Understanding the Trust Relationships of the Web PKI

Zane Ma

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

Web Server

Email Server

Signs

Leaf Certificate

Understanding the Trust Relationships of the Web PKI  Zane Ma
Understanding the Trust Relationships of the Web PKI

Zane Ma

Web PKI

Trust Management

CA Trust Decisions

Certificate Authority

CA Certificate

Root store inclusion

Web Browser
Email Client

Certificate Issuance

Identity Verification

Leaf Certificate

Signs

Web Server
Email Server

Certificate Issuance

Identity Verification

Web PKI

Trust Management

CA Trust Decisions

Certificate Authority

CA Certificate

Root store inclusion

Web Browser
Email Client

Certificate Issuance

Identity Verification

Leaf Certificate

Signs

Web Server
Email Server
Understanding the Trust Relationships of the Web PKI

Web PKI

Trust Management

Certificate Authority

CA Trust Decisions

Identity Verification

Root store inclusion

Signs

Leaf Certificate

Certificate Issuance

Leaf Certificate

Web Browser

Email Client

Web Server

Email Server

TLS
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:**

**Attacks/defenses:**

**Automating issuance:**

### Protocol bugs / verified implementations:


### Implementation bugs:


---

**TLS**

- 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.++
Research Questions

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

Trust Management

Root Store Program

Root Inclusion

Certificate Authority

CA Certificate

Identity Verification

Signs

Leaf Certificate

Root Store

Web Browser

Email Client

TLS

Web Server

Email Server

Identity Verification

Root Store

Web Browser

Email Client
Research Overview

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

1. CA certificate operators

CA Certificate

Root Store Program

Root Store

Web Browser

Email Client

2. Root store ecosystem / provenance

3. Root store injection

TLS Interceptors
Today’s Talk

1. Background + Motivation

2. Identifying CA certificate operators

3. Mapping root store provenance

4. Detecting TLS interception

5. Conclusions + Future Work
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!

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

Certificates
- Certificate Fingerprints
- foo.com
- AIA/OCSP/CRL
- FQDNS + IPS

Audits
- a2b3c4...
- Certificate SHA256/SHA1

Heuristic CA operator clusters
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

Heuristic CA operator clusters + CCADB Labels \rightarrow Label correction and expansion \rightarrow CA Operator Dataset
Pipeline

Certificates
- foo.com
- Certificate Fingerprints
- AIA/OCSP/CRL
- FQDNS + IPS

Audits
- a2b3c4...
- Certificate SHA256/SHA1

Heuristic CA operator clusters

CCADB Labels

Min. 2-edge Combination

Label correction and expansion

CA Operator Dataset

Audits + Certificates

Heuristic CA operator clusters + CCADB Labels
## Results

<table>
<thead>
<tr>
<th>Discovery</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improperly disclosed Camerfirma subordinate CA (MULTICERT)[1]</td>
<td>Camerfirma removed from Mozilla root store, distrusted by Google products</td>
</tr>
</tbody>
</table>
| 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] |

**CA operational transparency means:**

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

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=1672029  
Characterizing TLS trust

1. Background + Motivation

2. Identifying CA certificate operators

3. Mapping root store provenance

4. Detecting TLS interception

5. Conclusions + Future Work
TLS user agents
TLS user agents

4 different root stores!
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

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

- **User Agents**
  - Chrome
  - Chrome Mobile
  - Opera
  - Firefox
  - Safari
  - Mobile Safari
  - Edge
  - IE
  - Chromium

- **OS**
  - Windows
  - macOS
  - Alpine
  - iOS
  - Android
  - Ubuntu
  - Debian
  - Fedora
  - Amazon Linux

- **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

- **Libraries / Frameworks**
  - NSS
  - Electron
  - NodeJS
  - Java

Default / configured
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

![Diagram showing clustering of providers with metrics and dates](image)

- Microsoft
- NSS + others
- Java
- Apple

Legend:
- NSS
- Android
- Alpine
- Debian
- Java
- AmazonLinux
- NodeJS

Metric MDS

Date

2011 - 2021
Root store providers

- 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

- OS
  - Windows
  - macOS
  - Alpine
  - iOS
  - Android
  - Ubuntu
  - Debian
  - Fedora
  - Amazon Linux

- Libraries / Frameworks
  - NSS
  - Electron
  - NodeJS
  - Java

Default / configured
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

- NSS
- Apple
- Microsoft
- Java

- DigiNotar
- WoSign/StartCom
- Certinomis
- CNNIC
- 1024-bit RSA
- MD5 Signatures

Still trusted
Root program comparison

- NSS
- Apple
- Microsoft
- Java

DigiNotar
WoSign/StartCom
Certinomis
CNNIC
1024-bit RSA
MD5 Signatures

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.
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

NSS
Debian/Ubuntu (1.96 versions behind)
Alpine (2.18 versions behind)
NodeJS (2.46 versions behind)
Android (3.39 versions behind)
AmazonLinux (4.94 versions behind)
Derivative delay
Derivative delay

- NSS
- Debian/Ubuntu (1.96 versions behind)
- Alpine (2.18 versions behind)
- NodeJS (2.46 versions behind)
- Android (3.39 versions behind)
- AmazonLinux (4.94 versions behind)
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

3. Mapping root store provenance

4. Detecting TLS interception

5. Conclusions + Future Work
TLS interception

Client

Server

Certificate Chain

Client Hello

Server Hello

Remainder of TLS Handshake

HTTP Request

Get / HTTP/1.1
Host: www.illinois.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Understanding the Trust Relationships of the Web PKI

Zane Ma

TLS interception

Client Hello

Server Hello

Interceptor

Root injection

Client

Server

Certificate Chain

CA

Client Hello

Server Hello
Fingerprinting TLS

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
Host: www.illinois.edu
Connection: keep-alive
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
Host: www.illinois.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```
Detecting TLS interception

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
Host: www.illinois.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Detecting TLS interception

**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
- Host: www.illinois.edu
- Connection: keep-alive
- User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko

---

**Interceptor**

- **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**

---

**Server**

- **Expected UA**
  - Client Hello

---

**Vantage point**
Detecting TLS interception

Client

Interceptor

Server

![Diagram of TLS interception](image)

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

**Interceptor**

- 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**

**Interception Detected**
## TLS Interception Rates

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>% of HTTPS connections intercepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>Daily update check</td>
<td>4.0%</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Invisible pixel load via JavaScript</td>
<td>6.2%</td>
</tr>
<tr>
<td>Cloudflare</td>
<td>5% sample of all traffic</td>
<td>10.9%</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fingerprint</th>
<th>% Total Interception</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>Unknown A</td>
<td>17.1%</td>
</tr>
<tr>
<td></td>
<td>Avast Antivirus</td>
<td>10.8%</td>
</tr>
<tr>
<td></td>
<td>Unknown B</td>
<td>9.4%</td>
</tr>
<tr>
<td></td>
<td>Blue Coat Middlebox</td>
<td>9.1%</td>
</tr>
<tr>
<td></td>
<td>Unknown C</td>
<td>8.3%</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Avast Antivirus</td>
<td>9.1%</td>
</tr>
<tr>
<td></td>
<td>AVG Antivirus</td>
<td>7.0%</td>
</tr>
<tr>
<td></td>
<td>Unknown D</td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>Kaspersky Antivirus</td>
<td>5.0%</td>
</tr>
<tr>
<td></td>
<td>BitDefender Antivirus</td>
<td>3.1%</td>
</tr>
<tr>
<td>Cloudflare</td>
<td>Bouncy Castle (Android 5)</td>
<td>26.3%</td>
</tr>
<tr>
<td></td>
<td>Bouncy Castle (Android 4)</td>
<td>21.6%</td>
</tr>
<tr>
<td></td>
<td>Unknown E</td>
<td>5.0%</td>
</tr>
<tr>
<td></td>
<td>ESET Antivirus</td>
<td>2.8%</td>
</tr>
<tr>
<td></td>
<td>Dr. Web Antivirus</td>
<td>2.6%</td>
</tr>
</tbody>
</table>
# Top TLS Interceptors

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fingerprint</th>
<th>% Total Interception</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Firefox</strong></td>
<td><em>Unknown A</em></td>
<td>17.1%</td>
</tr>
<tr>
<td></td>
<td>Avast Antivirus</td>
<td>10.8%</td>
</tr>
<tr>
<td></td>
<td><em>Unknown B</em></td>
<td>9.4%</td>
</tr>
<tr>
<td></td>
<td>Blue Coat Middlebox</td>
<td>9.1%</td>
</tr>
<tr>
<td></td>
<td><em>Unknown C</em></td>
<td>8.3%</td>
</tr>
<tr>
<td><strong>E-commerce</strong></td>
<td>Avast Antivirus</td>
<td>9.1%</td>
</tr>
<tr>
<td></td>
<td>AVG Antivirus</td>
<td>7.0%</td>
</tr>
<tr>
<td></td>
<td><em>Unknown D</em></td>
<td>6.5%</td>
</tr>
<tr>
<td></td>
<td>Kaspersky Antivirus</td>
<td>5.0%</td>
</tr>
<tr>
<td></td>
<td>BitDefender Antivirus</td>
<td>3.1%</td>
</tr>
<tr>
<td><strong>Cloudflare</strong></td>
<td>Bouncy Castle (Android 5)</td>
<td>26.3%</td>
</tr>
<tr>
<td></td>
<td>Bouncy Castle (Android 4)</td>
<td>21.6%</td>
</tr>
<tr>
<td></td>
<td><em>Unknown E</em></td>
<td>5.0%</td>
</tr>
<tr>
<td></td>
<td>ESET Antivirus</td>
<td>2.8%</td>
</tr>
<tr>
<td></td>
<td>Dr. Web Antivirus</td>
<td>2.6%</td>
</tr>
</tbody>
</table>
# Top TLS Interceptors

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

1. CA certificate operators

2. Root store ecosystem / provenance

3. Root store injection

Certiﬁcate Authority

Root Store Program

Root Store

Web Browser
Email Client

CA Certificate

TLS Interceptors
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