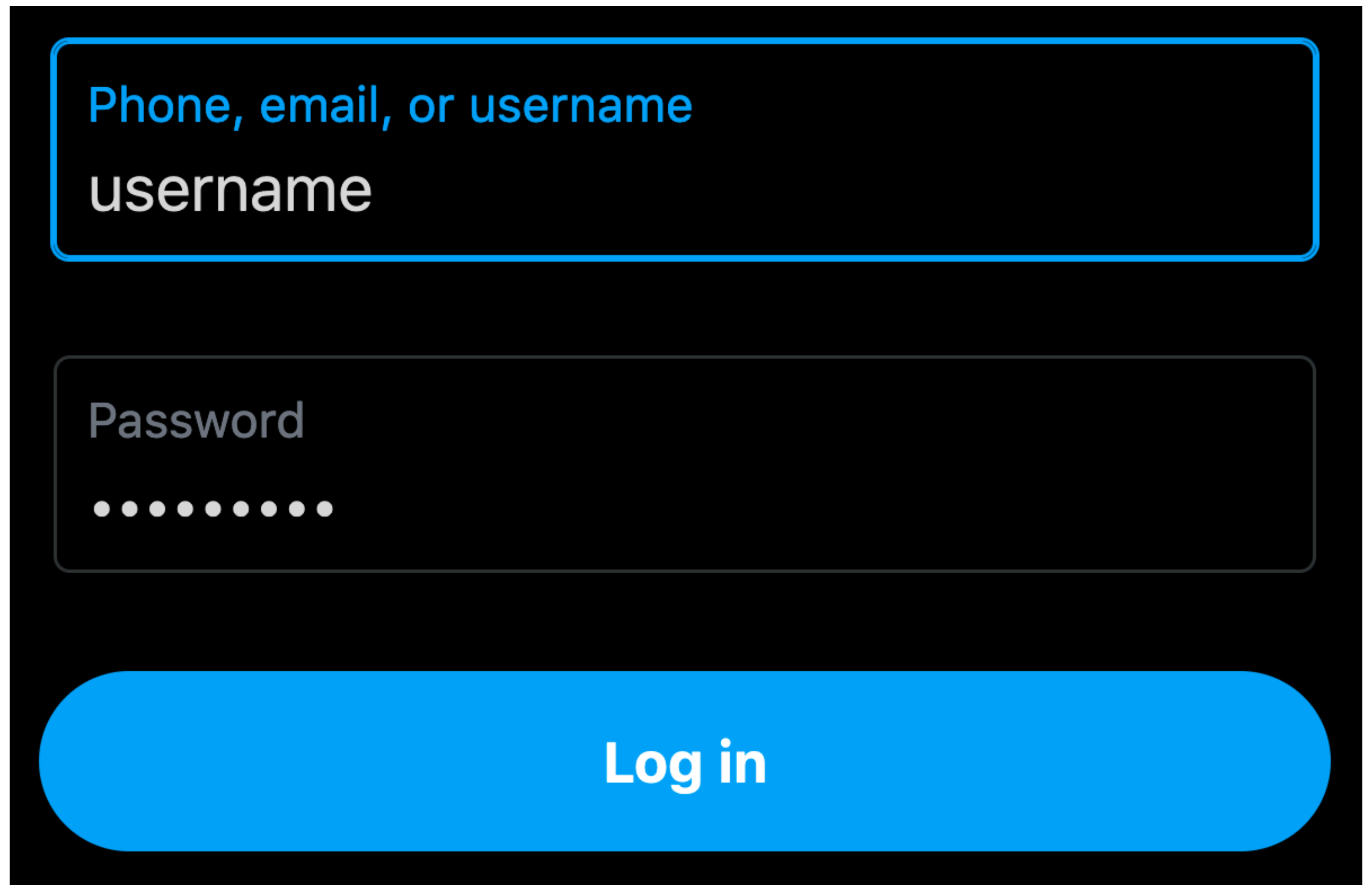
Understanding the Trust Relationships of the Web PKI

Zane Ma

Postdoc Researcher, Astrolavos Lab
zanema@gatech.edu
https://zanema.com

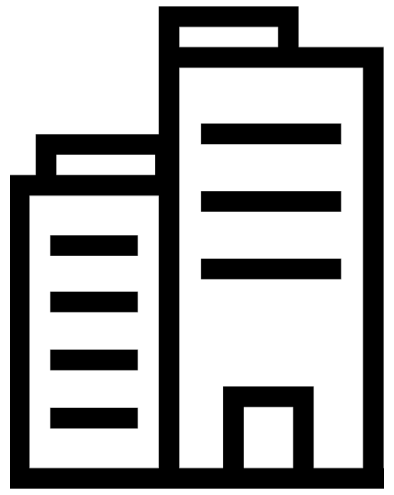
Web PKI



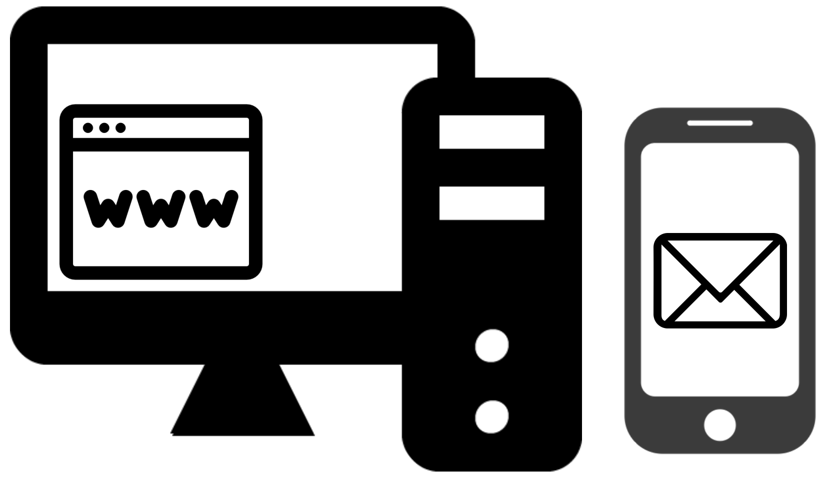
Web Public Key Infrastructure (PKI) enables TLS server authentication, the keystone for HTTPS / email transport security.

Most widely deployed PKI - scalable, delegated solution for linking an identity (e.g., DNS name or IP address) to a cryptographic public key

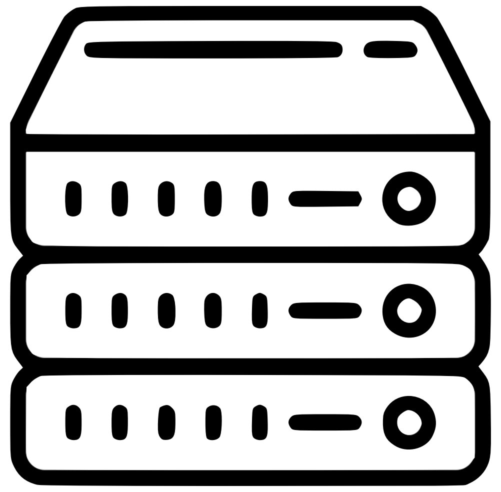
Web PKI



Certificate Authority

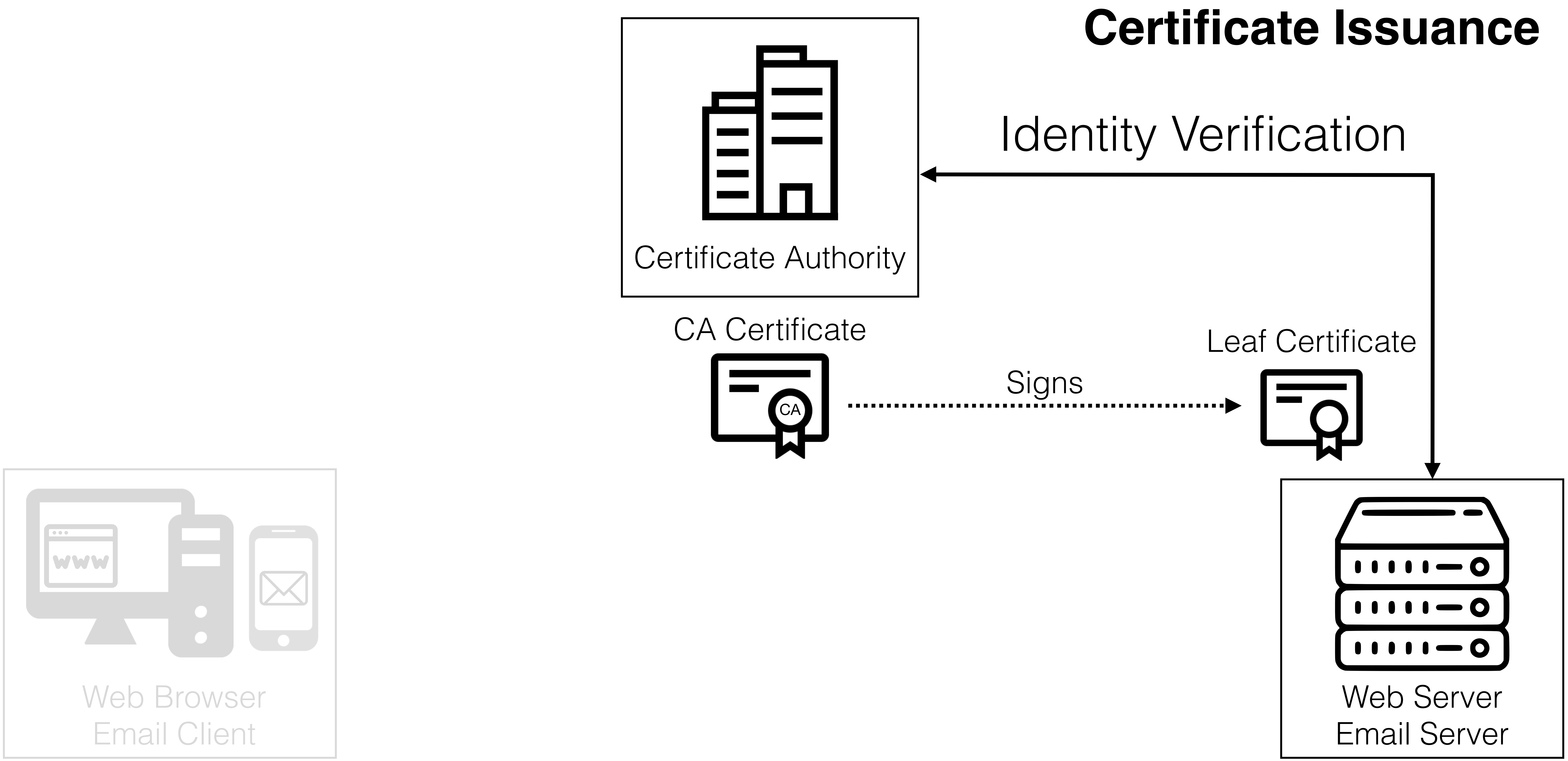


Web Browser
Email Client

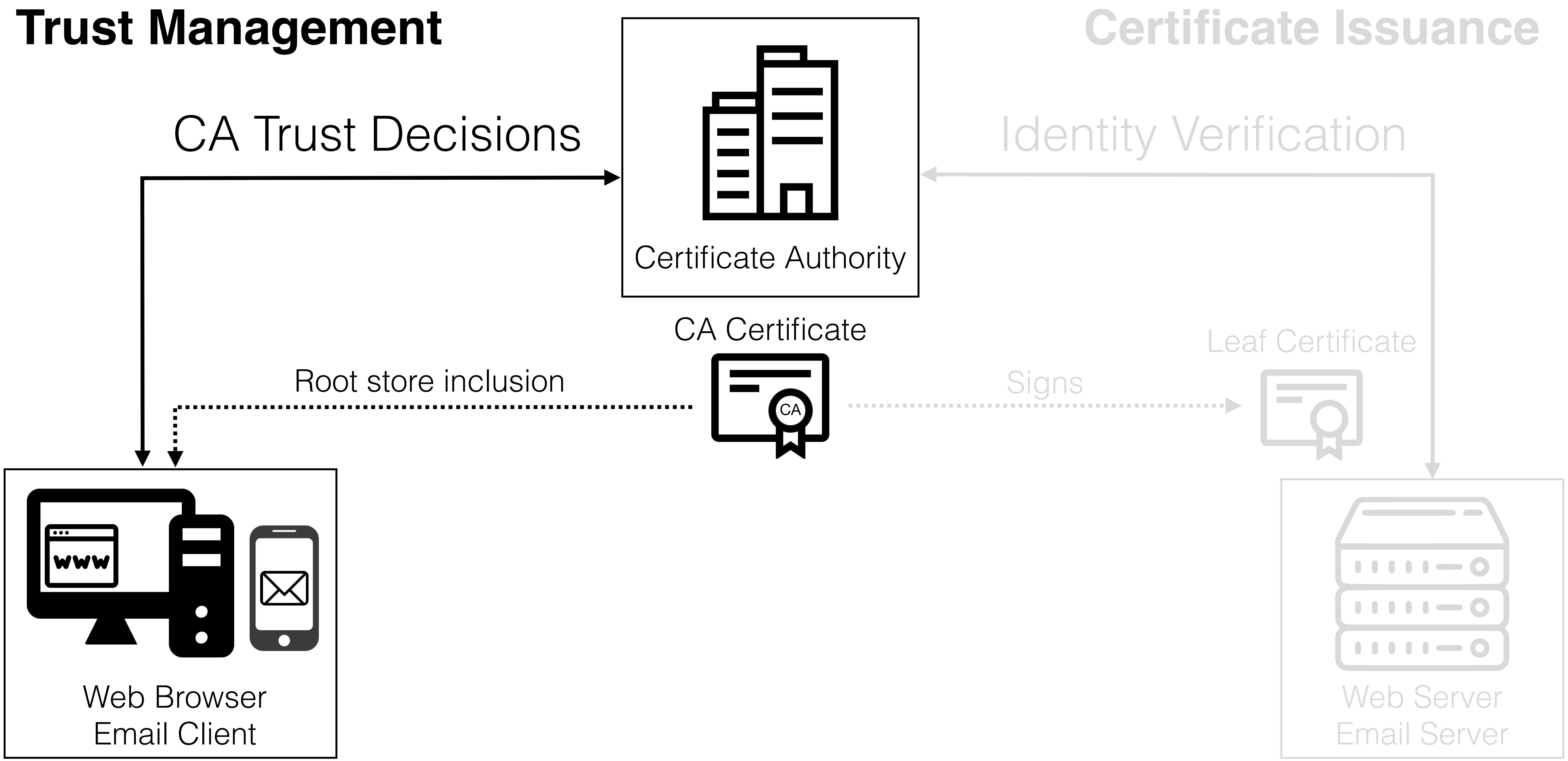


Web Server
Email Server

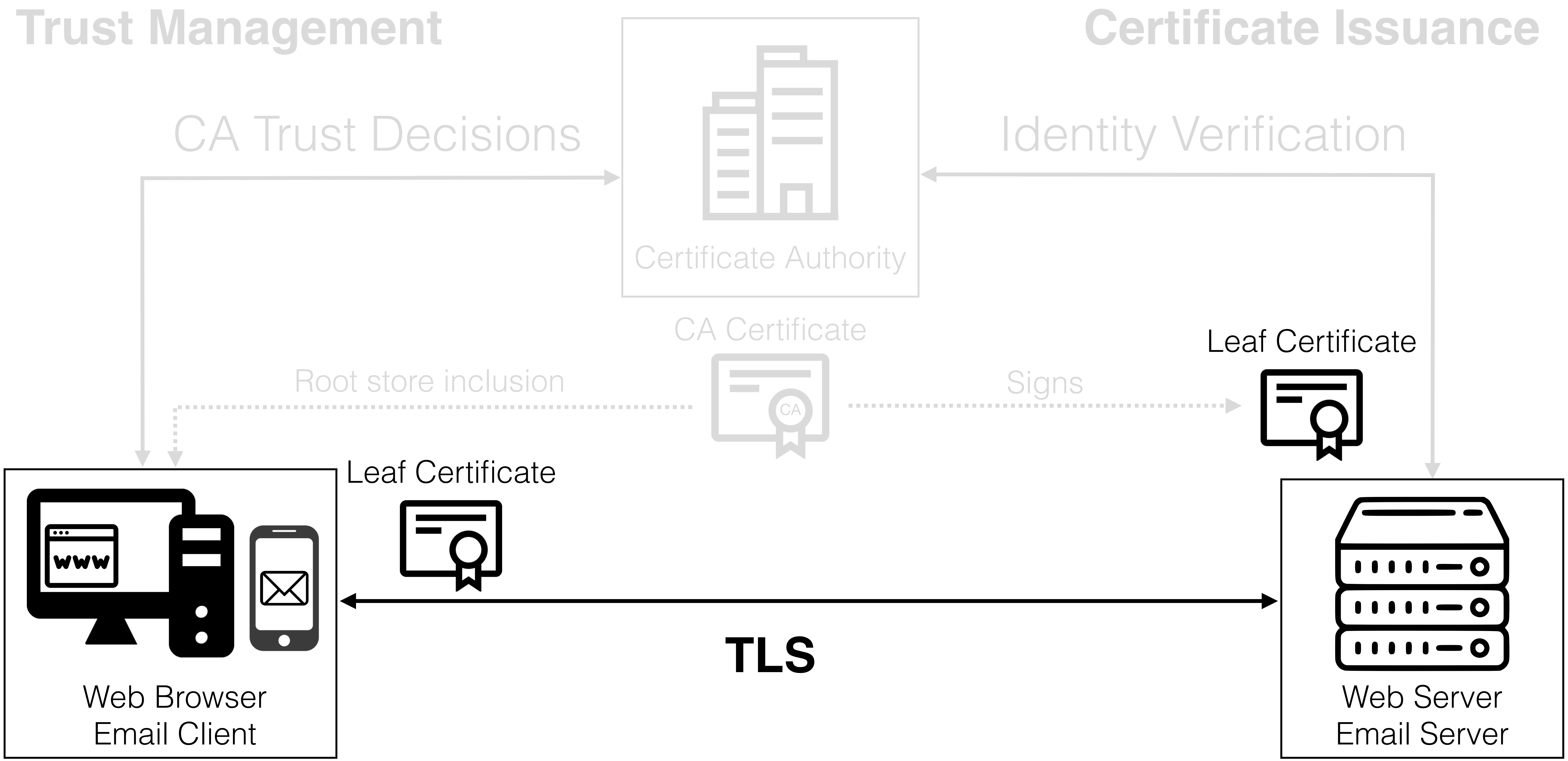
Web PKI


Certificate Issuance
Identity Verification
Certificate Authority
CA Certificate
CA
Signs
Leaf Certificate
Web Server
Email Server
Web Browser
Email Client

Web PKI


Trust Management
Certificate Issuance
CA Trust Decisions
Identity Verification
Certificate Authority
CA Certificate
Leaf Certificate
Root store inclusion
CA
Signs
www
Web Browser
Email Client
Web Server
Email Server

Web PKI


Trust Management
Certificate Issuance
CA Trust Decisions
Identity Verification
Certificate Authority
CA Certificate
CA
Root store inclusion
Signs
Leaf Certificate
Leaf Certificate
www
TLS
Web Browser
Email Client
Web Server
Email Server

Web PKI

Trust Management

Trust anchor inclusion / removal and distribution to TLS clients

Certificate Issuance

Identity verification protocols and the creation/ distribution of cryptographically signed certificates

TLS

Network protocol that establishes an encrypted channel between a client and server, typically relying on public keys to establish a unique session key

Existing Work

Trust Management

Ecosystem:
Vallina-Rodriguez (2014): Android roots
Perl (2014): Debloating root stores
Hiller (2020): Alt. chains (cross-sign)

Alternative designs:
Dacosta (2012), Kasten (2013), Matsumoto (2020), etc.++

Certificate Issuance

Ecosystem of issuers/subscribers: Holz (2011), Durumeric (2013), Vandersloot (2016), Chung (2016), Scheitle (2018), Kumar (2018)

Attacks/defenses: Marlinspike (2011), RFC 6962 (2013): Certificate Transparency, Bates (2014), Birge-Lee (2018, 2021), Scheitle (2018), Brandt (2018)

Automating issuance: Aas (2019), RFC 8737 (2020): Automated Certificate Management Environment (ACME)

TLS

Protocol bugs / verified implementations:

Cremers (2016), Li (2016), Bhargavan (2016), CRIME (2012), AlFardan (2013), POODLE (2014), Bhargavan (2008, 2013, 2017), Cremers (2017), Delignat-Lavaud (2017)

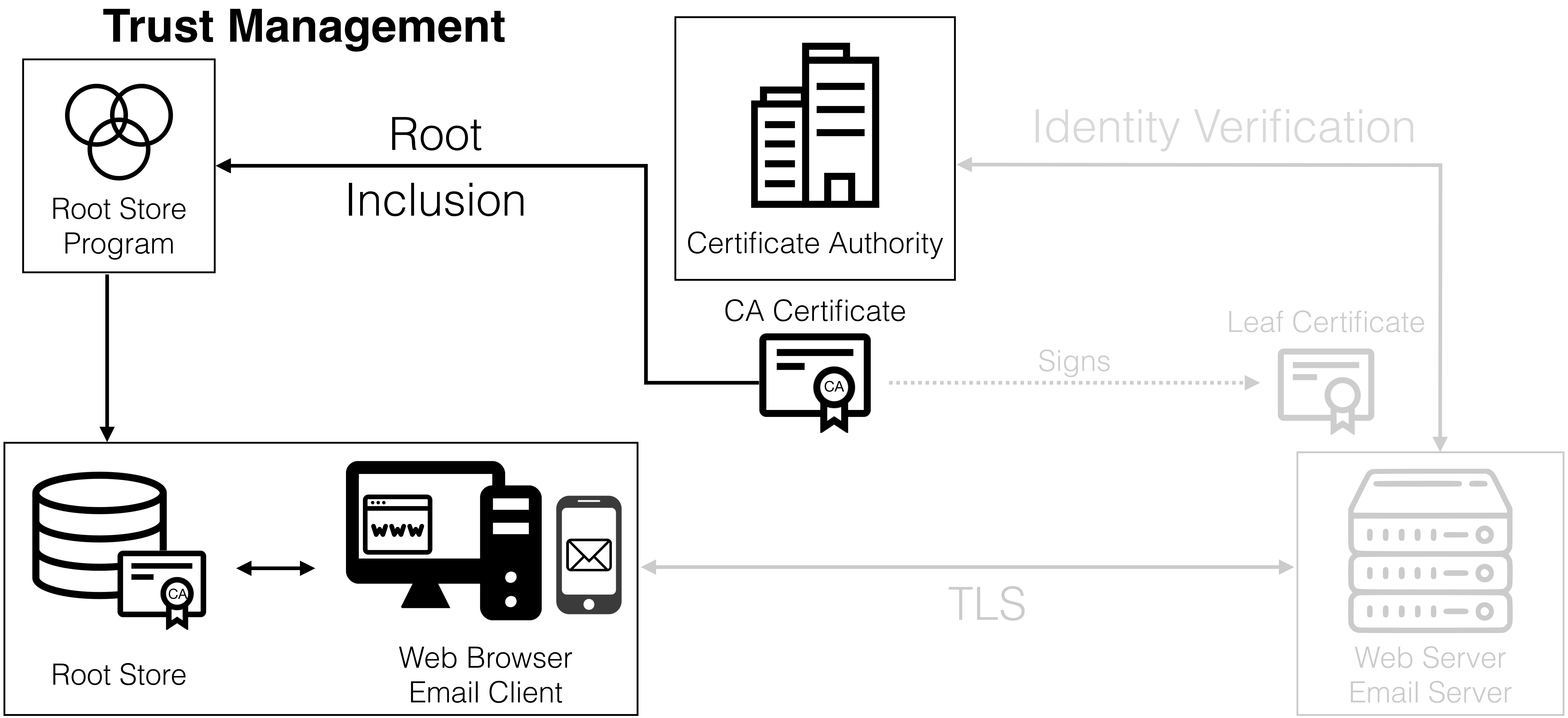
Implementation bugs:

Georgiev (2012), Durumeric (2014), Bates (2014), Brubaker (2014), Sounthiraraj (2014), Bhargavan (2014), Beurdouche (2015), Ruiter (2015), Chau (2017), Sivakorn (2017), Oltrogge (2015, 2021)

Research Questions

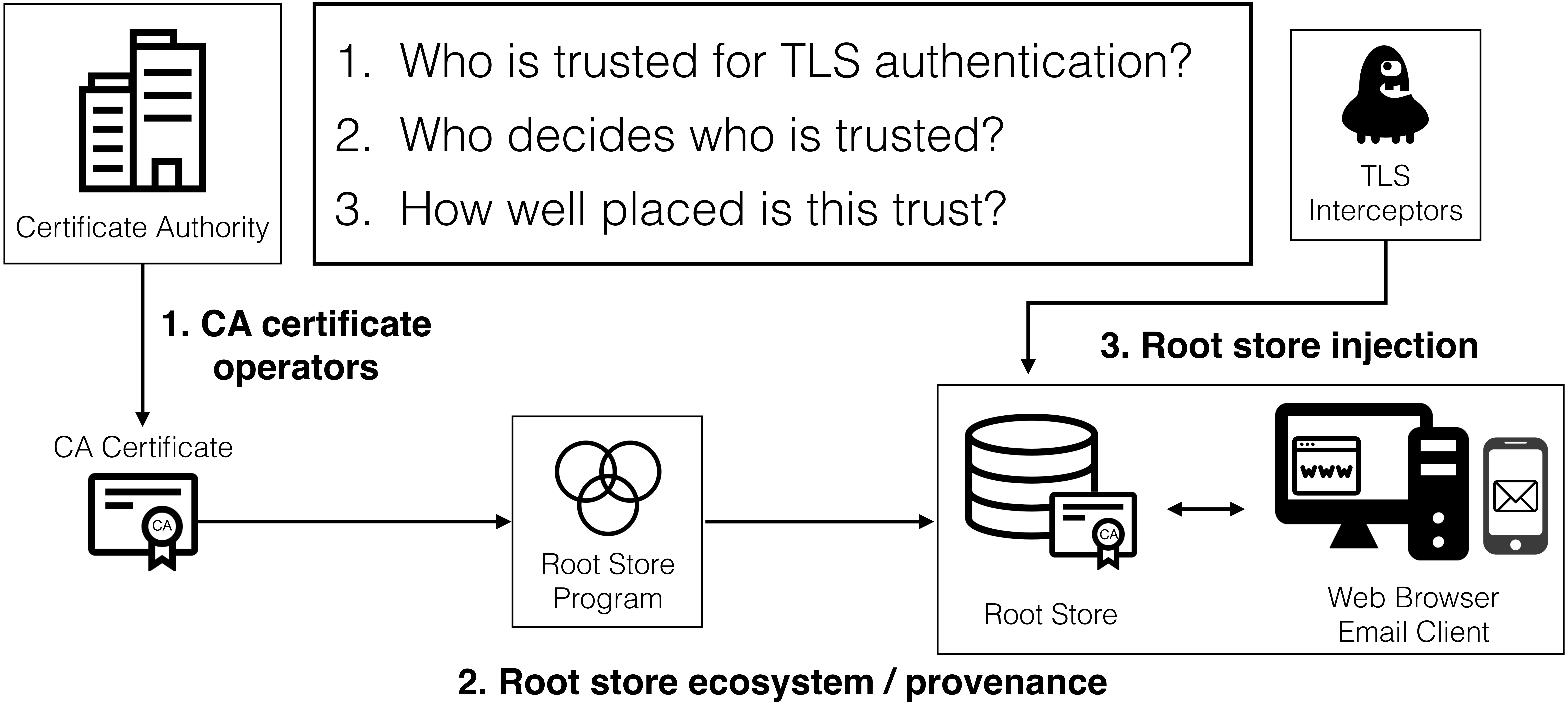
1. Who is trusted for TLS authentication?
2. Who decides who is trusted?
3. How well placed is this trust?

Trust Management

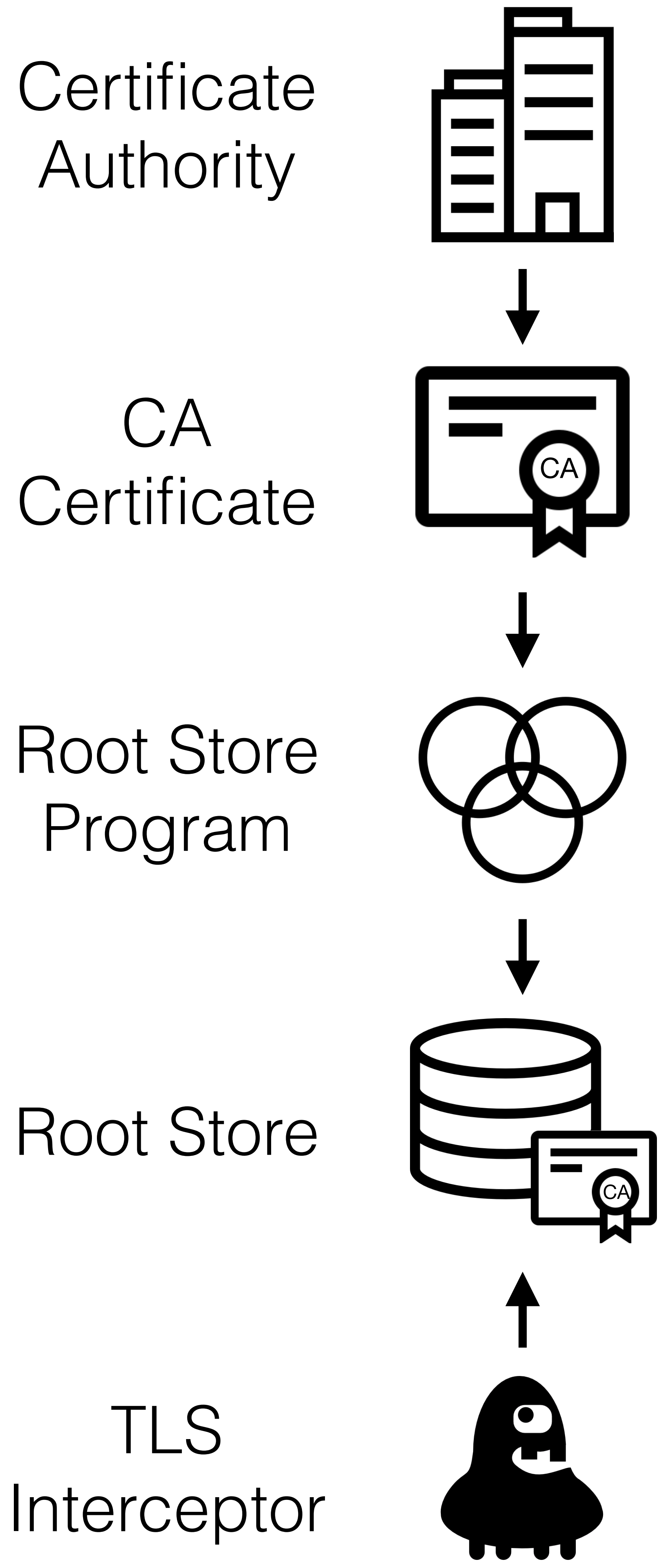


Research Overview

1. Who is trusted for TLS authentication?
2. Who decides who is trusted?
3. How well placed is this trust?

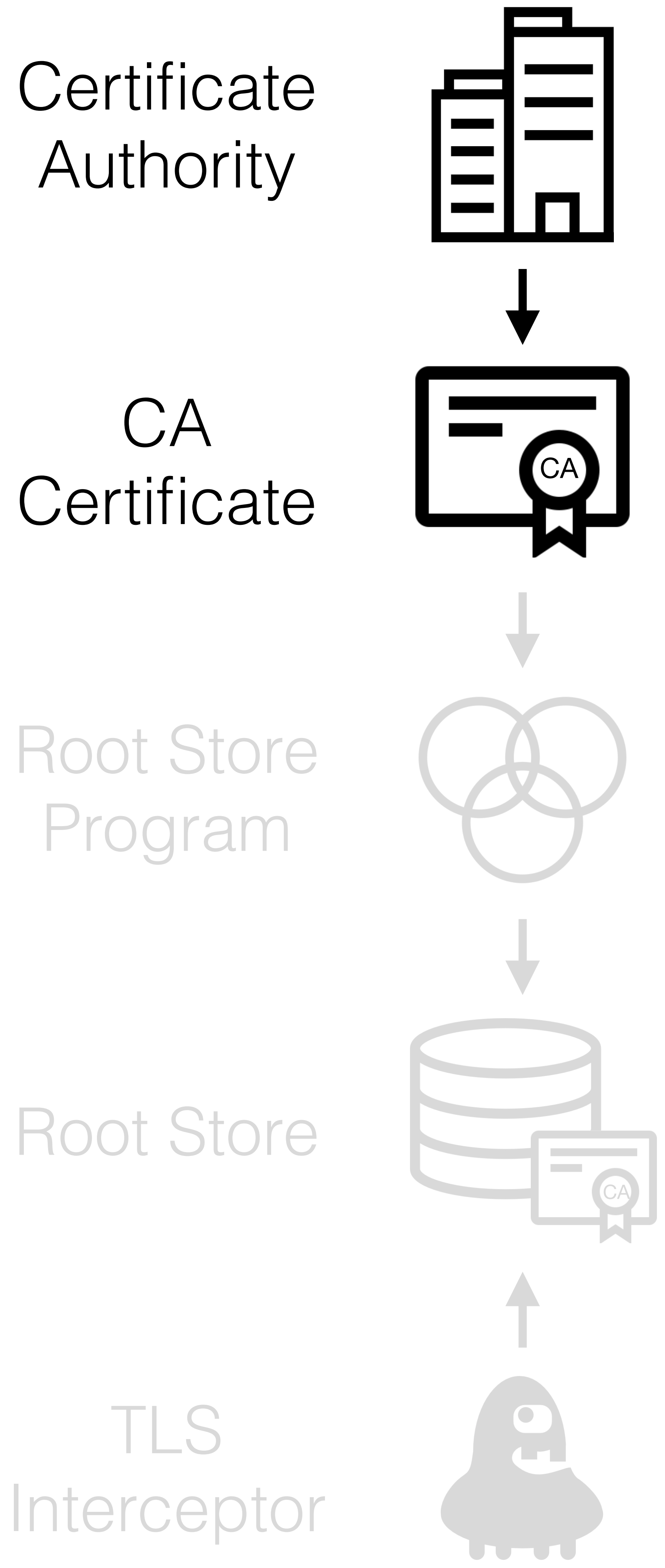


Today's Talk



1. Background + Motivation
2. Identifying CA certificate operators *USENIX 2021*
3. Mapping root store provenance *IMC 2021*
4. Detecting TLS interception *NDSS 2017*
5. Conclusions + Future Work

CA operator transparency



1. Background + Thesis

2. Identifying CA certificate operators *USENIX 2021*

3. Mapping root store provenance *IMC 2021*

4. Detecting TLS interception *NDSS 2017*

5. Conclusions + Future Work

Symantec Distrust

- From 2009-2017 Symantec was responsible for over a dozen issues[1] that prompted removal from browser root stores
- Difficult to determine which root CA certificates Symantec operated!

```
commonName          = UTN-USERFirst-Client Authentication and Email
orgUnitName         = http://www.usertrust.com
orgName             = The USERTRUST Network
localityName        = Salt Lake City
stateOrProvinceName = UT
countryName         = US
```

Comodo Root #1

```
commonName          = UTN-USERFirst-NetworkApplications
orgUnitName         = http://www.usertrust.com
orgName             = The USERTRUST Network
localityName        = Salt Lake City
stateOrProvinceName = UT
countryName         = US
```

Symantec Root #2

[1] https://wiki.mozilla.org/CA:Symantec_Issues

Takeaways

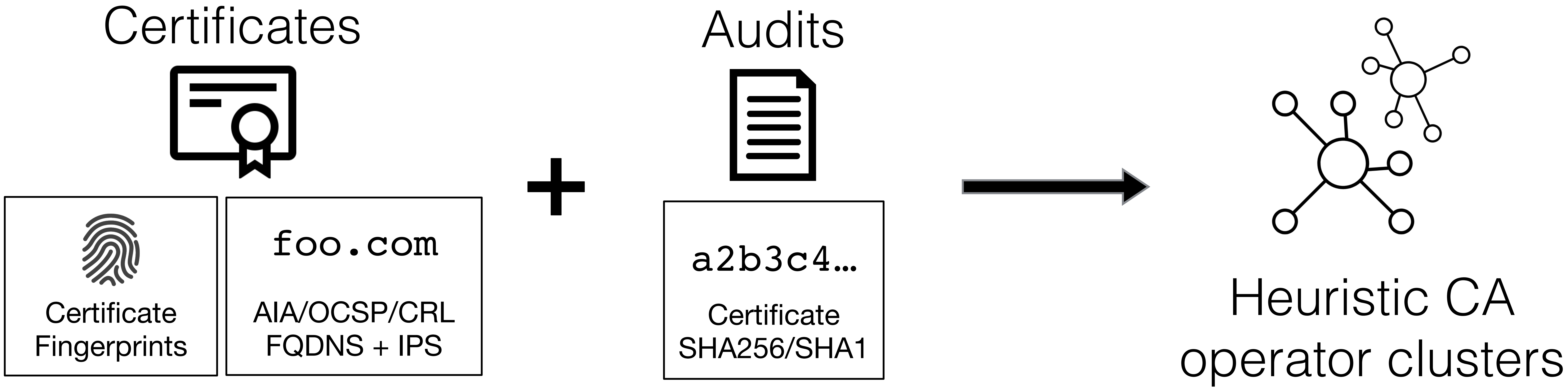
1. TLS authentication trust occurs at the level of CAs (a.k.a. CA certificate operators), not CA certificates.

2. There are no guarantees that the identity in a CA certificate reflects the operator of the CA certificate.

Approach

How can we determine the *operator* of a CA certificate / issuer?

1. Measure CA operational features to detect CA certificates with shared CA operators

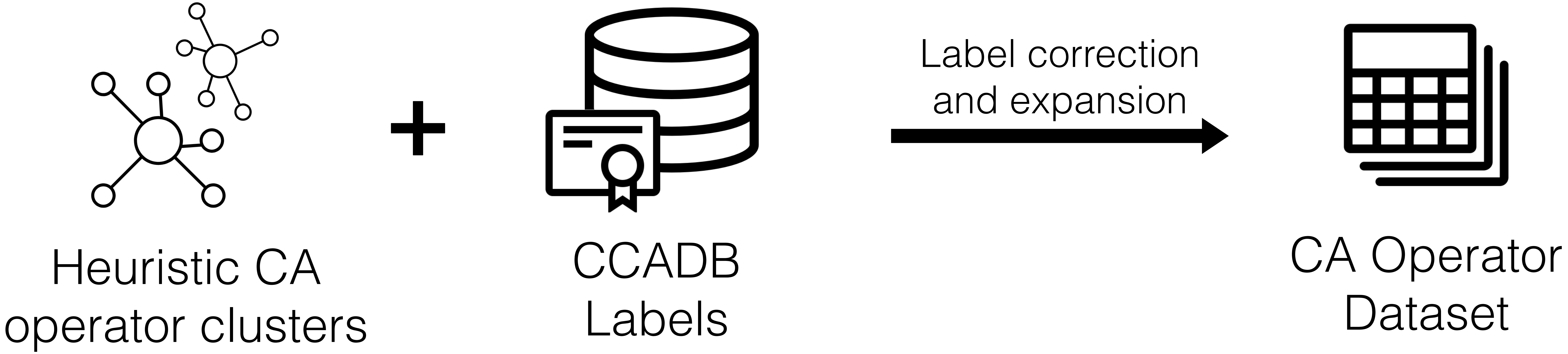


Approach

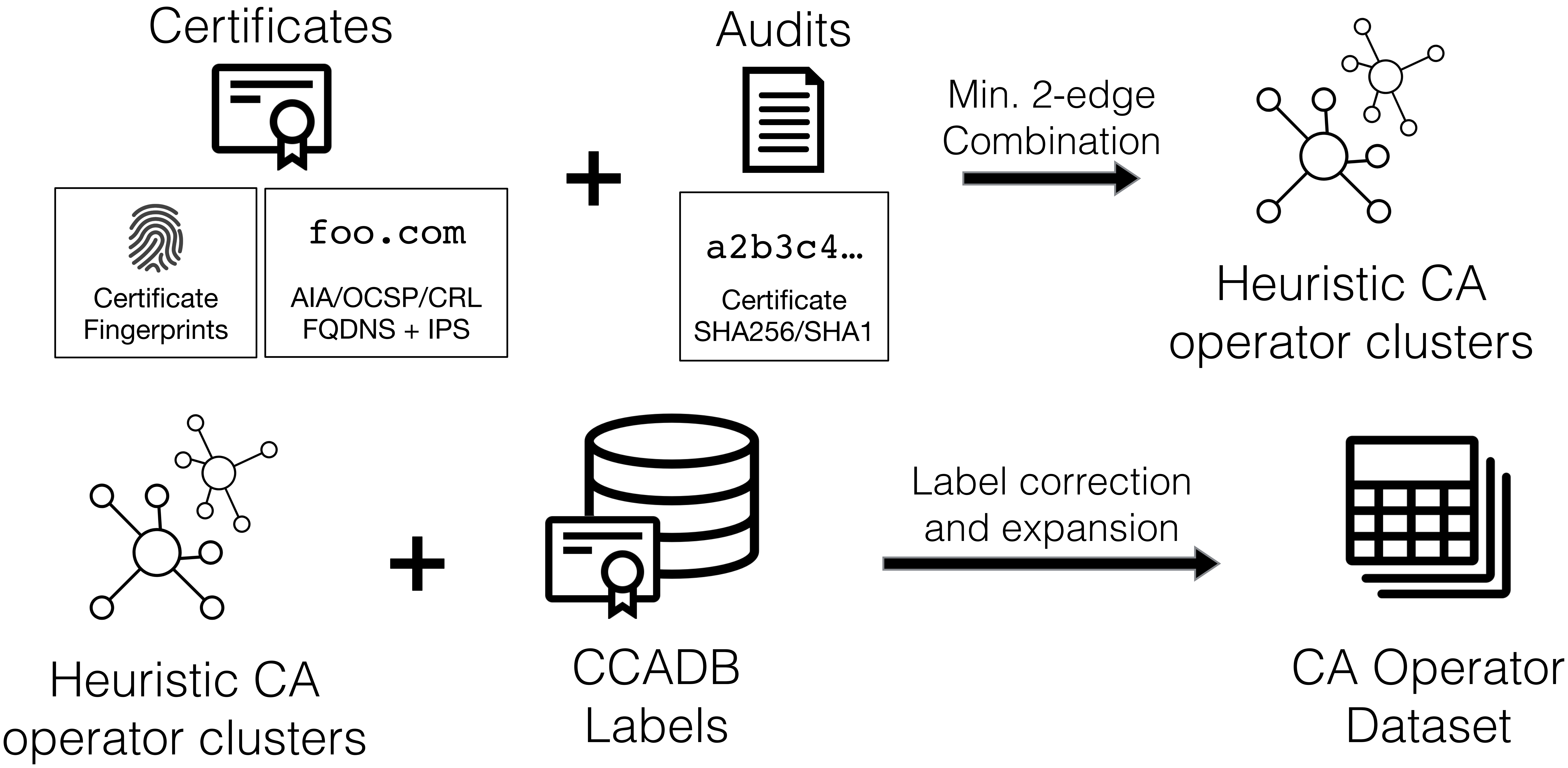
How can we determine the *operator* of a CA certificate / issuer?

1. Measure CA operational features to detect CA certificates with shared CA operators

2. Carefully apply CCADB to label CA operator clusters



Pipeline


Certificates
Certificate Fingerprints
foo.com
AIA/OCSP/CRL FQDNS + IPS
+
Audits
a2b3c4…
Certificate SHA256/SHA1
Min. 2-edge Combination
Heuristic CA operator clusters
Heuristic CA operator clusters
+
CCADB Labels
Label correction and expansion
CA Operator Dataset

Results

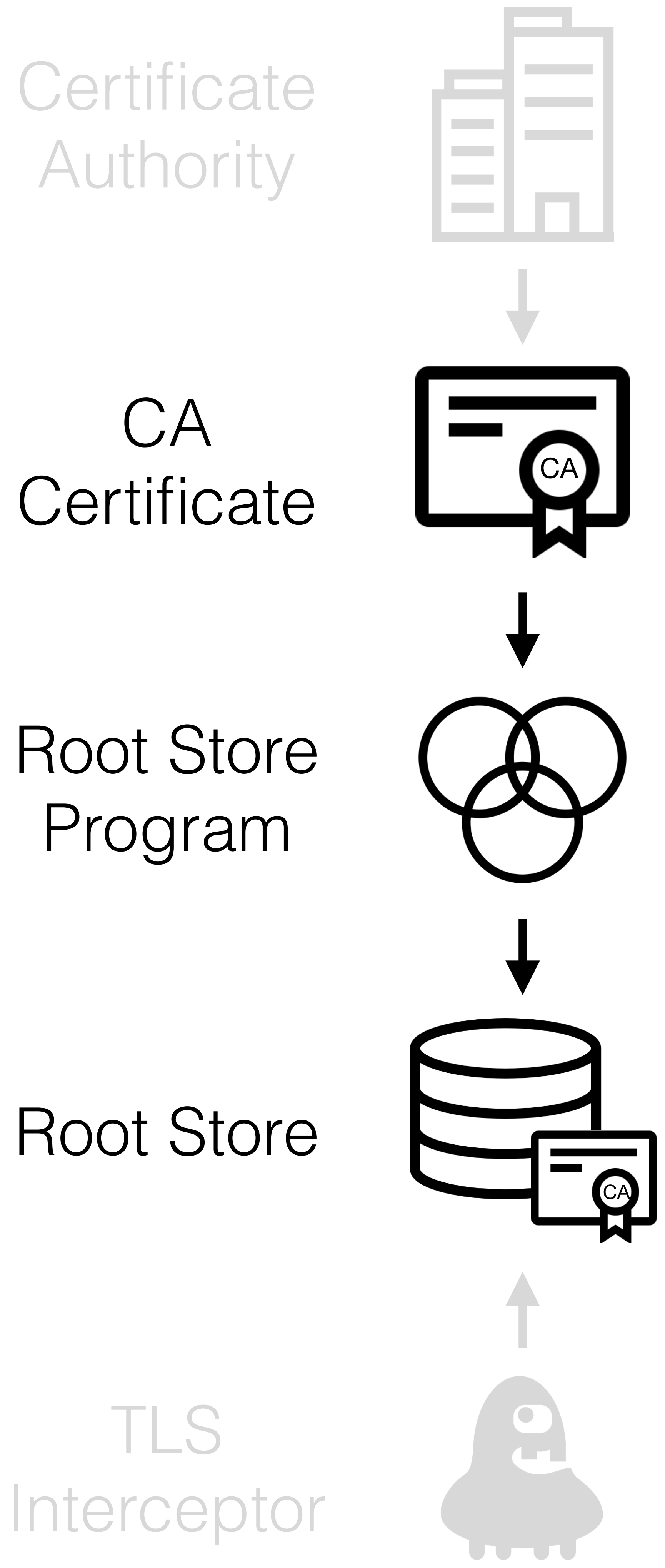
Discovery	Outcome
Improperly disclosed Camerfirma subordinate CA (MULTICERT)[1]	Camerfirma removed from Mozilla root store, distrusted by Google products
Refined CA operator labels for 241 CA certs Added new labels for 651 unlabeled CA certs	CCADB exploring automated sub-CA consistency checking [2] and ownership annotation [3]

<u>CA operational transparency means:</u>

1) More informed root store decision making
2) More accurate research / issue attribution

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=1672029
[2] https://bugzilla.mozilla.org/show_bug.cgi?id=1727204
[3] https://bugzilla.mozilla.org/show_bug.cgi?id=1727205

Characterizing TLS trust



1. Background + Motivation
2. Identifying CA certificate operators *USENIX 2021*
3. Mapping root store provenance *IMC 2021*
4. Detecting TLS interception *NDSS 2017*
5. Conclusions + Future Work

TLS user agents

TLS user agents

Research Questions

1. Which root store providers do TLS user agents rely on?

2. How do root store providers determine which CAs to trust?

3. Characterization of root store programs

4. How faithfully do providers copy root program trust?

TLS user agents

Challenge: Lots of TLS user agents

Approach:

1) Study popular user agents, look at the top 200 for global CDN

2) Best-effort collection of popular libraries / clients / OSes

Limitations: misses the long tail of TLS authentication trust

Data collection

User Agents

Web Browsers

Chrome	Chrome Mobile	Opera	Firefox	
Safari	Mobile Safari	Edge	IE	Chromium

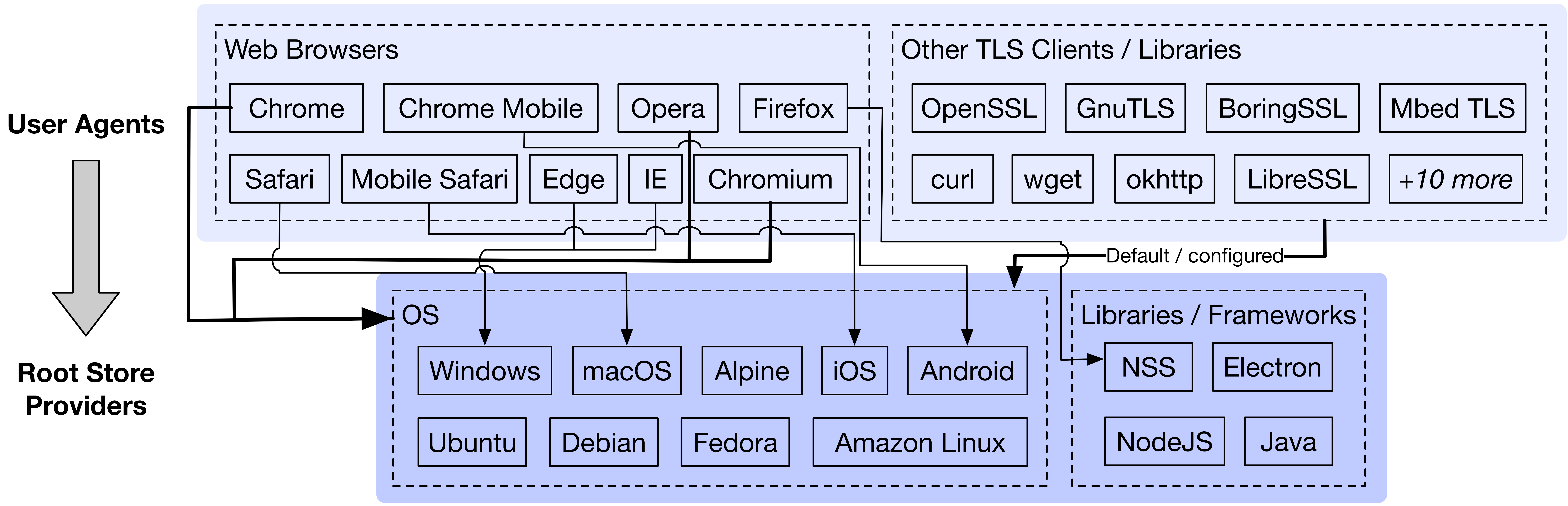
Other TLS Clients / Libraries

OpenSSL	GnuTLS	BoringSSL	Mbed TLS	
curl	wget	okhttp	LibreSSL	*+10 more*

Collected root stores for 83% of global CDN top 200 user agents

Determined default root store for dozens of libraries / TLS clients

Root store providers



Research Questions

1. Which root store providers do TLS user agents rely on?

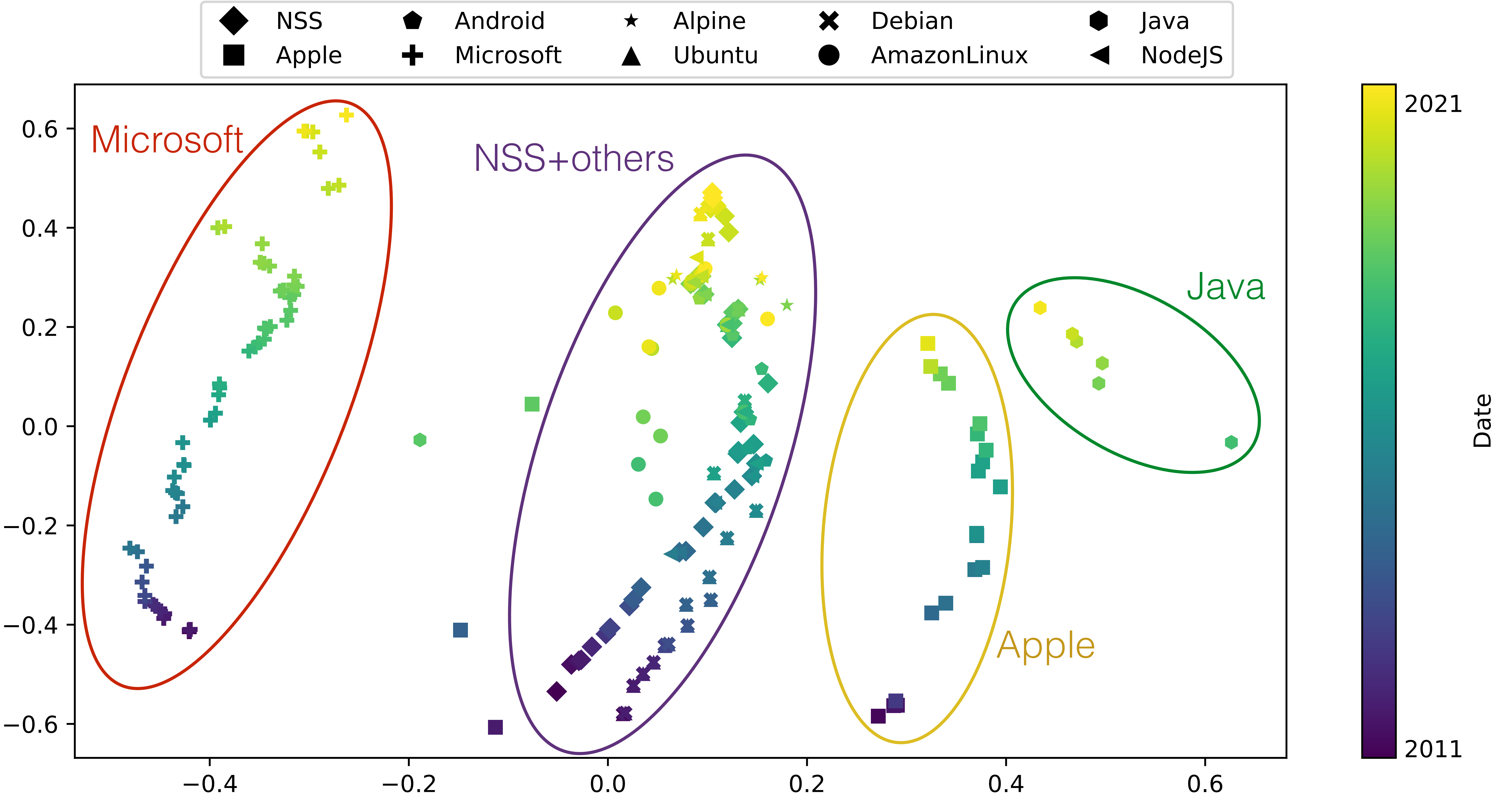
2. How do root store providers determine which CAs to trust?

3. Characterization of root store programs

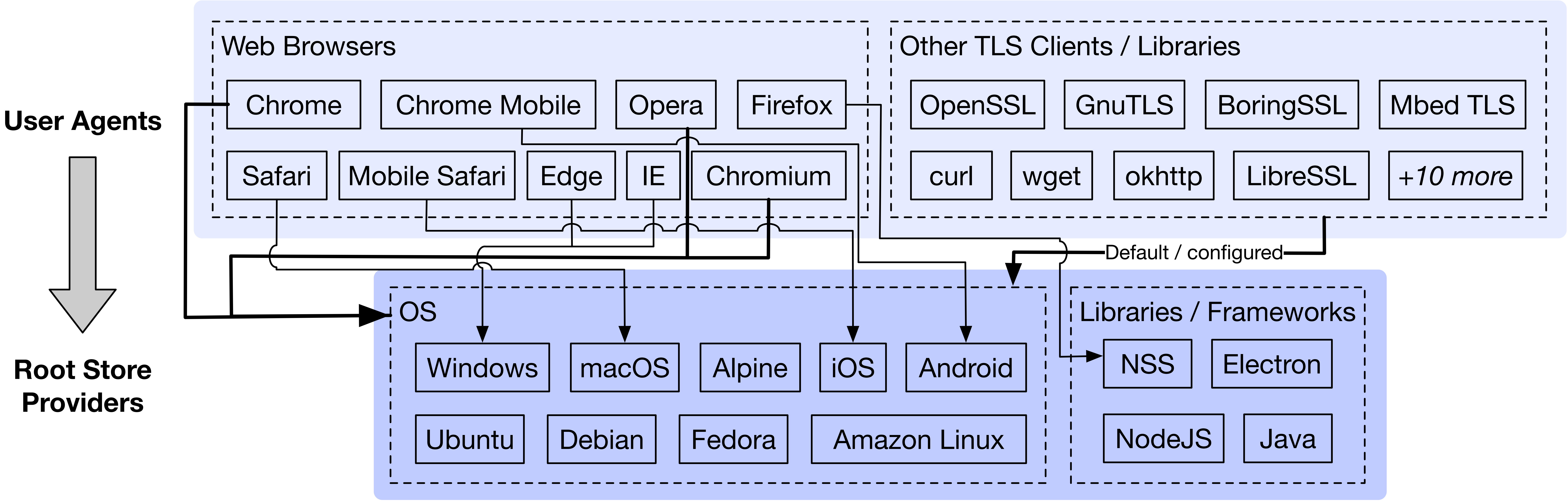
4. How faithfully do providers copy root program trust?

Clustering providers

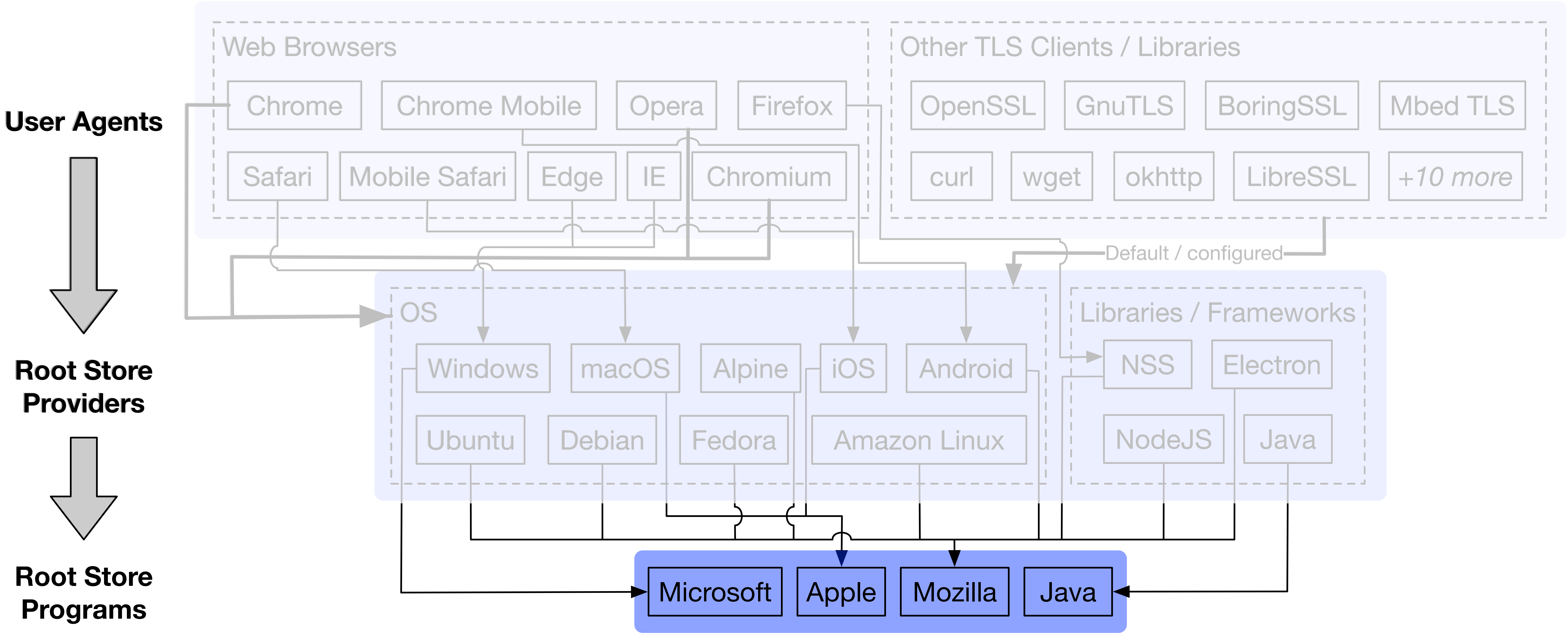
metric MDS



Root store providers



Root store programs



Research Questions

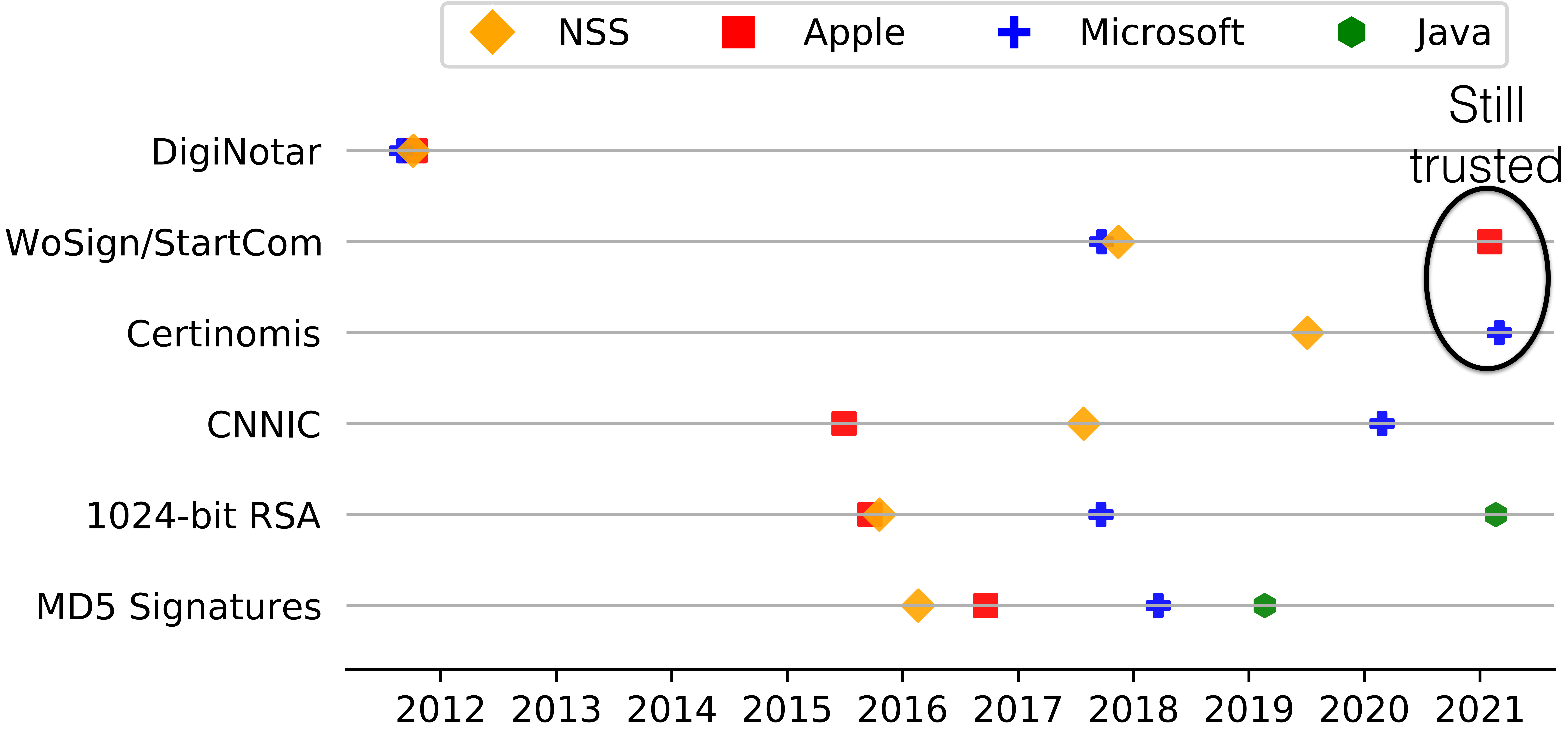
1. Which root store providers do TLS user agents rely on?

2. How do root store providers determine which CAs to trust?

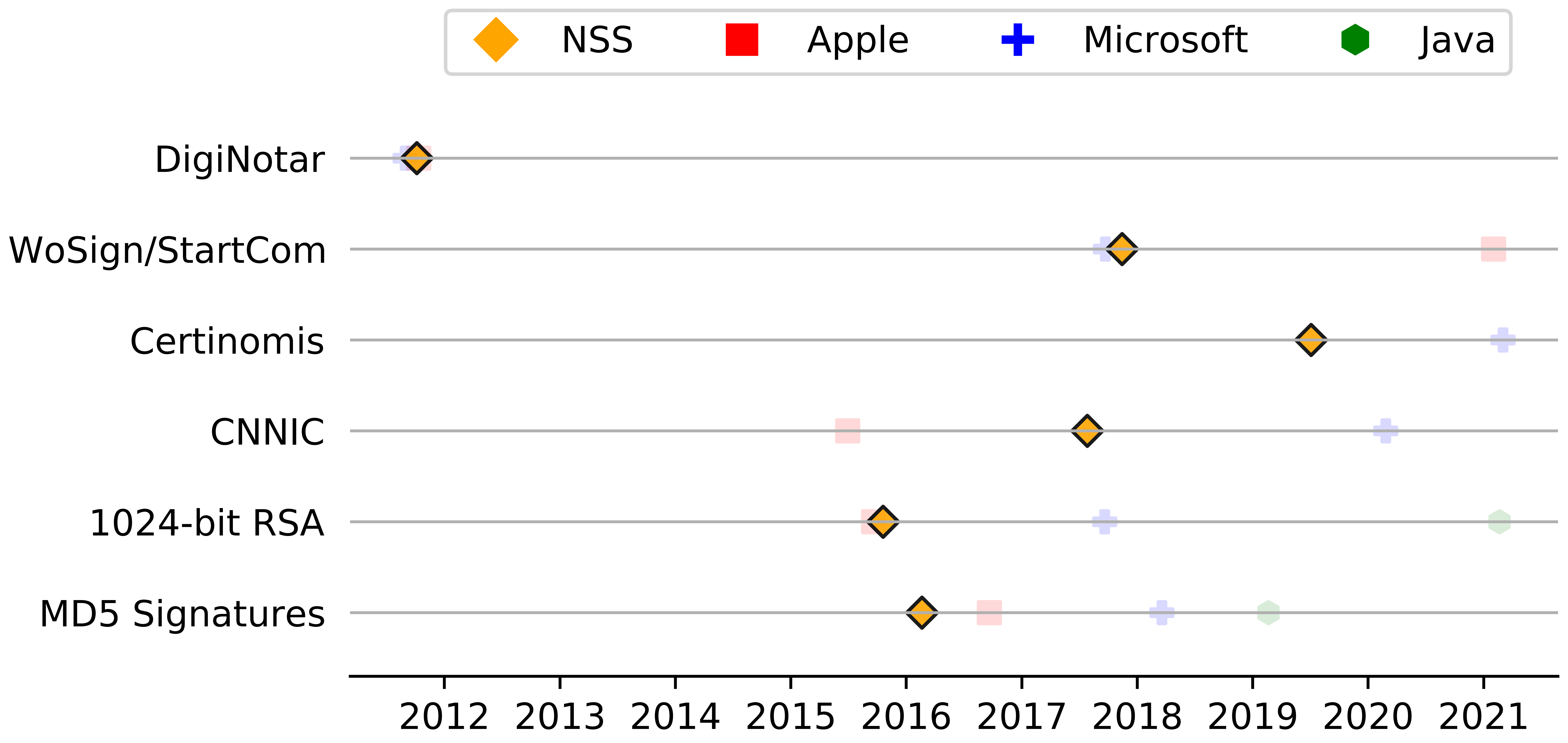
3. Characterization of root store programs

4. How faithfully do providers copy root program trust?

Root program comparison



Root program comparison



Georgia Tech

Root program comparison

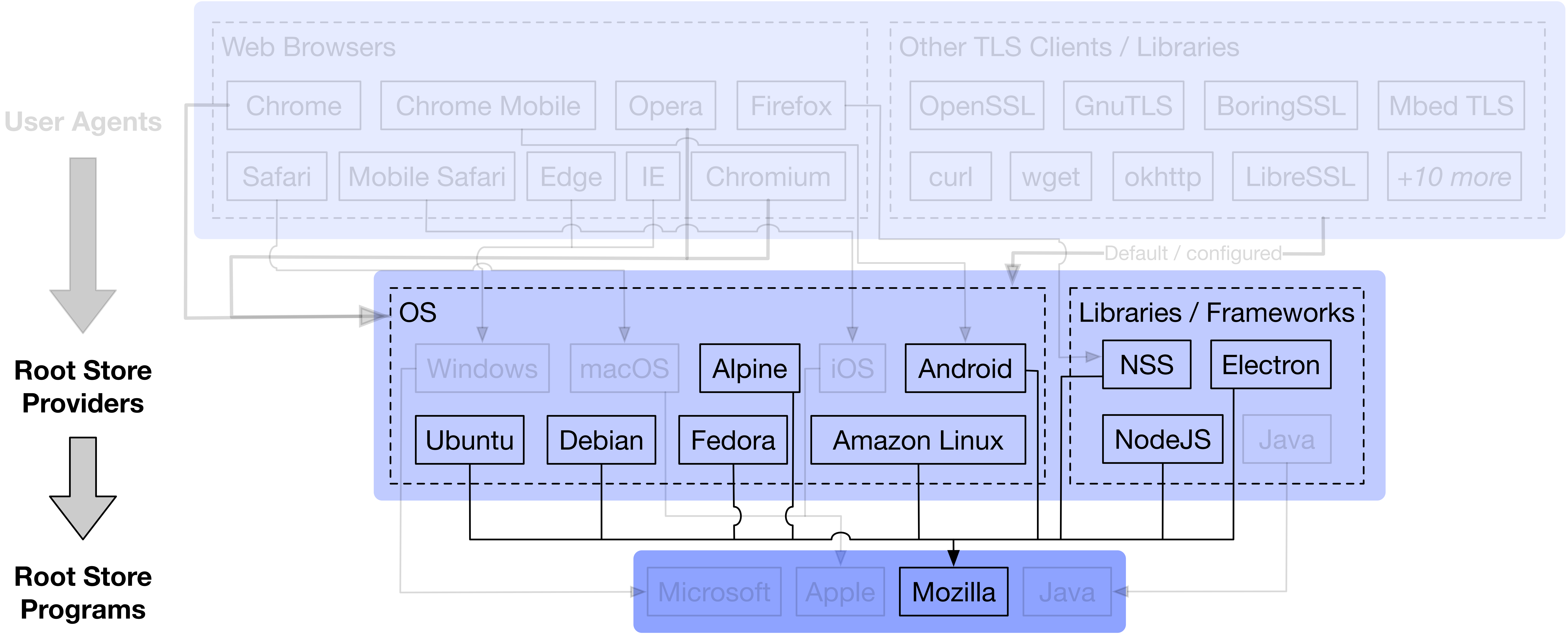
1. Mozilla responds quickly to CA distrust incidents; Microsoft relatively slow, Apple varies.

2. Apple/Mozilla operate relatively hygienic root stores.

3. Size: Mozilla < Apple < Microsoft; Mozilla most restrictive, Microsoft allows government super-CAs.

4. Mozilla runs the most transparent root store program.

Mozilla derivatives

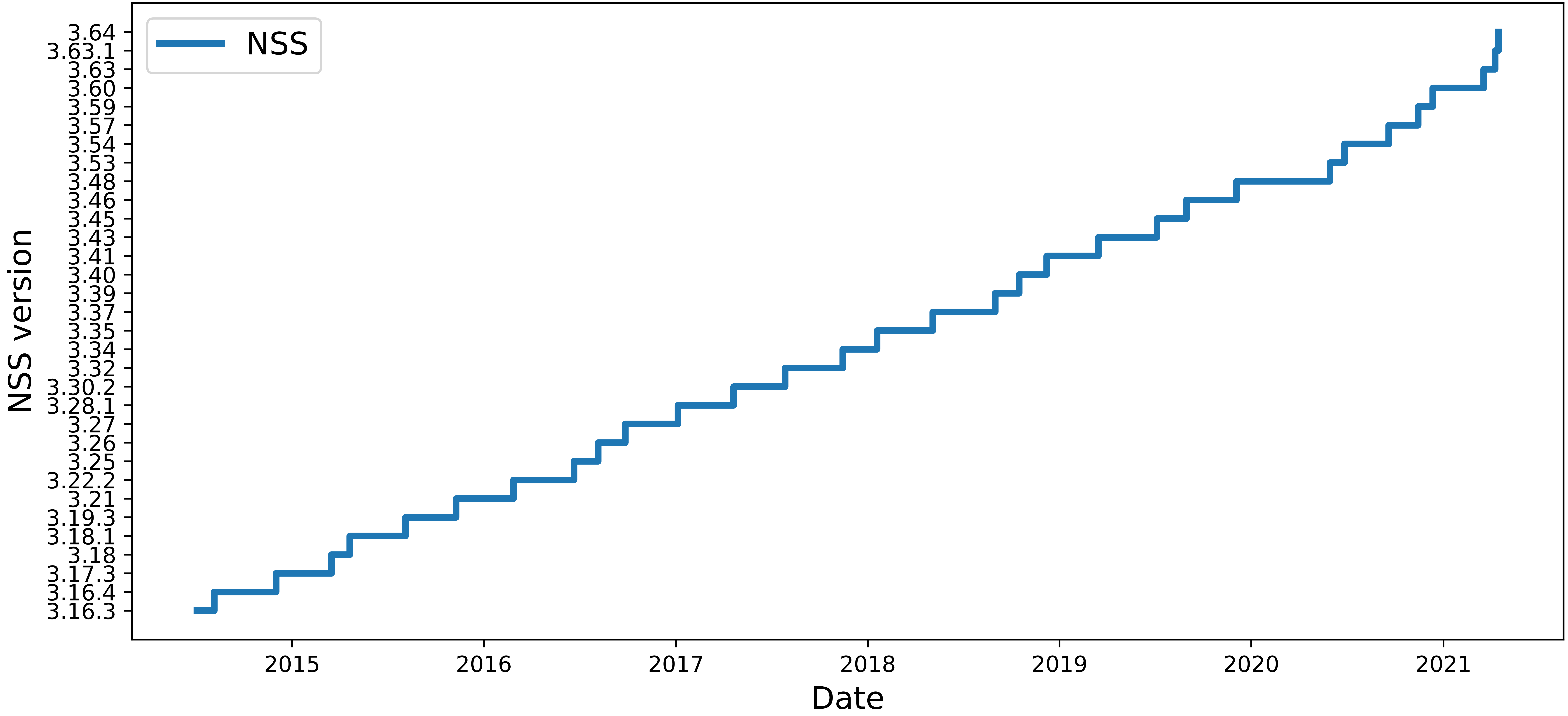


Georgia Tech

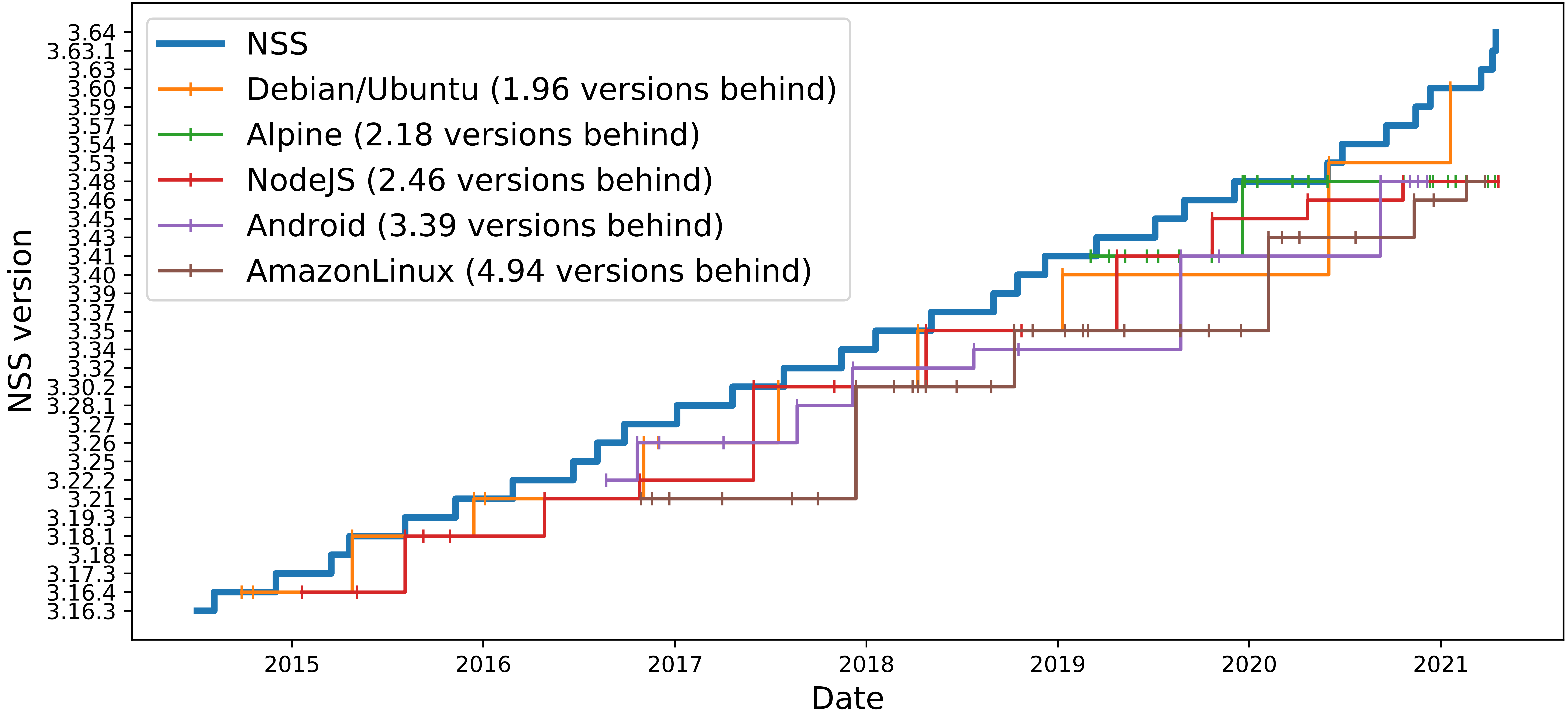
Research Questions

1. Which root store providers do TLS user agents rely on?
2. How do root store providers determine which CAs to trust?
3. Characterization of root store programs
4. How faithfully do providers copy root program trust?

Derivative delay

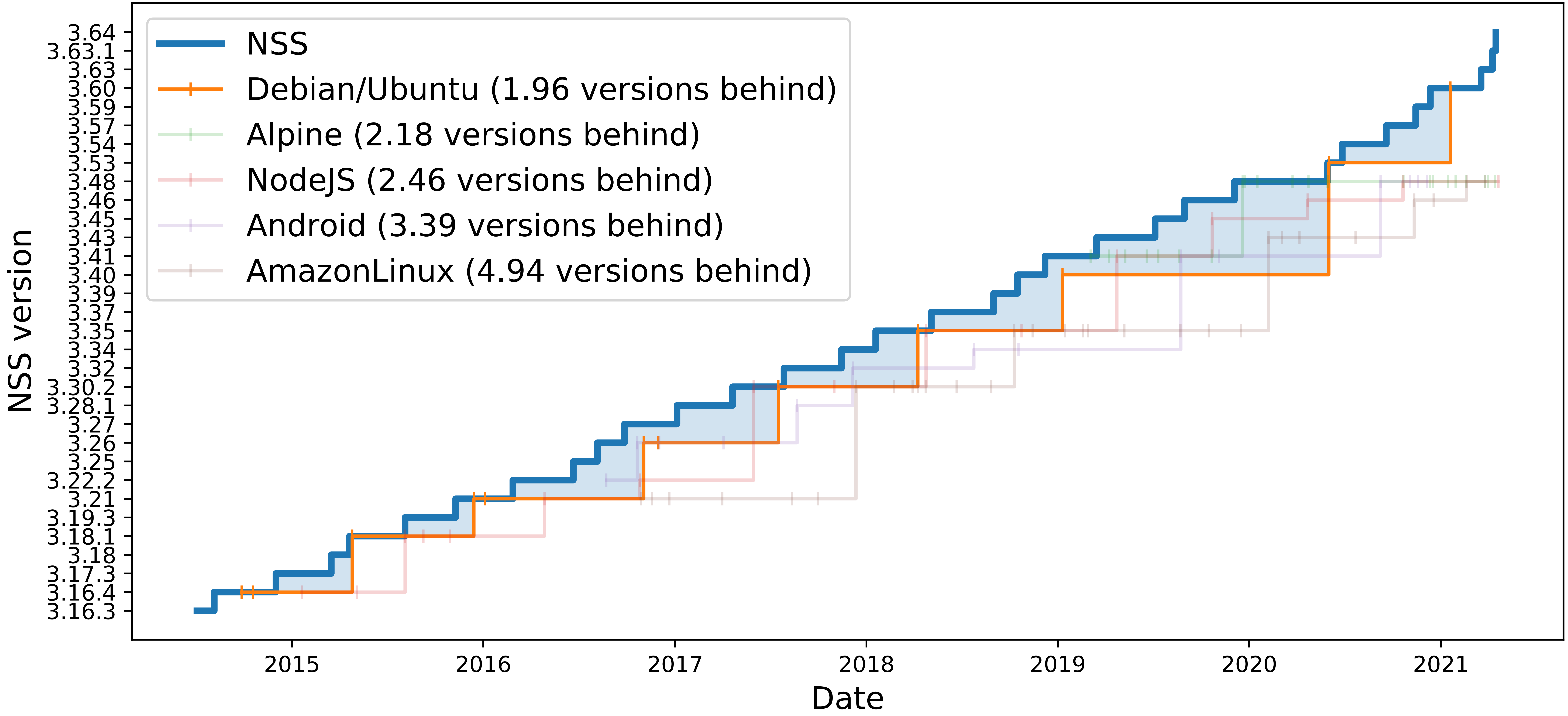


Derivative delay

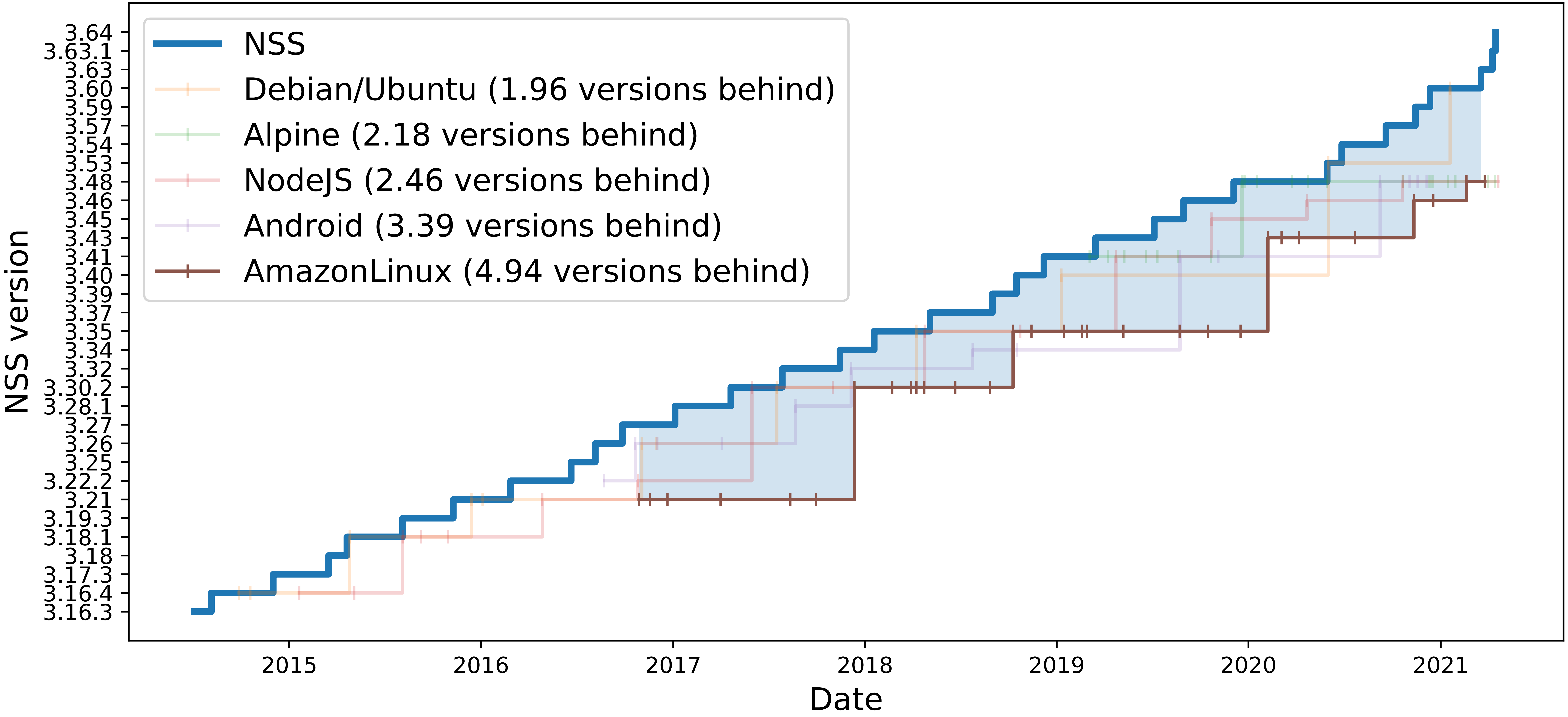


Georgia Tech

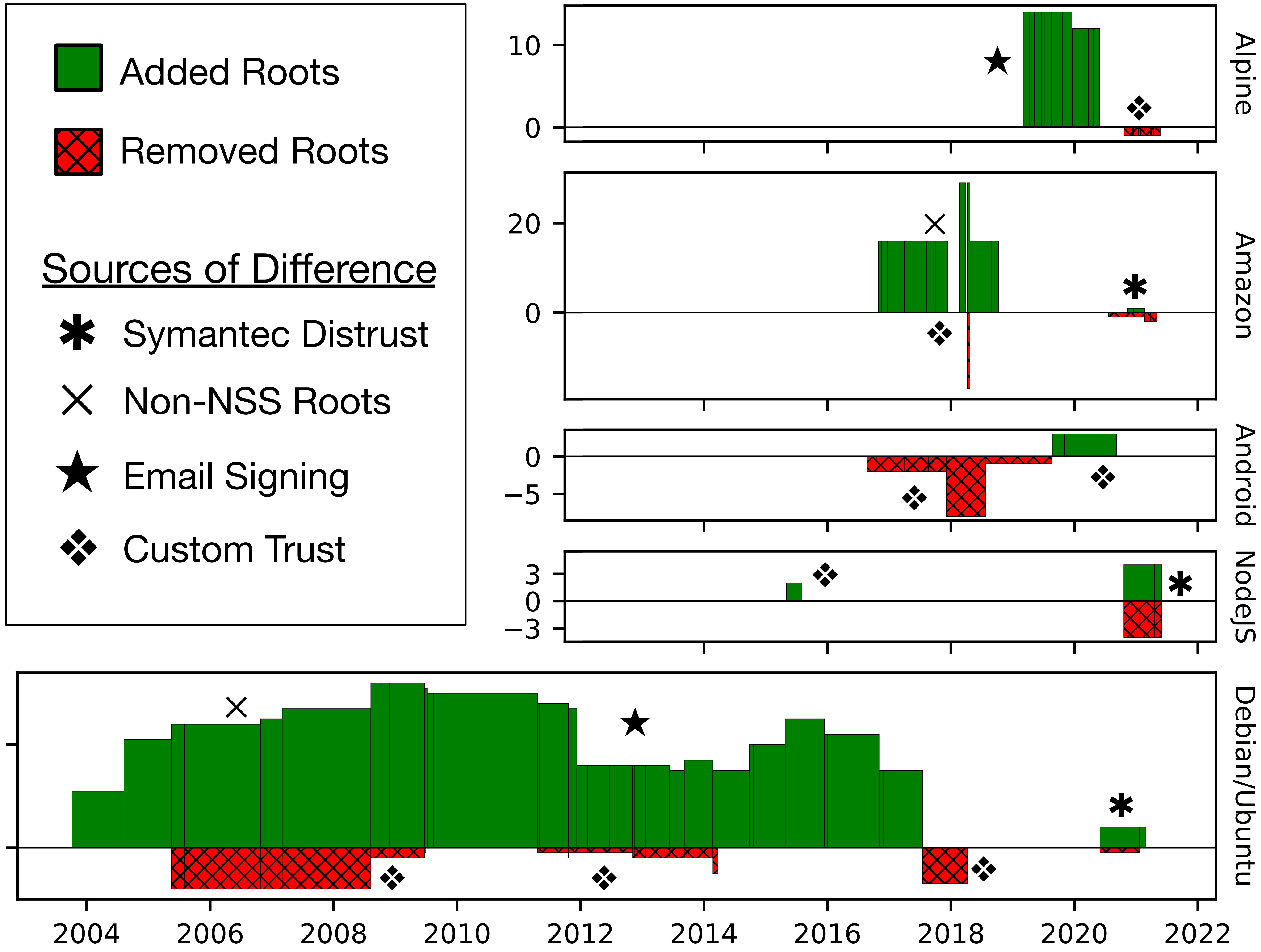
Derivative delay



Derivative delay



Trust deviations



Trust purpose conflation: trusting non-TLS certificates

Partial trust incapability: Symantec distrust dilemma

Non-NSS trusted CAs: questionable trust

App. developer confusion: trusting CAs for code signing, timestamping

Summary

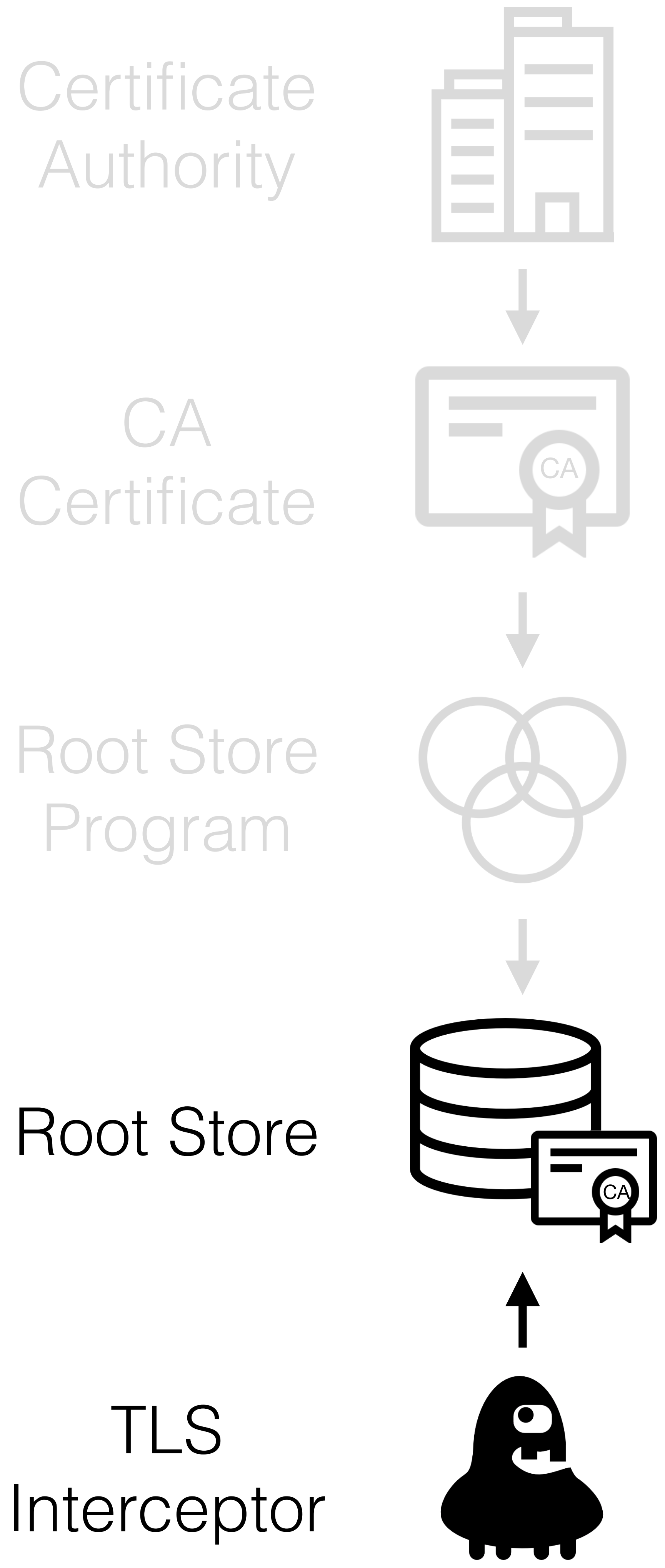
Popular TLS user agents infrequently make their own TLS trust decisions and rely on the OS.

Apple, Microsoft run major root programs; all other root providers originate from Mozilla's NSS root program.

NSS derivatives copy poorly: delayed updates, questionable bespoke trust, incompatible trust scope.

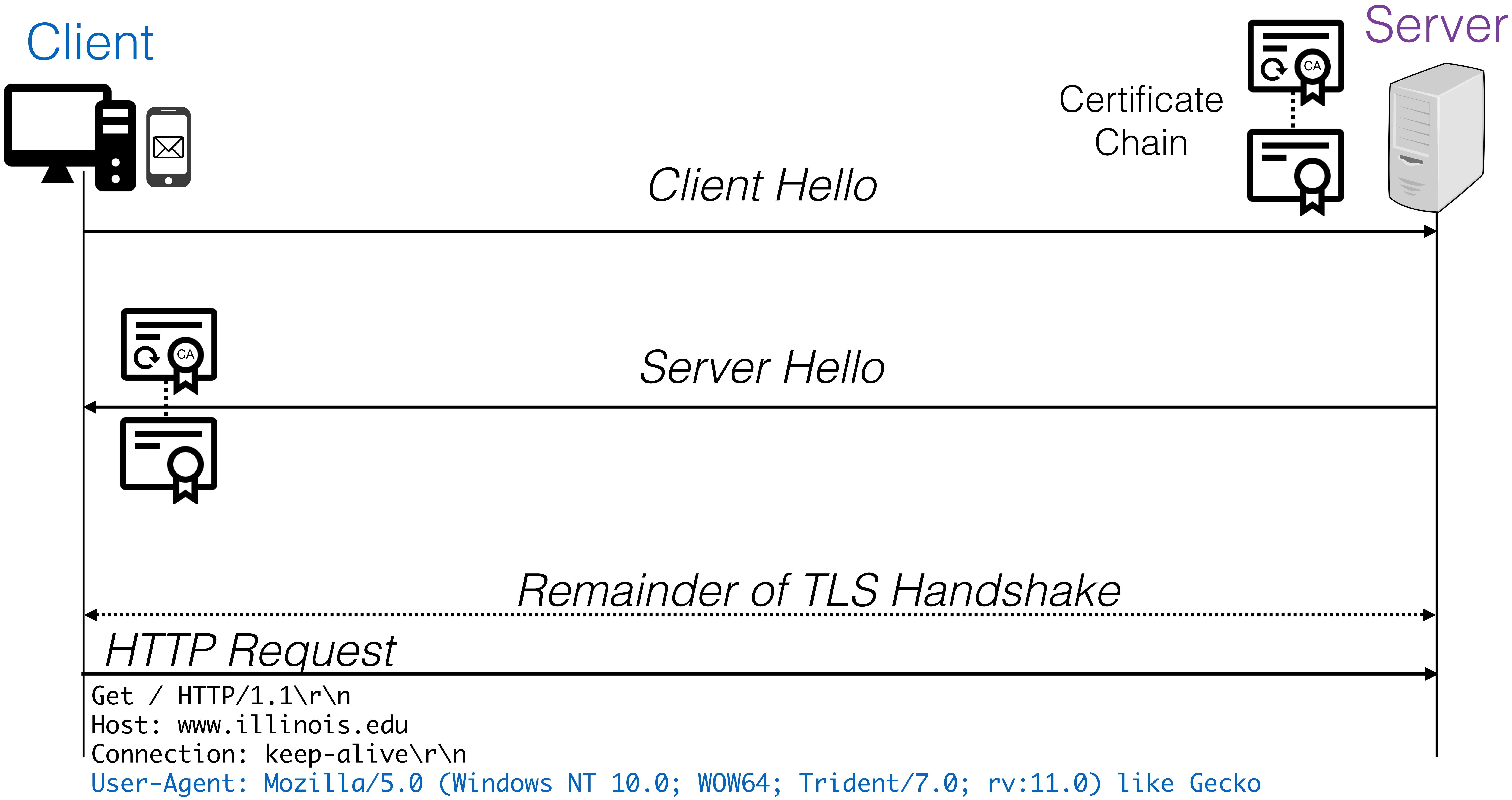
Explicit provenance transparency and consistent trust schema would reduce manual copying, spotlight questionable custom trust.

TLS Interceptor Trust Injection

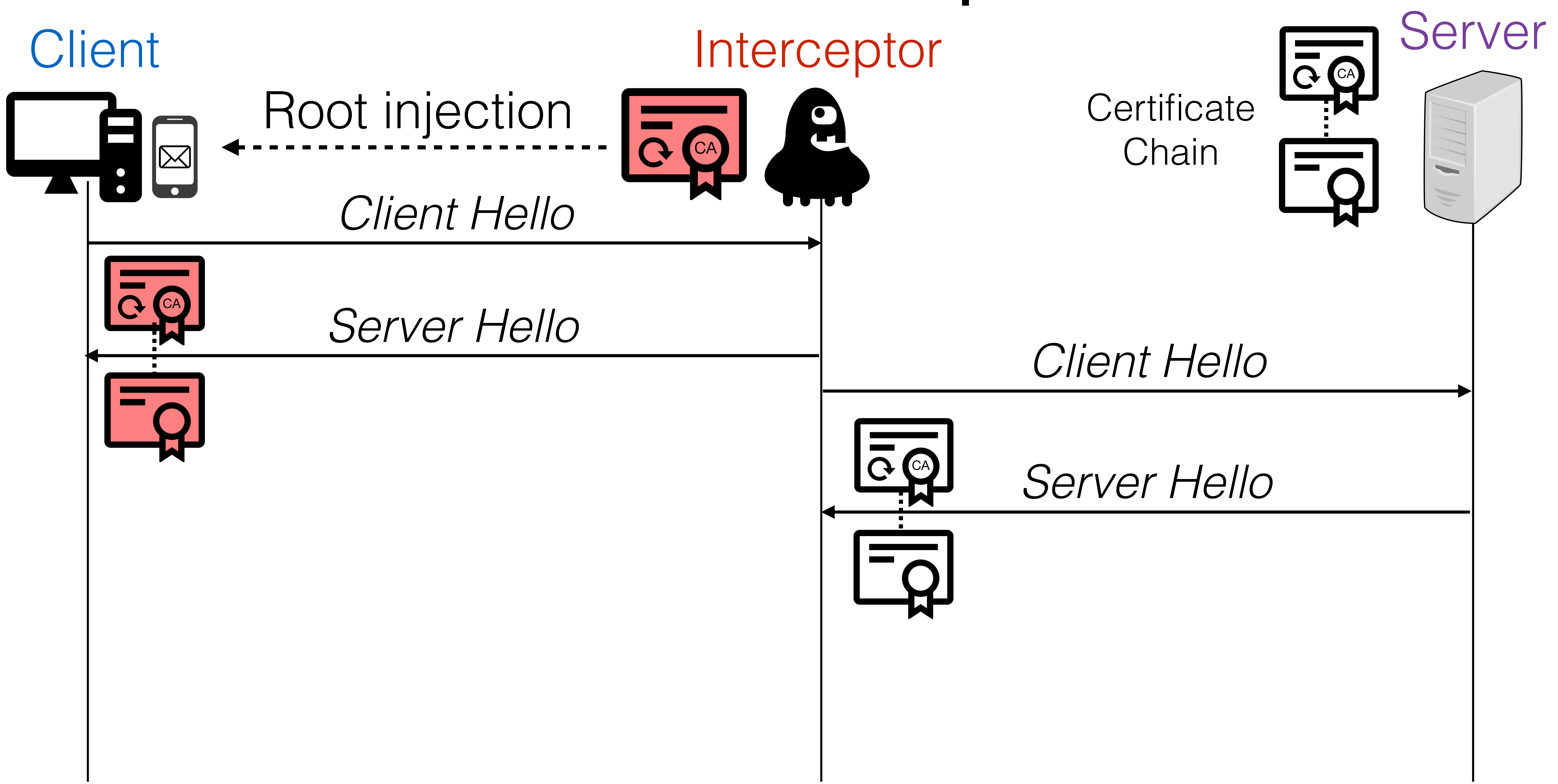


1. Background + Motivation
2. Identifying CA certificate operators *USENIX 2021*
3. Mapping root store provenance *IMC 2021*
4. Detecting TLS interception *NDSS 2017*
5. Conclusions + Future Work

TLS interception

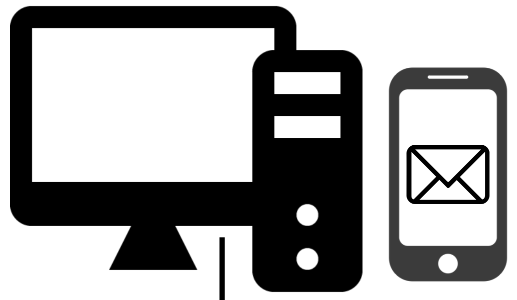


TLS interception



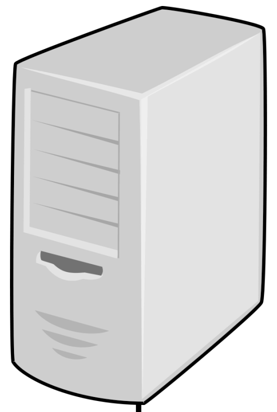
Fingerprinting TLS

Client



Vantage point

Server



Client Hello

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Extension: ec_point_formats
 EC point format: uncompressed
Extension: elliptic_curves
 secp256r1 (0x0017)
 secp256r1 (0x0018)
Extension: Application Layer Protocol Negotiation
```

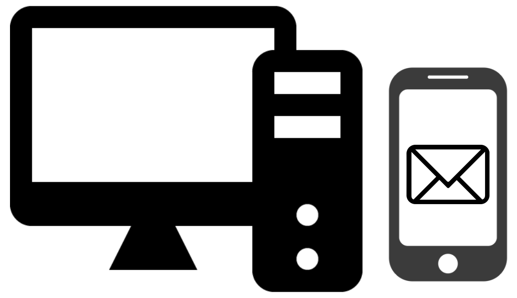
Remainder of TLS Handshake

HTTP Request

```
Get / HTTP/1.1\r\n
Host: www.illinois.edu
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Fingerprinting TLS

Client



Client Hello

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Extension: ec_point_formats
 EC point format: uncompressed
Extension: elliptic_curves
 secp256r1 (0x0017)
 secp256r1 (0x0018)
Extension: Application Layer Protocol Negotiation
```

HTTP Request

```
Get / HTTP/1.1\r\n
Host: www.illinois.edu
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Catalog of TLS fingerprints
for popular TLS clients

Detecting TLS interception

Client

Interceptor

Vantage point

Server

Client Hello (Client → Interceptor)

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Extension: ec_point_formats
 EC point format: uncompressed
Extension: elliptic_curves
 secp256r1 (0x0017)
 secp256r1 (0x0018)
Extension: Application Layer Protocol Negotiation
```

Client Hello (Interceptor → Server)

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
Extension: server_name
```

Remainder of TLS Handshake

HTTP Request

HTTP Request

```
Get / HTTP/1.1\r\n
Host: www.illinois.edu
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Detecting TLS interception

Client

Interceptor

Vantage point

Server

Client Hello

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Extension: ec_point_formats
 EC point format: uncompressed
Extension: elliptic_curves
 secp256r1 (0x0017)
 secp256r1 (0x0018)
Extension: Application Layer Protocol Negotiation
```

Client Hello

```
Version: TLS 1.2 (0x0303)
Cipher Suites
 TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
Extension: server_name
```

Remainder of TLS Handshake

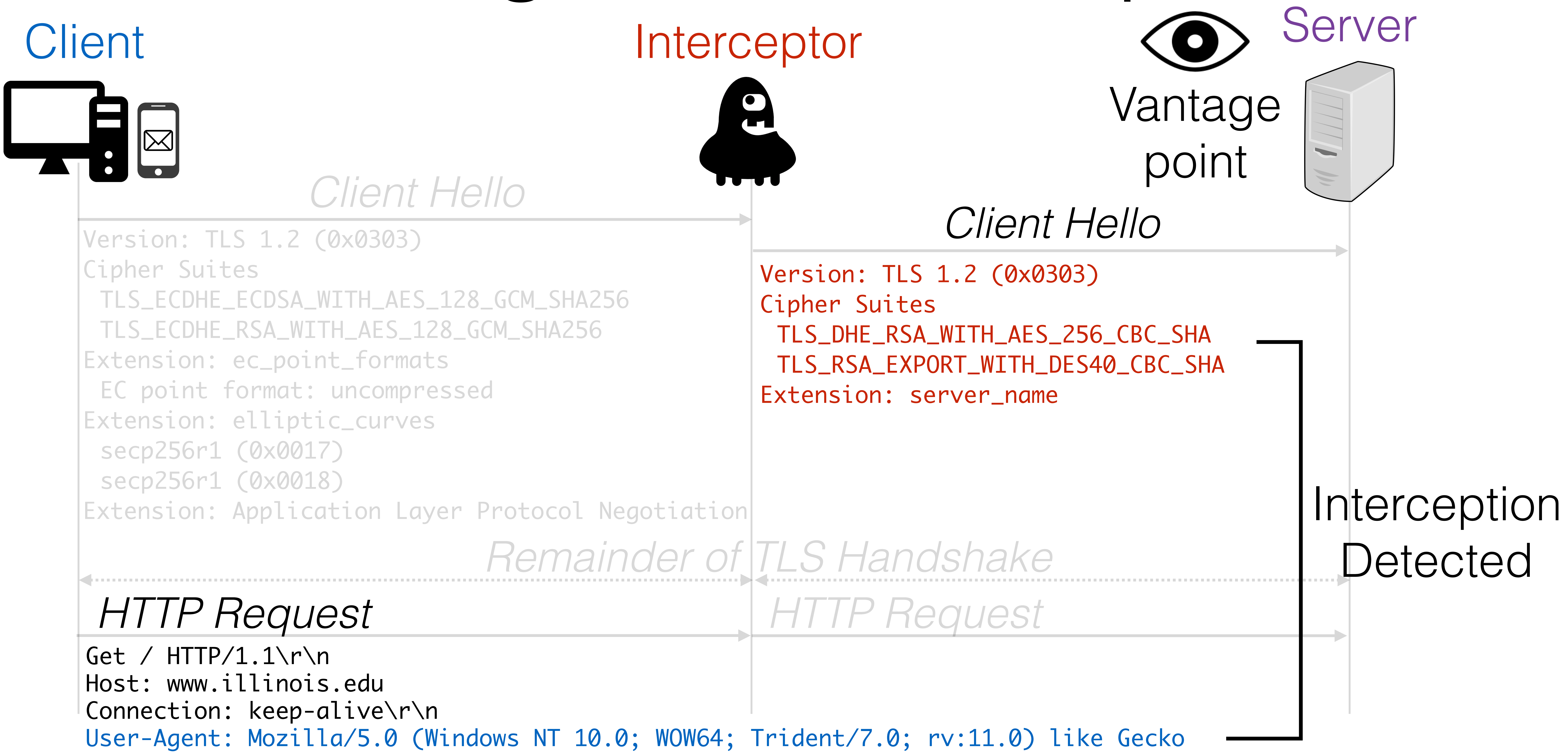
HTTP Request

```
Get / HTTP/1.1\r\n
Host: www.illinois.edu
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

HTTP Request

Expected UA Client Hello

Detecting TLS interception



TLS Interception Rates

Dataset	Description	% of HTTPS connections intercepted
Firefox	Daily update check	4.0%
E-commerce	Invisible pixel load via JavaScript	6.2%
Cloudflare	5% sample of all traffic	10.9%

Attributing Interception

Library of TLS ClientHello fingerprints for known interception products

12 client-side antivirus + parental control software (Windows + Mac)

12 TLS interception middleboxes

Common TLS libraries (e.g. OpenSSL, GnuTLS, BouncyCastle)

Top TLS Interceptors

Dataset	Fingerprint	% Total Interception
Firefox	*Unknown A*	17.1%
	Avast Antivirus	10.8%
	Unknown B	9.4%
	Blue Coat Middlebox	9.1%
	Unknown C	8.3%
E-commerce	Avast Antivirus	9.1%
	AVG Antivirus	7.0%
	Unknown D	6.5%
	Kaspersky Antivirus	5.0%
	BitDefender Antivirus	3.1%
Cloudflare	Bouncy Castle (Android 5)	26.3%
	Bouncy Castle (Android 4)	21.6%
	Unknown E	5.0%
	ESET Antivirus	2.8%
	Dr. Web Antivirus	2.6%

Top TLS Interceptors

Dataset	Fingerprint	% Total Interception
Firefox	*Unknown A*	17.1%
	Avast Antivirus	10.8%
	Unknown B	9.4%
	Blue Coat Middlebox	9.1%
	Unknown C	8.3%
E-commerce	Avast Antivirus	9.1%
	AVG Antivirus	7.0%
	Unknown D	6.5%
	Kaspersky Antivirus	5.0%
	BitDefender Antivirus	3.1%
Cloudflare	Bouncy Castle (Android 5)	26.3%
	Bouncy Castle (Android 4)	21.6%
	Unknown E	5.0%
	ESET Antivirus	2.8%
	Dr. Web Antivirus	2.6%

Top TLS Interceptors

Dataset	Fingerprint	% Total Interception
Firefox	*Unknown A*	17.1%
	Avast Antivirus	10.8%
	Unknown B	9.4%
	Blue Coat Middlebox	9.1%
	Unknown C	8.3%
E-commerce	Avast Antivirus	9.1%
	AVG Antivirus	7.0%
	Unknown D	6.5%
	Kaspersky Antivirus	5.0%
	BitDefender Antivirus	3.1%
Cloudflare	Bouncy Castle (Android 5)	26.3%
	Bouncy Castle (Android 4)	21.6%
	Unknown E	5.0%
	ESET Antivirus	2.8%
	Dr. Web Antivirus	2.6%

Takeaways

4–10% of global HTTPS traffic is intercepted, a tenfold increase over prior detection efforts.

Nearly all interception products increase user exposure to security risks, especially middlebox products (58% severe vulnerabilities) that are typically utilized in corporate networks.

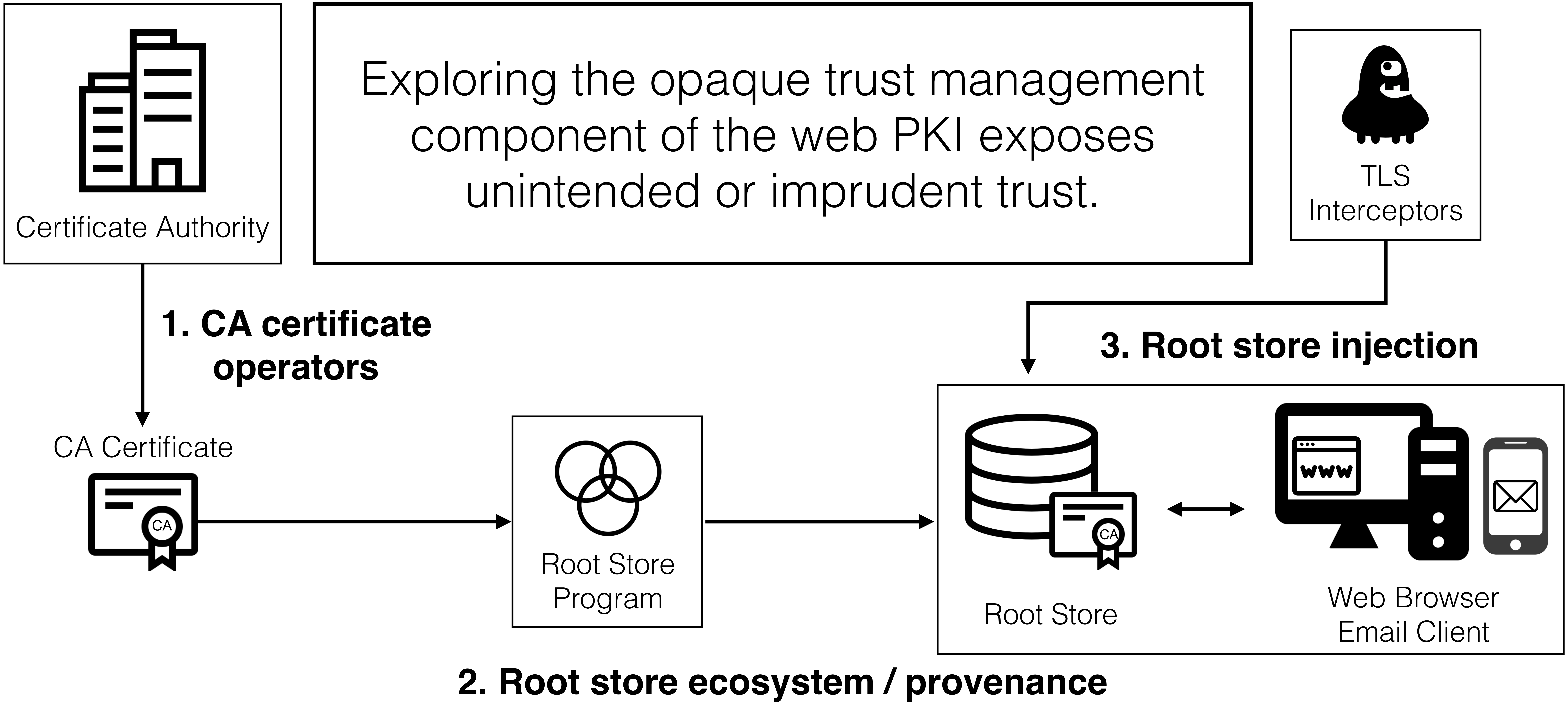
These injected roots are widespread and operated by CAs (i.e., local antivirus/middlebox software) with harmful security practices.

Conclusions + Future Work

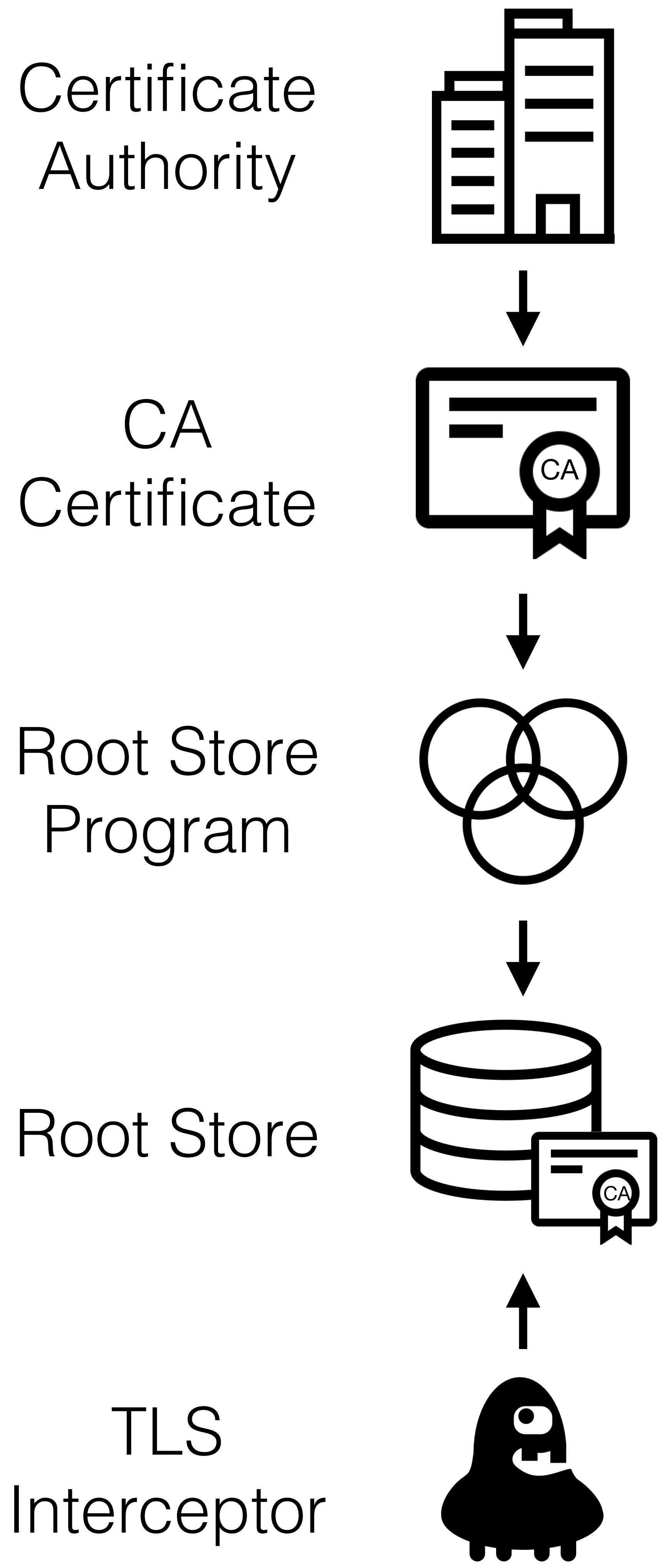
1. Background + Motivation
2. Identifying CA certificate operators *USENIX 2021*
3. Mapping root store provenance *IMC 2021*
4. Detecting TLS interception *NDSS 2017*
5. Conclusions + Future Work

Overview

Exploring the opaque trust management component of the web PKI exposes unintended or imprudent trust.



Summary



1. New fingerprinting methods to detect previously opaque aspects of the web PKI at scale

2. Identification of CA certificate operators

3. Discovering root store provenance in the web PKI

4. Detecting and characterizing the (in)security of TLS interception products

Insights

1. Externalities of extensibility: fingerprinting, ecosystem fracturing, misconfiguration.

2. Digital identity is often a loose proxy for real-life identity.

3. Good trust is hard; imitation is tricky.

4. Trust is everywhere, and secure trust requires transparency.

Future Work

1. Extending web PKI transparency

   Codifying trust policy and automating trust management

   Uncovering the long tail of TLS authentication trust

2. PKI system design

   A history of the web PKI: intentional design or runaway accident?

   PKI in the sky: a comparative evaluation of PKI deployments

   A universal schema for contextual trust sharing

Acknowledgements

2017	NDSS	TLS Interception
2021	USENIX	CA certificate operators
2021	IMC	Root store ecosystem



Not pictured: J. Alex Halderman, Drew Springall, Elie Bursztein, James Austgen, Nick Sullivan, Richard Barnes, Vern Paxson

Other Research Interests

2017	WWW	Web dependencies
2017	USENIX	Mirai botnet
2018	IMC	Ethereum P2P
2019	WWW	Cryptojacking
2020	CSET	HTTPS phishing
2020	CHI	URL Usability
2021	WWW	Websocket adoption

Questions?

Understanding the Trust Relationships of the Web PKI

Zane Ma

https://zanema.com
zanema@gatech.edu