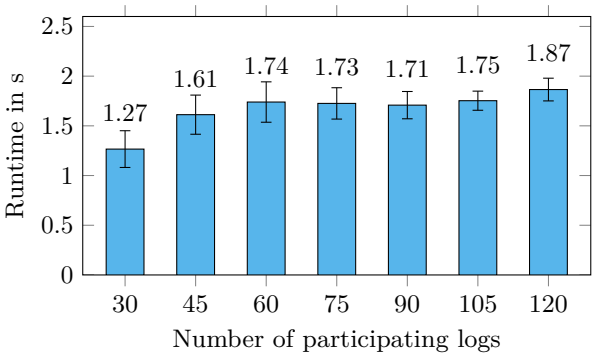

Fig. 7. Time taken to complete one LogPicker run

rent Web PKI's scale, the additional time required by LogPicker is still below 2 s, a delay we consider to be acceptable.

The LogPicker protocol requires the leader to receive $1 + 3 \cdot n_{log}$ and send $1 + 4 \cdot n_{log}$ messages for each run. The number of processed messages thus scales linearly with the size of the log pool. Each participating log receives 4 messages and sends 3 messages per LogPicker run. The communication complexity is therefore linear for the leader and constant for each log. This matches the results shown in Fig. 7 where the required time scales roughly linear with the number of participating logs.

As the participating logs do not need to perform any kind of synchronization between concurrent LogPicker runs, the issuance speed is the main performance criteria. Additional measurements regarding the throughput of our prototype are provided in Apx. A, Fig. 8.

7 Limitations and Future Work

LogPicker adds additional complexity to the issuance process of certificates. This is largely confined to CT logs, which must implement the core of the protocol. In addition, CAs must modify their issuance process by taking the detour via LogPicker instead of directly submitting certificates to their favorite logs. Although due to our experiments we consider the resulting slowdown to be acceptable, we point out that there is some room for extensions.

Monitoring still suffers from several problems, as shown 2019 by Li et al. [44]. The interplay of auditing and monitoring allows the incorporation of pro-active notifications during certificate creation. This can be done for example by utilizing *DNS txt records* [20] that stores a domain owner's callback URL. Correct logs can use it to send a notification about each certificate creation attempt targeting this domains during the protocol's first phase. This way protocol aborts by a malicious leader are reported automatically. As soon as a leader becomes malicious participating logs can force a protocol abort. However, this requires the establishment of a new LogPicker round with a new leader, which slows down the generation time of the certificate. A proof of the leader's misbehavior could help to fasten up protocol restarts by e.g. reusing parameters from the previous round.

As soon as a rogue certificate is discovered during e.g., auditing it must be revoked, which is still a problem of active research. Combining LogPicker with other solutions, e.g., Laurie and Kasper's idea of Revocation Transparency [42] may lead to promising new approaches.

The protocol's cryptographic building blocks, e.g., Aggregate Signatures, provide a perfect fit for the design of the protocol. As with several other protocols, however, other designs and combinations of cryptographic primitives are also possible and might provide similar results.

8 Conclusion

LogPicker enhances the privacy of web users by making auditing of certificates more effective. Instead of individual trust anchors, the protocol enables collaboration of CT logs in order to withstand byzantine CA and CT logs, likewise.

The adoption of LogPicker enhances the protection of a user's privacy on the internet by hardening the Web PKI against attacks resulting from collaboration of a byzantine CA and CT logs. In addition, LogPicker makes auditing more practicable by involving witnesses in the issuance process of certificates.

We justified our threat by reviewing the history of certificate mississuance and the weaknesses of the current Web PKI and its alternative approaches. From this we derived our attacker model and the goals required to protect the Web PKI. To create a feasible solution, we identified additional protocol design goals. We introduced the LogPicker protocol, which extends the CT-based PKI to comply with our goals and presented its building blocks. Our probabilistic analysis shows a significant improvement of the correctness probability of the LP-based PKI compared to others. To get a sense of how LogPicker performs, we developed a prototype of our solution and presented its promising results. Finally, we reviewed LogPicker's limitations and the remaining challenges.

The design of LogPicker builds on two general concepts for improving security: collaboration and reduced complexity. We believe that this combination does not only help to protect the Web PKI from certificate misissuance but ultimately provides a general basis for managing and monitoring trust in the internet. Consequently, we conjecture that variants of LogPicker might also be applicable in other scenarios where multiple parties need to audit themselves, such as in distributed authentication and application mash-ups.

Acknowledgments

First of all we thank Florentin Rochet who, with a lot of persistence, helped us to improve the paper in his function as shepherd. Furthermore we thank Ralf Oppliger for his feedback on the probabilistic analysis and Heinrich Behle for his constructive criticism of the manuscript. We acknowledge funding by the Federal Ministry of Education and Research of Germany in the framework of KIWI (16KIS1145) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972

References

[1] 2016. Secure Logging Schemes and Certificate Transparency. *Computer Security – ESORICS 2016. ESORICS 2016. Lecture Notes in Computer Science* (2016).

[2] 2019. How Certificate Transparency Works. https://www.certificate-transparency.org/how-ct-works

[3] 2020. CA/Browser Forum. https://cabforum.org/

[4] 2020. CT2 Log Compromised via Salt Vulnerability. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/aKNbZuJzwfM

[5] Apple. 2019. Apple's Certificate Transparency policy. https://support.apple.com/en-us/HT205280

[6] Apple. 2020. List of available trusted root certificates in iOS 12, macOS 10.14, watchOS 5, and tvOS 12. https://support.apple.com/de-de/HT209144

[7] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. [n. d.]. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[8] Yonatan Aumann and Yehuda Lindell. 2010. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology* 23, 2 (2010), 281–343.

[9] Andrew Ayer. 2018. How will Certificate Transparency Logs be Audited in Practice? https://www.agwa.name/blog/post/how_will_certificate_transparency_logs_be_audited_in_practice

[10] Andrew Ayer. 2018. Timeline of Certificate Authority Failures. https://sslmate.com/certspotter/failures

[11] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2014. ARPKI: Attack Resilient Public-Key Infrastructure. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14* (2014).

[12] Enrico Bocchi, Luca De Cicco, and Dario Rossi. 2016. Measuring the quality of experience of web users. *Computer Communication Review* 46, 4 (2016), 8–13.

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*.

[14] Matthieu Bussiere and Marcel Fratzscher. 2008. Low probability, high impact: Policy making and extreme events. *Journal of Policy Modeling* 30, 1 (2008), 111–121.

[15] Sergej Chernov. 2015. Implement Certificate Transparency support (RFC 6962). https://bugzilla.mozilla.org/show_bug.cgi?id=1281469

[16] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. 2015. Efficient gossip protocols for verifying the consistency of Certificate logs. *2015 IEEE Conference on Communications and NetworkSecurity, CNS 2015* (2015).

[17] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard).

[18] Peter Eckersley. 2012. *Sovereign Key Cryptography for Internet Domains*. Technical Report.

[19] C. Evans and C. Palmer. 2011. Public Key Pinning Extension for HTTP. https://datatracker.ietf.org/doc/rfc7469/

[20] C. Evans, C. Palmer, and R. Sleevi. 1993. RFC1464: Using the Domain Name System To Store Arbitrary String Attributes. *IETF RFC* (1993). https://doi.org/10.17487/RFC7469

[21] CA/Browser Forum. 2019. *Guidelines For The Issuance And Management Of Extended Validation Certificates*. cabforum.org. https://cabforum.org/wp-content/uploads/CA-Browser-Forum-EV-Guidelines-v1.7.1.pdf.

[22] CA/Browser Forum. 2020. *Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates*. cabforum.org. https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.6.8.pdf.

[23] Eva Galperin, Seth Schoen, and Peter Eckersley. 2013. A Post Mortem on the Iranian DigiNotar Attack. https://www.eff.org/de/deeplinks/2011/09/post-mortem-iranian-diginotar-attack

[24] Artyom Gavrichenkov. 2015. Breaking HTTPS with BGP hijacking. *Black Hat. Briefings* (2015).

[25] Oded Goldreich. 2006. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA.

[26] Google. 2020. Certificate Transparency - Known Logs. https://www.certificate-transparency.org/known-logs

[27] Google. 2020. Google Root Store: 2020-10-21 - Proposed. https://docs.google.com/spreadsheets/d/e/2PACX-1vQ7Jtb4NxCSaEtCaisz2u3NQZcHejDUjI3Q-utBnL-C5E7w4crv6QZ9GRDb2bFGbLgUQsgQyF0Y8eoN/pubhtml

[28] Google. 2020. Transparency report: HTTPS encryption on the web (2020-01-23). https://transparencyreport.google.com/https/overview?hl=en

[29] Charles Miller Grinstead and James Laurie Snell. 2012. *Introduction to probability*. American Mathematical Soc.

[30] P. Hallam-Baker and R. Stradling. 2013. RFC6844: NS Certification Authority Authorization (CAA) Resource Record. *IETF RFC* (2013).

[31] B. Hof. 2017. STH Cross Logging. *IETF RFC draft* (2017). https://tools.ietf.org/id/draft-hof-trans-cross-00.html

[32] P. Hoffman and J. Schlyter. 2012. RFC6698: The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. *IETF RFC* (2012).

[33] R. Housley and K. O'Donoghue. 2017. Problems with the Public Key Infrastructure (PKI) for the World Wide Web. *IETF Draft* (2017). https://tools.ietf.org/html/draft-iab-web-pki-problems-01

[34] David Huang and Brad Hill. 2016. Early Impacts of Certificate Transparency. https://www.facebook.com/notes/protect-the-graph/early-impacts-of-certificate-transparency/1709731569266987/

[35] Kazakhtelecom JSC. 2015. Kazakhtelecom JSC notifies on introduction of National security certificate from 1 January 2016. https://web.archive.org/web/20151202203337/http://telecom.kz/en/news/view/18729/

[36] J. Katz and Y. Lindell. 2014. *Introduction to Modern Cryptography, Second Edition*. Taylor & Francis.

[37] S Kent. 2018. Attack and Threat Model for Certificate Transparency. *Internet Engineering Task Force* (2018).

[38] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* (1982), 382–401.

[39] Adam Langley. 2013. Fraudulent Digital Certificates Could Allow Spoofing. https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html

[40] Adam Langley. 2013. Further improving digital certificate security. https://security.googleblog.com/2013/12/further-improving-digital-certificate.html

[41] Ben Laurie. 2014. Certificate Transparency. *ACM Queue* 8 (2014).

[42] Ben Laurie and Emilia Kasper. 2012. Revocation transparency. *Google Research, September* (2012).

[43] B. Laurie, A. Langley, and E. Kasper. 2013. RFC6962: Certificate Transparency. *IETF RFC* (2013).

[44] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. 2019. Certificate transparency in the wild: Exploring the reliability of monitors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*.

[45] Wouter Lueks and Ian Goldberg. 2015. Sublinear Scaling for Multi-Client Private Information Retrieval. In *Financial Cryptography and Data Security*, Rainer Böhme and Tatsuaki Okamoto (Eds.). 168–186.

[46] Gervase Markham. 2016. Incidents involving the CA WoSign. https://groups.google.com/forum/#!topic/mozilla.dev.security.policy/k9PBmyLCi8I%5B1-25%5D

[47] M. Marlinspike and T. Perrin. 2013. Trust Assertions for Certificate Keys. *IETF Draft* (2013).

[48] Mozilla. 2019. Mozilla takes action to protect users in Kazakhstan. https://blog.mozilla.org/blog/2019/08/21/mozilla-takes-action-to-protect-users-in-kazakhstan/

[49] Johnathan Nightingale. 2011. Revoking Trust in DigiCert Sdn. Bhd Intermediate Certificate Authority. https://blog.mozilla.org/security/2011/11/03/revoking-trust-in-digicert-sdn-bhd-intermediate-certificate-authority/

[50] L. Nordberg, D. Gillmor, and T. Ritter. 2018. Gossiping in CT. *IETF Draft* (2018). https://tools.ietf.org/html/draft-ietf-trans-gossip-05

[51] Devon O'Brien. 2018. Certificate Transparency Enforcement in Chrome and CT Day in London. https://groups.google.com/a/chromium.org/d/msg/ct-policy/Qqr59r6yn1A/2t0bWblZBgAJ

[52] Devon O'Brien. 2020. Chrome CT 2020 Plans. https://groups.google.com/a/chromium.org/g/ct-policy/c/dqFtoFBy8YU/m/Xa67FWVCEgAJ

[53] Lukasz Olejnik, Claude Castelluccia, and Artur Janc. 2012. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2012)*. Vigo, Spain. https://hal.inria.fr/hal-00747841

[54] Rolf Oppliger. 2014. Certification authorities under attack: A plea for certificate legitimation. *IEEE Internet Computing* (2014).

[55] Serguei Popov. 2017. On a decentralized trustless pseudo-random number generation algorithm. *Journal of Mathematical Cryptology* (2017).

[56] J.R. Prins. 2011. *DigiNotar Certificate Authority breach "Operation Black Tulip"*. Technical Report. Fox-IT, Delft.

[57] Ram Sundara Raman, Leonid Evdokimov, Eric Wurstrow, J Alex Halderman, and Roya Ensafi. 2020. Investigating Large Scale HTTPS Interception in Kazakhstan. In *Proceedings of the ACM Internet Measurement Conference*. 125–132.

[58] Tom Ritter. 2016. a bit on certificate transparency gossip. https://ritter.vg/blog-a_bit_on_certificate_transparency_gossip.html

[59] Mark D. Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2014.23379

[60] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. 2013. RFC6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. *IETF RFC* (2013). https://tools.ietf.org/html/rfc6960

[61] Seht Schoen. 2015. Please support wildcard certificates [Online discussion group]. https://community.letsencrypt.org/t/please-support-wildcard-certificates/258/19

[62] Ryan Sleevi. 2016. Announcement: Requiring Certificate Transparency in 2017. https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/78N3SMcqUGw/yklwHXuqAQAJ

[63] Ryan Sleevi. 2016. *Certificate Transparency in Chrome*. Technical Report. https://groups.google.com/g/mozilla.dev.security.policy/c/VJYX1Wnnhiw/m/ecenP98wBgAJ

[64] Ryan Sleevi and Eran Messeri. 2017. *Certificate Transparency in Chrome: Monitoring CT logs consistency*. Technical Report. Google. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZIZe_z-ls/edit

[65] Christopher Soghoian and Sid Stamm. 2010. Certified Lies: Detecting and defeating government interception attacks against SSL. In *Proceedings of ACM Symposium on Operating Systems Principles*. 1–18.

[66] Stephan Somogyi. 2015. Improved Digital Certificate Security. https://security.googleblog.com/2015/09/improved-

digital-certificate-security.html
[67] Sooel Son and Vitaly Shmatikov. 2010. The hitchhiker's guide to DNS cache poisoning. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering* (2010).
[68] Nick Sullivan. 2018. Introducing Certificate Transparency and Nimbus. https://blog.cloudflare.com/introducing-certificate-transparency-and-nimbus/
[69] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, and Nicolas Gailly. 2017. Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy*.
[70] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping Authorities 'Honest or Bust' with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy*. 526–545.
[71] Tom Van Goethem, Ping Chen, Nick Nikiforakis, Lieven Desmet, and Wouter Joosen. 2017. Large-scale security analysis of the web: Challenges and findings. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8564 LNCS (2017), 110–126.
[72] Jeremy Wagner. 2020. Why Performance Matters. https://developers.google.com/web/fundamentals/performance/why-performance-matters
[73] Dan Wendlandt, David G. Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. *USENIX Annual Technical Conference* (2008).
[74] Jiangshan Yu, Vincent Cheval, and Mark Ryan. 2016. DTKI: A new formalized PKI with verifiable trusted parties. (2016), 1695–1713.
[75] Jiangshan Yu and Mark Ryan. 2017. Evaluating Web PKIs. *Software Architecture for Big Data and the Cloud* (2017).
[76] Bryant Zadegan and Ryan Lester. 2016. Abusing Bleeding Edge Web Standards for AppSec Glory. In *DEF CON 24*.
[77] Torsten Zimmermann, Jan Ruth, Benedikt Wolters, and Oliver Hohlfeld. 2017. How HTTP/2 pushes the web: An empirical study of HTTP/2 server push. In *2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops*.

9 Appendix

A Details on Prototype

We implemented LogPicker as a C++ prototype, that performs each phase of the protocol. However, our prototype does not model the interaction with the actual CT log software, i.e., once the leader receives the SCT and LPP our prototype concludes the LogPicker run. We use RSA signatures with keylengths of 2048 bit from the RELIC toolkit [7] to verify the authenticity of the exchanged messages. In **P4** we use BLS signatures with aggregation based on *bls-signatures*[3] for the creation of the LPP. The TCP communication of the protocol's participants was realized with *msgpack-RPC* using the *rpclib*[4] library.

We calculated the throughput shown in Fig. 8 based on the elapsed time at the leader for processing 500 certificates. Please note that to increase the throughput the log's operator can simply deploy more LogPicker processes and distribute the workload via a regular load balancer.

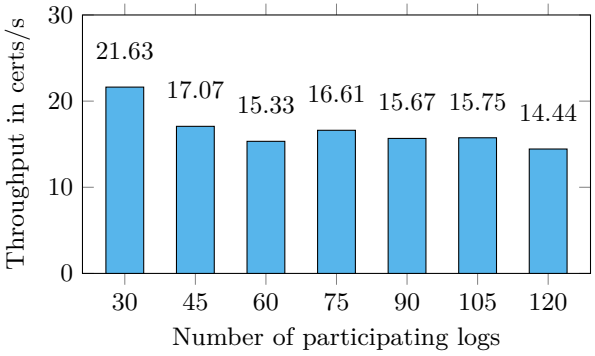

Fig. 8. Certificate Throughput

B Details on Probabilistic Analysis

Since the correctness probability for each entity in a system is hard to determine, Oppliger chooses [54] an equal correctness probability for all entities in the system in order to make them comparable at all.
In our work we apply his approach to the subsequent evolutions of the Web PKI including LogPicker. We are aware of the fact that this procedure does not reflect the real behavior of the system. However, it is still valuable as it represents the worst case behavior for each system equally.

CA-based PKI

The number of all CAs in the CA-based PKI is denoted by n_{CA}. The attacker can choose which CA they wish to attack, such that the CA-based PKI is only correct if each CA is correct, even the *weakest* one. This implies that the correctness of the CA-based PKI is dependent on the correctness of the weakest CA, denoted by p. Using this assumption, Eq. 2 expresses the correctness probability for the whole CA-based PKI.

3 https://github.com/Chia-Network/bls-signatures
4 https://github.com/qchateau/rpclib

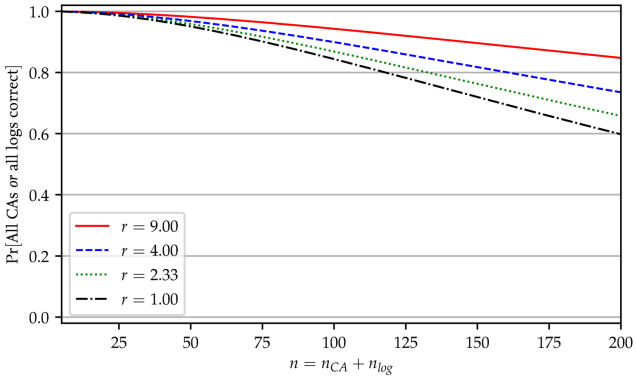

Fig. 9. Probability that all CAs or all logs are correct for $p = 0.99$ and a variable ratio $r = \frac{n_{CA}}{n_{log}}$

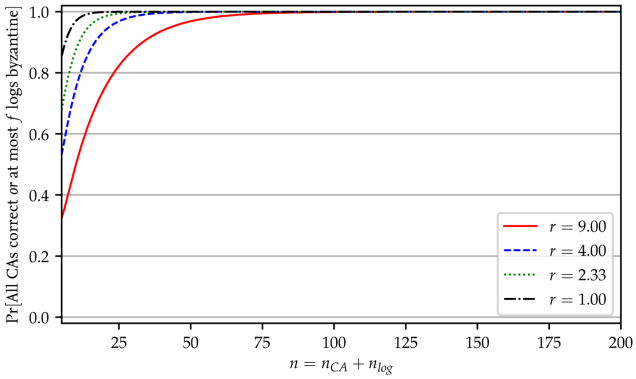

Fig. 10. Probability that all CAs are correct or at most f logs byzantine for $p = 0.50$ and a variable ratio $r = \frac{n_{CA}}{n_{log}}$

$$\Pr[\text{all CA correct}] = \prod_{i=1}^{n_{CA}} p = p^{n_{CA}} \tag{2}$$

CT-based PKI

During certificate creation CAs can freely choose the log that shall record their certificates. The same way as for CA-based PKI the attacker can choose the weakest log to attack from the set of all participating CT logs. Again this implies the set of all logs is correct if the weakest log in this set is correct as well, defined by Eq. 3.

$$\Pr[\text{all logs correct}] = p^{n_{log}} \tag{3}$$

In the CT-based PKI incorrect behavior is detected if at least one correct log publishes this misbehavior in its history. Taking into account the assumptions above it can be assumed that the CT-based PKI is correct if either even the weakest CA is correct *or* the weakest log is correct. The correctness probability for the CT-based PKI yields in Eq. 4.

$$\Pr[\text{all CA} \vee \text{all log corr.}] = 1 - (1 - p^{n_{CA}})(1 - p^{n_{log}}) \tag{4}$$

LP-based PKI

For each log in the log pool a coin toss decides whether this log behaves correct or not, which can be modeled as a Bernoulli Trial B, described by a binomial distribution [29].

The number of logs in a log pool is denoted by n_{log}. The probability for each of those logs being correct is p, the counter-event is denoted by $\overline{p}$. Let k be a number with $k < n_{log}$. The probability that *exactly* k logs are byzantine, is denoted by:

$$\Pr[\text{Exactly } k \text{ logs byzantine}] = B(\overline{p}, n_{log}, k) \tag{5}$$

We seek the probability that *at most* f logs are byzantine, by inserting each possible probability outcome for $k \leq f$ from Eq. 5 and adding them up:

$$\Pr[\text{At most } f \text{ logs byzantine}] = \sum_{k=0}^{f} B(f, n_{log}, k) \tag{6}$$

Hence, the overall correctness probability for the LP-based PKI results from the probability that *all CAs are correct* or *at most f logs are byzantine*:

$$\begin{aligned} &\Pr[\text{all CA correct} \vee \text{at most } f \text{ logs byzantine}] \\ &= 1 - (1 - \Pr[\text{all CA correct}])(1 - \Pr[\text{at most f logs byz}]) \\ &= 1 - (1 - p^{n_{CA}}) \left(1 - \sum_{k=0}^{f} B(f, n_{log}, k)\right) \end{aligned} \tag{7}$$