Subversion Transparency through Blockchain

Siddharth Malhotra 934336
Vedant Chauhan 892758
Nikhil Chitale 892109

A report submitted in fulfilment of the requirements for COMP90043 of Research Project in the Team 75 Cryptography and Security

October 23, 2018

Contents

1 Contribution

In this research project, we have

- Siddharth Malhotra
  - Problem identification and Conceptualization: Analyzed techniques used and problems faced with current TLS/SSL public key encryption. Applied these learning's to architecture a plausible approach towards software release.
  - Verifiable Sharing Scheme and Feldman's Scheme: Analyzed Feldman's Scheme and discussed how this technology can be used to solve the privacy issue of binary transparency.
  - Software lifecycle and Evaluation: Designing subversion transparency to build a structured flow. Then, performing a comparative analysis to existing approaches and critically analyzing components.
- Vedant Chauhan
  - Current system: The working of current system of binary transparency is analyzed and mentioned thoroughly with the help of diagram.
  - Verifiable Sharing Scheme and Feldman's Scheme: Analyzed and explained the existing technology of Feldman's Scheme with an example and discussed how this technology can be used to solve the privacy issue of binary transparency.
  - Proposed System and Design Goals: Integrated the existing technology of blockchain and Feldman's Scheme to solve the issues of binary transparency. In addition discussed the goals of the said proposed system and focused on software release life cycle.
- Nikhil Chitale
  - Fundamental concepts: The working of current system of binary transparency, blockchain and certificate transparency is analyzed and mentioned thoroughly with the help of diagram.
  - Evaluating past security attacks: Thoroughly studying the past security attacks and understanding why they failed so as to incorporate the known issues to the system and design.

2 Overview and Motivation

In the past, there have been several malware attacks such as Petya and Flame. With the recent rise in Ransomware based attacks, there is a growing need to redesign the way we release software currently. We propose and design subversion transparency which aims to ensure that all parties are provided with the same binaries through secret checking and targeted parties are not subjected to malicious software.

Subversion transparency is achieved through amalgamation of blockchain, altering fundamental concepts adapted from certificate transparency and employs Feldmans verifiable sharing scheme for anonymous auditing. Through this research report, we explore the cost of some of the current attacks. Finally, we compare existing approaches to our proposed system.

3 Introduction

3.1 BlockChain

A blockchain is a decentralized, distributed public ledger which records transactions in chronological order across many computers. The transaction record can be altered only by the consensus of the network or if all the subsequent blocks are altered in the blockchain. The copy of blockchain is downloaded automatically to each node which allows the participants to keep track of the transaction without any central recordkeeping.

3.2 X.509 Certificates

Public key certificates have a robust sets of requirements, these requirements are identified by X.509 public key infrastructure (PKI) standards. So X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. It is based on the use of public-key cryptography and digital signatures and each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. Now, if the CA verifies requester's identity, it then encrypts or signs, then encodes, and issues the respective certificate.

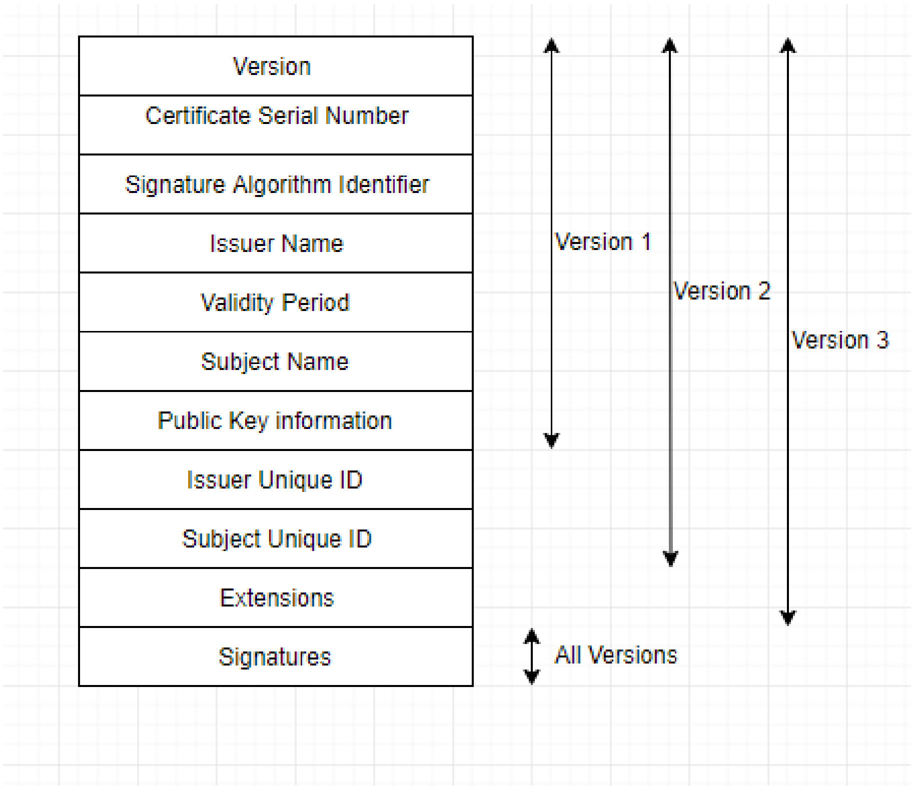

Figure 1: X.509 Certificate

3.3 Certificate Transparency

The certificate authority (CA) is a trusted third party that issues us digital signature. The current cryptographic mechanisms cannot detect if certificate issued by CA are compromised or have gone rogue. Personal data and information of the users were compromised, so Certificate Transparency provides us with an elegant defense which is being actively deployed by web browser vendors and CAs to limit the security breach incidences.

Certificate Transparency is open framework for monitoring the TLS/SSL certificate system and auditing specific TLS/SSL certificates.

The framework consists of three main components:

- Certificate logs: Certificate logs maintain append-only, publicly auditable and cryptographically assured records of all the certificates. These certificates are generally submitted by the certificate authorities. We query the log to check if the certificate is logged or check if the log is behaving correctly.
- Monitors: Monitors are publicly run servers that watch for suspicious certificates and periodically contact all of the log servers. These servers are run by various companies and government, also there are subscription services that domain owners and certificate authorities buy into.
- Auditors: Auditors is an integral component of a browsers TLS client that verify if the logs are cryptographically consistent and behaving correctly and check if particular certificate appears in a log. In case of unregistered or suspected certificates in the log, TLS refuses the connection.

When a CA wishes to issues a certificate in CT log it sends the certificate data to the log called a pre-certificate. The log then generates a Signed Certificate Timestamp (SCT) using a secret signing key which acts as a promise that the log will add the certificate to its public log within a specific time. This SCT is sent back to the CA and the CA typically embeds it in the final signed certificate. Two types of entities ensure that CAs and log servers properly follow CT procedures which are monitors and auditors. The way certificate transparency works are all the major browser only trust the certificate which comes with the proof that it was recorded in the CT public log. So if a rogue certificate appears on one of the public logs, the administrator monitoring these log can detect and revoke the certificate.

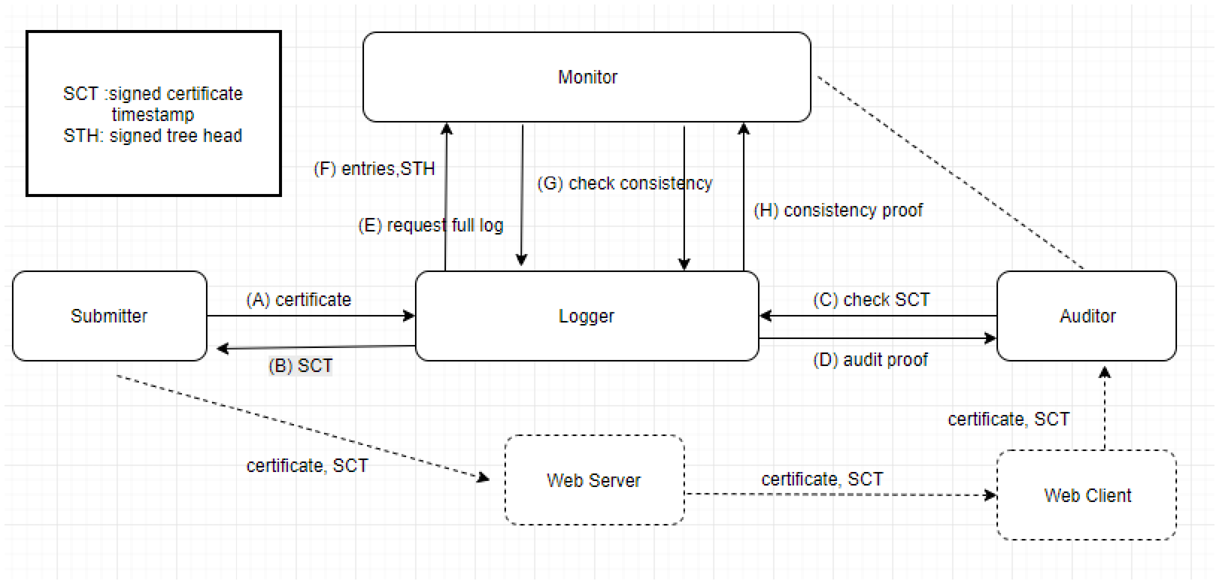

Figure 2: Overview of interactions between entities in Certificate Transparency

3.4 Binary Transparency

When we download or update a software there is no assurance that the download or update corresponds to the software source code. There have been examples like in 2012, NSA malware called Flame used rogue Microsoft binary signing certificate to infect users via Windows update. So Binary transparency in a way provides us with the assurance to guard us against the large-scale shipment of malware delivered by bad updates or downloads. The way binary transparency works is:

a) Ensure that the download binaries are logged in public verify log.

b) When downloading verify that the update has been logged before accepting and installing them.

Some of the terminology used in binary transparency are:

- Binary: Build process results with a specific executable file or update file.
- Release: In a given version of the software, set of all the binaries.
- Merkle Tree: This is a cryptographic data structure which helps in the creation of certain artifacts from input set:
  - Merkle tree root head is used to represent the set which is a single hash value.
  - Total number of hash value should be equal to size of the set of log given by inclusion proof.

3.5 Malware Security Attacks

- Flame cyber espionage malware: In 2012, hackers generated code- validated certificate signed by Microsoft by exploiting the flaw in Microsofts certificate authority terminal services. Thus exploiting the windows update mechanism. The Flames web server could detect windows update and send a downloader disguised as a legitimate update from the windows. Thus Flames could infiltrate Microsoft services and servers to force feed malicious files to users.

- Petya malware: In 2016, a malware attack exploded out of Ukraine and paralyzed millions of computer networks around the world when a tweaked version of the update of the software started spreading version of Medoc software injecting the petya ransomware. The ransomware used victims Medoc entry point to affect the victim network and later spread through primary addresses on LAN and remote IPs.

4 The Current System

Open source software provides us a platform into viewing what goes into the making of the software. However, for most, the use of this comes from directly using the compiled binaries with no assurance that this is the same binary as everyone else has downloaded. The use of a public append only log can solve the issue wherein a user can verify that the latest updates have been logged prior to downloading.

4.1 Current model of Binary Transparency

In the last section, we discussed about basics of Binary transparency. Those basics are widely used in the current system. Currently, binary transparency contains five sections as shown below in Fig. 3:

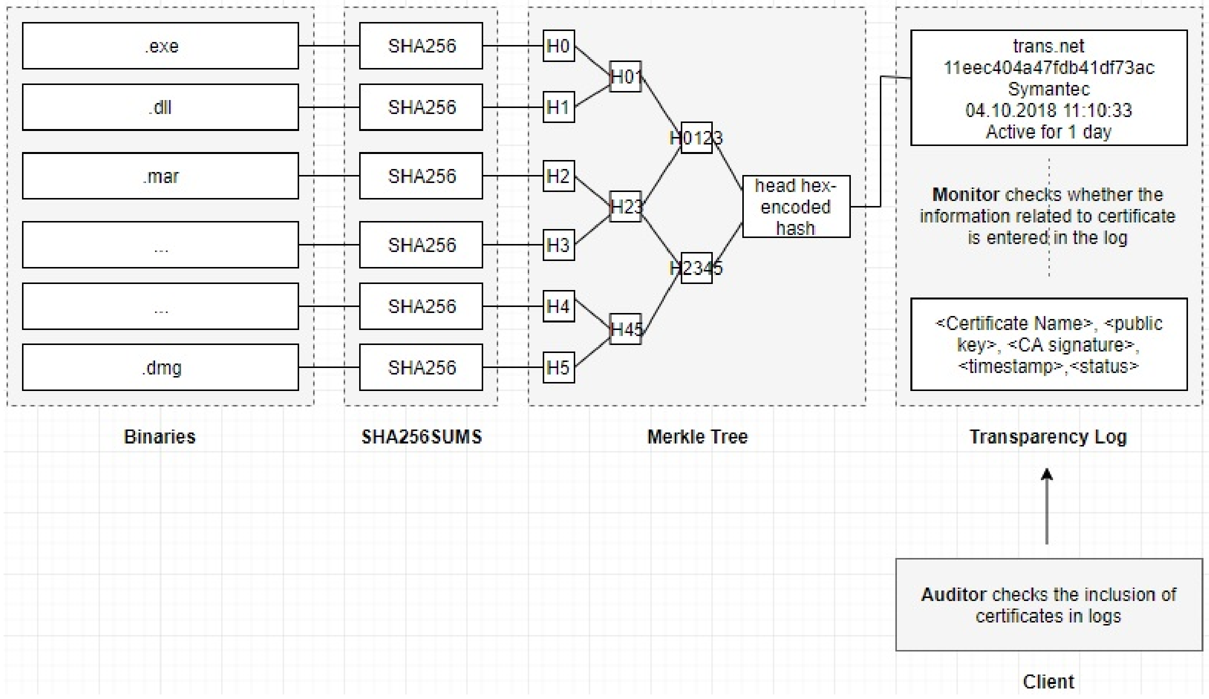
.exe
.dll
.mar
...
...
.dmg
SHA256
H0
H1
H2
H3
H4
H5
H01
H23
H45
H0123
H2345
head hex-encoded hash
trans.net
11eec404a47fdb41df73ac
Symantec
04.10.2018 11:10:33
Active for 1 day
Monitor checks whether the information related to certificate is entered in the log
<Certificate Name>, <public key>, <CA signature>, <timestamp>,<status>
Binaries
SHA256SUMS
Merkle Tree
Transparency Log
Auditor checks the inclusion of certificates in logs
Client

Figure 3: Current model of Binary Transparency

- First, we start by obtaining artifacts such as binaries, SHA256SUMS file, certificate entries, and so on.
- These binaries can be any releases such as exe, dmg, mar, and so on. SHA256SUMS file contains digest of all these binaries which will eventually be used in creation of log.

- Once all the artifacts are acquired, compute the Merkle tree root using SHA-256 hash function and SHA256SUMS digest.
- Using the 16 byte Merkle tree head hex-encoded value construct a domain name.
- Obtain the certificate comprising domain name.
- For verification, inclusion proof and the files in release are checked to confirm the certificate is correctly logged and the release consists of SHA256SUMS entries.
- At this point monitor traverses the log and inspects the right certificate information is entered or not.
- At the client side, auditor inspects whether the certificate is added in the log or not.

4.2 Issues with current Model

Several threats arise as entities interact. These include- compromised log servers, malicious monitors, malicious CAs(in our case, software publishers), and malicious web clients. Furthermore, the problems can be classified as syntactical inconsistencies pertaining to the log entries, failure to notify affected parties and presenting false information on inclusion of the entries.

4.2.1 Split-View Attack

An additional line of communication(through gossip protocol) is introduced between the auditors and monitors to periodically exchange information. This is done in order to check if their view is consistent with each other. The logging server, could thus, be susceptible to split-view attack. This could potentially lead to different views of the releases to different users.

Split-world attack deals with the situation wherein adversary manipulates the log such that specific clients believe the certificate exists in the log. Thus monitoring clients are hidden from this view and are unaware of this. Furthermore an introduction of malicious clients/servers and inject specific messages into the gossip protocol can degrade the performance of the system.

4.2.2 One-way visibility

Ideally, the blockchain should assist in continuously updating the auditors view and the auditor should not have to constantly request for the latest view of the log. In the current system, this is not achieved. Through the usage of blockchain, consistent view and automatic one way visibility reduces bandwidth cost and lowers latency on the entire system.

4.2.3 Privacy/Anonymous Auditing

User's privacy w.r.t. download behaviour is neglected in the current system due to presence of public-append log and auditing mechanism. We enforce privacy while checking contents of the software binary. Thus, when a client wants to check whether they received the right binary it can simply check through the use of auditors without revealing what exactly is it looking for and other peers in the system are unaware of its actions.

5 Verifiable Sharing Scheme

Before we propose the model, it's important to discuss the commitment scheme or inclusion proof which will improve the privacy of log process. We are using Feldman's Scheme as an improvement for privacy issue of Binary Transparency. It is a variation of Verifiable Sharing Scheme (VSS) (Verifiable Sharing Scheme, 2018a).

A sharing scheme is said to be verifiable if any supplementary information when incorporated into the system allows the shareholders to verify their shares. This scheme consists of two phases:

- Sharing: Dealer is a player who wants to share some information (secret). At the preliminary stage, dealer contains a secret input and all other shareholders have a individual random input. In this phase there are several rounds. In each round, each player sends or broadcast messages to others. Message can be based on input, random input, or previous conversations.
- Reconstruction: In this phase each player reviews the information from sharing phase and a reconstruction function is applied and final output is generated.

This technique ensures that even though dealer is hostile, the players can reconstruct the well-defined secret. This technique is suitable for certificate and binary transparency. This technique is suitable for certificate or binary transparency. If a user wants to check whether the binary he is installing is authorized, he will check using corresponding share (random input) by applying the reconstruction function.

5.1 Feldman's Scheme

This variation of VSS is based on Shamir's secret sharing scheme and homomorphic encryption (Verifiable Sharing Scheme, 2018a).

In Shamir's secret sharing scheme, a secret is split-ted into different parts, every player has its own part. In reconstruction phase, minimum number of parts are required. Now, this number should be less than the total parts, otherwise every player will reconstruct the secret (Secret sharing, 2018b).

Homomorphic encryption usually works for encrypted data. When an encrypted data is decrypted, the result should equal with the operation conducted on plain-text (Homomorphic encryption, 2018c). Together both fabricates Feldman's Scheme. In our use-case it works perfectly to solve privacy issue by distributing shares and reconstructing the secret. Following is the description of the scheme:

- Initially, choose a large prime 'p' and create a Cyclic group 'G' with generator 'g' using p. Now, 'G' should be selected such that discrete logarithm problem computation is hard.
- Choose another large prime number 'q' and create a subgroup of order q, such that q divides p-1.

- Dealer computes a secret random polynomial f(x) with degree m, such that f(0) = s (secret)

$$f(x) = s + a_1 x + ... + a_t x^m \tag{1}$$

- Each 'n' share holders receives f(1) ... f(n) modulo q.
- At this moment, secret (s) can be recover by m+1 share holders, but at most m share holders can't.
- This is Shamir's secret sharing scheme. For verification, commitments are calculated based on coefficients of polynomial f(x) and shared to share holders by dealer:

$$c_0 = g^s, c_1 = g^{a_1} \cdots c_m = g^{a_m} \tag{2}$$

- Share holders can verify their shares using v = f(i) modulo p with the following equation:

$$g^v = c_0 c_1^i c_2^{i^2} \cdots c_m^{i^m} = \prod_{j=0}^{m} c_j^{i^j} = \prod_{j=0}^{m} g^{a_j i^j} = g^{\sum_{j=0}^{m} a_j i^j} = g^{f(i)} \tag{3}$$

5.2 Example of Feldman's Scheme

Let's take an example of the above scheme.

- Consider a field Z_{11} with generator g = 3. Therefore, p = 11.
- Choose another sub-group of prime q = 5 (q divides p-1).
- The random polynomial should be modulo q. Therefore, coefficients can be between 0,..,4 based on q value.
- Let us consider a polynomial

$$f(x) = 0 + 3x + 3x^2 \tag{4}$$

- The first share s_1 will be represented as f(1) for share holder h_1. Thus,

$$\begin{aligned} f(1) = 0 + 3 * 1 + 3 * 1^2 \, mod \, q \\ f(1) = 6 \, mod \, 5 \\ f(1) = 1 = s_1 \end{aligned} \tag{5}$$

- Commitments are given as $g^{\text{coefficients}}$ modulo p,

$$c_0 = 3^0 \, mod \, 11 = 1, c_1 = 3^3 \, mod \, 11 = 5, c_2 = 3^3 \, mod \, 11 = 5 \tag{6}$$

- For verification, using equation (5), compute

$$g^{s_1} \, mod \, p = 3^1 \, mod \, 11 = 3 \tag{7}$$

- Now, using equation (6), compute check value when i = 1 for share holder h_1,

$$\begin{aligned} c_0 c_1^i c_2^{i^2} \, mod \, p = \\ 1 * 5^1 * 5^{1^2} \, mod \, 11 = \\ 25 \, mod \, 11 = 3 \end{aligned} \tag{8}$$

- From equation (7) and (8), it shows that g^{s_1} is equal to check value. Hence, share s_1 is correct.

6 The Proposed System

We have covered all the basics for our proposed model. Now, we will consider how our proposal will combine all those basics and make a complete solution to solve the issues of binary transparency.

6.1 Design Goals

- Privacy problem for binary transparency using Feldman's Scheme.
- Issues in Gossip protocol in auditor and monitor in certificate transparency resolvable using blockchain.
- Availability attacks are also solved by using blockchain wherein auditor waits for the monitor to inspect the published certificate, but it can be possible that certificate is not publish yet.
- Operations such as communication costs, computations, storage should happen efficiently.
- Efficient setting up of the system which requires minimal effort and computation overhead. This is achieved by checking the block headers, rather than individual transactions.

6.2 Usage with Blockchain

Blockchain plays an important role in the proposal. It provides security and authenticity, however, this is achieved at the comprise of privacy. Sometimes organisation want their binaries to remain private. Also, it shouldn't be the case where some of the users downloading the software are specifically targeted. The technique we are proposing perfectly solves the problem of privacy faced by binary transparency. Feldman's Scheme makes the content private for said subsidiaries. It distributes the shares to individual parties to verify their content on blockchain.

The simplified model for binary transparency is shown in Fig. 4,

- Blockchain involves transactions. Content of binaries are taken as individual transactions.
- These transactions are used to generate intermediate hashes for each binary content and combined to form a Merkle tree root. This Merkle tree root corresponds to the release version of the software. Note: Only the Merkle root of these transactions is published to the blockchain.
- Blocks in the chain are linked with the previous block. This ledger will be public and shows the time, date, name, and organization issuing the certificate.

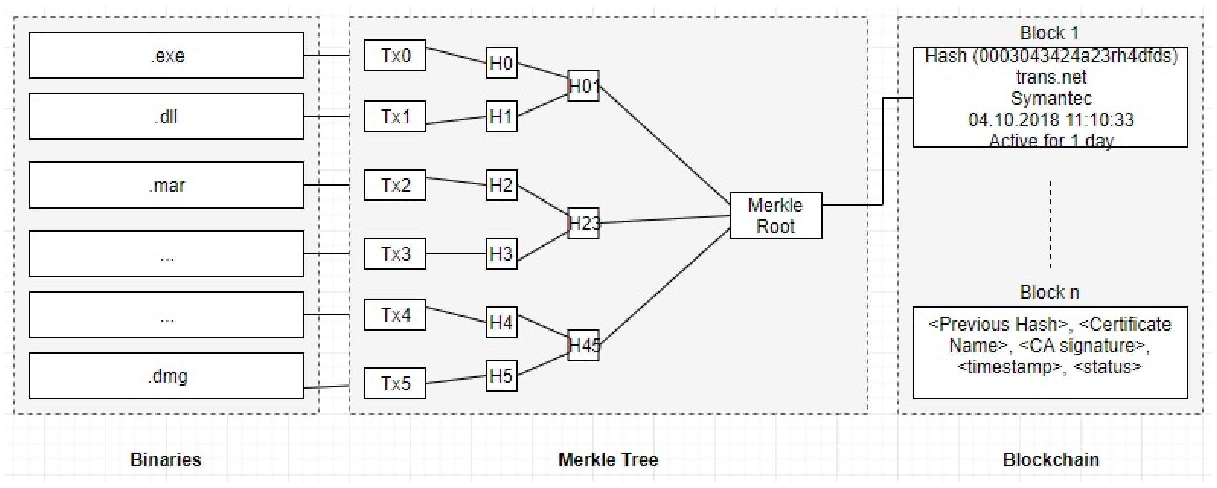

Figure 4: Proposed model

7 Critical Evaluation

7.1 Software release life cycle

The fundamental concept behind the architecture lies in the fact that unique share/commitment value is given to each client as they request access to the software. This unique share (due to its public visibility) can be verified by any client independently without relying on other peers. Furthermore, as more clients join the system, the polynomial $f(x) = s + a_1x + ... + a_tx^m$ i.e. held privately by the software publisher, contains values which are unique to a particular release.

As the software owner publishes new versions of the software, the coefficients of the secret polynomial change. We propose, the following coefficients for the the same:

- Hash value of the certificate (generated through a collision-resistant hash function)
- Hash value of the certificate + Time-stamp value
- Hash value of the certificate + Index of the certificate in the blockchain

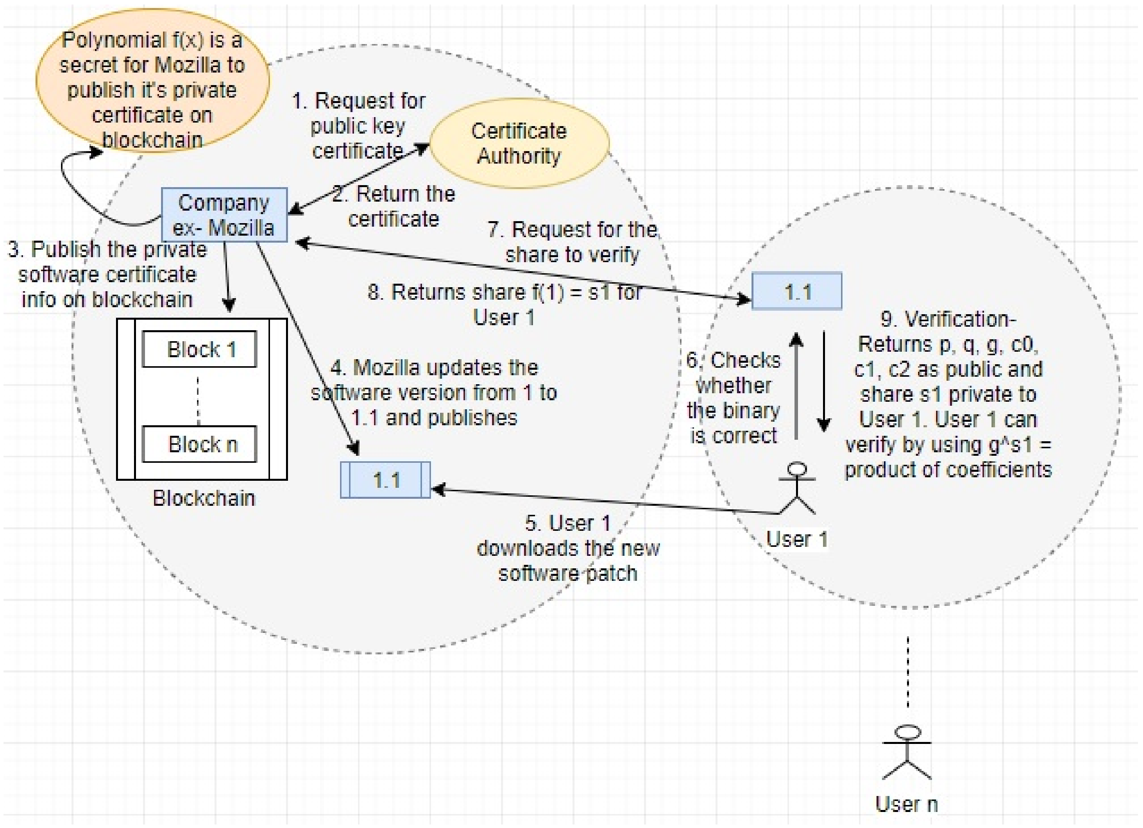
Polynomial f(x) is a secret for Mozilla to publish it's private certificate on blockchain
1. Request for public key certificate
Certificate Authority
Company ex- Mozilla
2. Return the certificate
3. Publish the private software certificate info on blockchain
7. Request for the share to verify
8. Returns share f(1) = s1 for User 1
1.1
Block 1
Block n
Blockchain
4. Mozilla updates the software version from 1 to 1.1 and publishes
1.1
6. Checks whether the binary is correct
9. Verification- Returns p, q, g, c0, c1, c2 as public and share s1 private to User 1. User 1 can verify by using g^s1 = product of coefficients
5. User 1 downloads the new software patch
User 1
User n

Figure 5: Software Release Life cycle. We have consider an example of company Mozilla while issuing binary updates.

The CA helps the software publisher attain a public key certificate. The software publisher can verify that its published release to the blockchain is indeed signed by CA. This avoids the issuance of fraudulent certificates and also helps in detecting cases where the software company was unaware that a certificate was published. Clients can decrypt this as they have the CA's public key stored on their machines block store.

7.2 Comparative Analysis

	Certificate Transparency	CONIKS	Bitcoin	Subversion Transparency
Split Views	Only detected	Only detected	Avoided	Avoided
Availability	Not available	Not available	Available	Available
Auditor Privacy	Not available	Not available	Available	Available
Efficiency (size)	1	1	No. of transactions	No. of blocks
Minimal Setup	Not available	Not available	Available	Available

Table 1: Comparison of different schemes in-terms of the design goals described as adapted from Table 4 in Al-Bassam and Meiklejohn [2018].

CT and COINKS require only single bit hash to be stored. However, occurrence of split view attack is possible in such system due to gossip protocol. In terms of efficiency, CT servers are mainted by Goggle and in the case of CONIKS, user decide their identity providers. Subversion transparency make it easier to set up because it does not have such requirements.

7.3 Ethereum Cost

Currently based on more elaborate GHOST protocol compared to Bitcoin. The speed of Ethereum is faster due to convenient number theoretic happenstance. During happenstance the faster running chain have low certainty at the beginning but have much higher certainty after an amount of time.

Metric	Value
Difficulty	3,163,168,251,853,340
Hashrate	254.61 Thash/s
Blockchain Size	667.10 GB

Table 2: Reviewing cost of Ethereum blockchain.

8 Conclusion

Over the course of this report we have discussed a lot about binary transparency. The existing process is a creative process which solve some issues that surround binary transparency, but it creates a problem of it's own such as Split-View attack, privacy, efficiency, and so on. We have a proposed a system that solves all these issues via blockchain and Feldman's Scheme. Our proposed system is efficient in comparison with other techniques and provides the privacy which is essential for the organization.

References

Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate transparency with privacy. *Proceedings on Privacy Enhancing Technologies*, 2017(4):329–344, 2017.

Mustafa Al-Bassam and Sarah Meiklejohn. Contour: A practical system for binary transparency. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 94–110. Springer, 2018.

Mozilla Wiki. Security/binary transparency. 2017. URL `https://wiki.mozilla.org/Security/Binary_Transparency`.

Oded Leiba, Yechiav Yitzchak, Ron Bitton, Asaf Nadler, and Asaf Shabtai. Incentivized delivery network of iot software updates based on trustless proof-of-distribution. *arXiv preprint arXiv:1805.04282*, 2018.

Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *European Symposium on Research in Computer Security*, pages 140–158. Springer, 2016.

Shashee Kumari. Smartdnspki. 2017.

Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 38–49. ACM, 2012.

Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. *arXiv preprint arXiv:1511.01514*, 2015.

Bastian Fredriksson. A distributed public key infrastructure for the web backed by a blockchain, 2017.

Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, 2013.

Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *Draft NISTIR*, 8202, 2018.

Wikipedia contributors. Verifiable secret sharing — Wikipedia, the free encyclopedia, 2018a. URL `https://en.wikipedia.org/w/index.php?title=Verifiable_secret_sharing&oldid=837606767`. [Online; accessed 17-October-2018].

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

Seyedehmahsa Moosavi and Jeremy Clark. Ghazal: toward truly authoritative web certificates using ethereum.

Mustafa Al-Bassam. Scpki: A smart contract-based pki and identity system. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 35–40. ACM, 2017.

Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. In *USENIX Security Symposium*, volume 2015, pages 383–398, 2015.

LM Axon and Michael Goldsmith. Pb-pki: a privacy-aware blockchain-based pki. 2016.

Alexander Yakubov, Wazen Shbair, Anders Wallbom, David Sanda, et al. A blockchain-based pki management framework. In *The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018, Tapei, Tawain 23-27 April 2018*, 2018.

Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. Chainiac: Proactive software-update transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, 2017.

M Satran. X.509 public key certificate, 2018. URL `https://docs.microsoft.com/en-us/windows/desktop/seccertenroll/about-x-509-public-key-certificates`. [Online; accessed 18-October-2018].

Greenberg A. Barrett B. Newman L. Dreyfuss E. Dreyfuss E. et al Greenberg, A. The petya plague exposes the threat of evil software updates, 2018. URL `https://www.wired.com/story/petya-plague-automatic-software-updates/`. [Online; accessed 18-October-2018].

Petya ransomware outbreak: Heres what you need to know., 2018. URL `https://www.symantec.com/blogs/threat-intelligence/petya-ransomware-wiper`. [Online; accessed 18-October-2018].

Apple's battle with the fbi leaves lingering questions., 2018. URL `https://www.cnet.com/news/apple-vs-fbi-one-year-later-still-stuck-in-limbo/`. [Online; accessed 18-October-2018].

G Keizer. Researchers reveal how flame fakes windows update., 2018. URL `https://www.computerworld.com/article/2503916/malware-vulnerabilities/researchers-reveal-how-flame-fakes-windows-update.html`. [Online; accessed 18-October-2018].

P Bajpai. Block (bitcoin block), 2018. URL `https://www.investopedia.com/terms/b/block-bitcoin-block.asp`. [Online; accessed 18-October-2018].

David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393. ACM, 2014.

Wikipedia contributors. Secret sharing — Wikipedia, the free encyclopedia, 2018b. URL `https://en.wikipedia.org/w/index.php?title=Secret_sharing&oldid=852592521`. [Online; accessed 18-October-2018].

Wikipedia contributors. Homomorphic encryption — Wikipedia, the free encyclopedia, 2018c. URL `https://en.wikipedia.org/w/index.php?title=Homomorphic_encryption&oldid=861560280`. [Online; accessed 18-October-2018].