CSE127, Computer Security

Cryptography III — TLS + HTTPS

UC San Diego

Housekeeping

General course things to know

- PA5 released!
  - Focuses on Cryptography!
  - A little delay in release, so extended deadline… now due **3/14 EOD**
- Note, due to travel class is cancelled on **3/10!**
- SETs are now available! We will take some time in the last lecture to fill them out. But please do them, they are very helpful to me.

Final Exam Logistics

- **Final exam time:** Thursday, **3/19** at **8am**
- **Final exam location:** Mosaic Lecture Hall 113 (has 250 seats so we don’t need to be so cramped!)

Final Exam Details

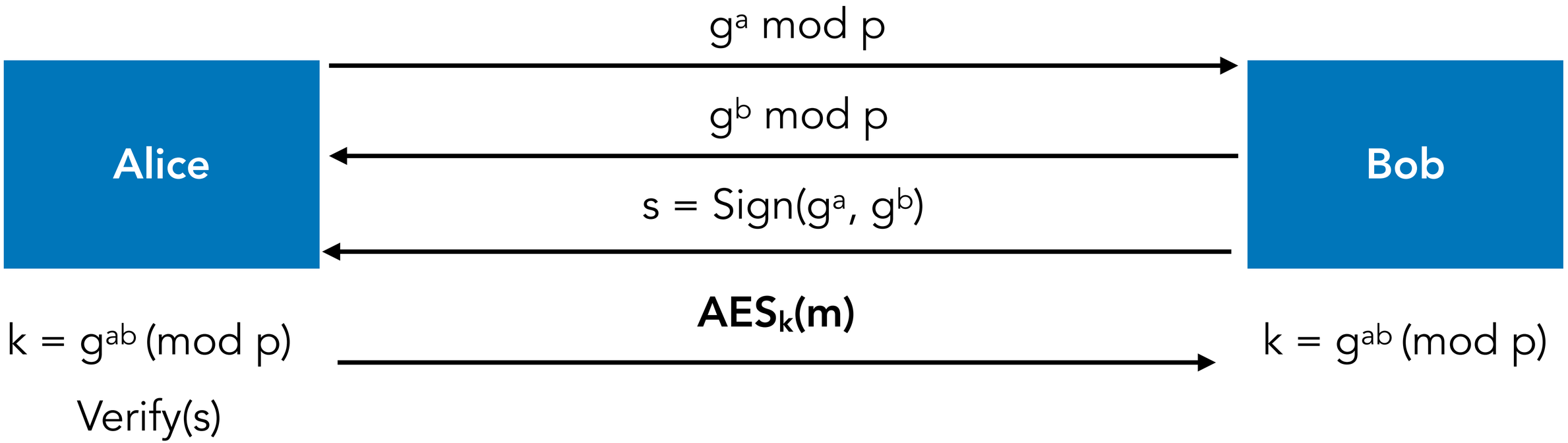
- Same format as the midterm: MCQ, SA, PA questions
- MCQ and SA are comprehensive over the entire class
  - My plan is to include ~5 – 10% of the midterm questions here; think of it like a “second chance.” **Looking over the midterm is a very good way to refresh yourself on the first half of the class!**
- PA questions will focus on PA4 and PA5
  - Best way to study for these is to refresh what you did (do) on PAs
  - Similarly, midterm will test your knowledge of PA material + extend a little bit
- No “practice” final

Previously on CSE 127…

Recap

- We've gone through simple cryptographic primitives, symmetric + asymmetric cryptography, key exchange, authentication….
- But we have a big problem!

The problem: bootstrapping



Key exchange let's us negotiate a symmetric key in the clear, and public key crypto lets us verify authenticity…. **but where do we get the public key from?**

The core of the problem

- At its core, the fundamental issue in cryptography is the same as the fundamental problem in computer security: **who do I trust to get the public key material from?**
  - I'd like to trust Bob, but how do I *know for sure* that I am communicating with Bob over the Internet?
    - In person, this is somewhat straightforward
    - Online, this requires other strategies to establish **trust…**

Today's lecture: TLS, PKI, and real trust

- Learn about Transport Layer Security — **the most widely deployed cryptographic protocol in the world.**
  - Basically the final boss of this class: combines systems, web, network, and cryptography
- Discuss the history of TLS, how it came to where it is today, and all the places where TLS is deployed
- Learn about *public key infrastructure* – how we **actually** enable authentication and trust on the Internet, and the different strategies we take for establishing trust
  - Specifically, we'll talk about the HTTPS PKI and the organizational bureaucracy that keeps us all secure on the web

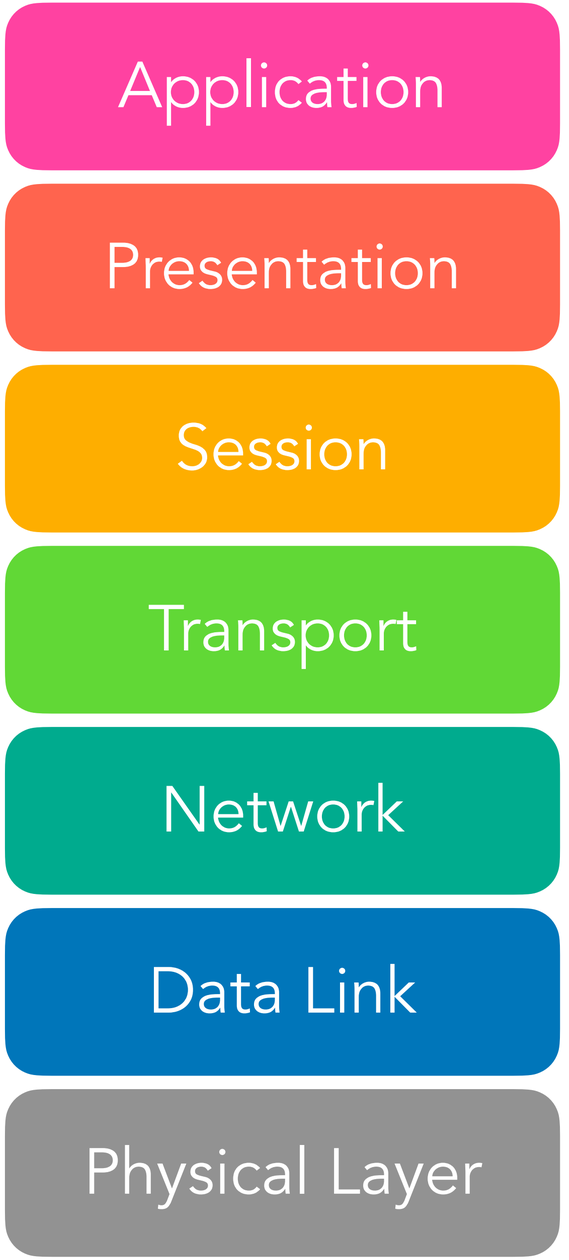
TLS

Transport Layer Security (TLS)

- TLS — a protocol that provides **data confidentiality, data integrity**, and **identity authentication** all in one!
  - We have already learned the basic tools to enable confidentiality and integrity… remember *encryption* and *signatures*
- **Authentication remains the last big hurdle!**
- TLS is *protocol agnostic* — it most commonly runs on top of HTTP (HTTPS), but it can secure any protocol, e.g.,
  - SMTPS (SMTP Secure), FTPS (File Transfer Protocol Secure), DoT (DNS over TLS), DoH (DNS over HTTPS)… the list goes on
  - Mostly today we'll be talking about HTTPS

TLS in OSI Model

- TLS kind of breaks the neat OSI model of computer networking…
- TLS requires reliable transport, so in that sense, it's > L4 (where TCP lives)
- TLS also *encrypts* application layer data, so in that sense, it's L7 (where HTTP lives)
- …But TLS also establishes sessions (so L5?) and also formats data (so L6?)…
  - Basically, the model doesn't really work, but it's somewhere between L5 – L7 depending on what it's doing



A brief history of TLS Part 1

- **November 1994 — Secure Sockets Layer V2** released by Netscape, to encrypt this hot new thing called “the web”; deprecated in 2011
- **November 1995 – SSL V3**; SSL V2 was horribly busted, relied on MD5, shared keys for encryption /authentication…. etc; deprecated in 2015
- **January 1999 — TLS v1**; IETF group established for working on secure web (no longer relying on a single browser vendor to do this…)
  - Name change sparked because of 1990s browser wars…
  - Fairly decent protocol, only deprecated in 2020, but fundamentally tied to weak and broken cryptographic protocols

https://www.feistyduck.com/ssl-tls-and-pki-history/

A brief history of TLS Part 2

- **April 2006 — TLS v1.1**; New TLS, updated cryptographic schemes, protection against some attacks against **TLS 1.0** (that won't be learned about until 2011!); deprecated in 2020

- **August 2008 — TLS v1.2**; still very widely used today, and the resulting ten years is a litany of deployment challenges and victories, attacks and defenses, and lots of exciting new ideas

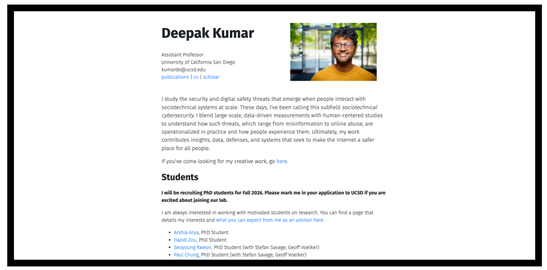
- **August 2018 — TLS v1.3** is released, and the deployment and rollout continues to this day!

- According to SSLLabs, as of June 2025, ~75% of websites they survey support TLS 1.3 (compared to 100% TLS 1.2 support)

https://www.feistyduck.com/ssl-tls-and-pki-history/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Alice



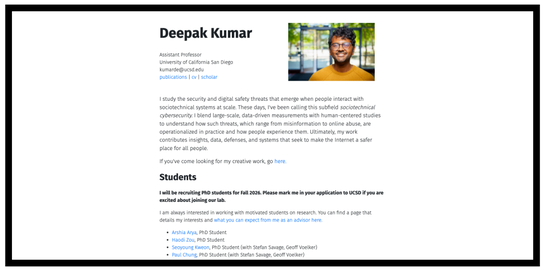
kumarde.com

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 1: The client says hello!

client hello: client random

[list of cipher suites]



Alice

kumarde.com

- **Client hello includes…** protocol version, client random data, cipher suits, compression methods, extensions — basically, “I’m the client and here’s what I can do”

https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

```
TLS_ECDHE_RSA_WITH_CHAHA20_POLY1305_SHA256
```


https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

```
TLS_ECDHE_RSA_WITH_CHAHA20_POLY1305_SHA256
```

- I speak the TLS protocol

https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

`TLS_**ECDHE**_RSA_WITH_CHAHA20_POLY1305_SHA256`

- I speak the TLS protocol
- I want to use elliptic curve diffie hellman ephemeral key exchange

https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

`TLS_ECDHE_RSA_WITH_CHAHA20_POLY1305_SHA256`

- I speak the TLS protocol
- I want to use elliptic curve Diffie-Hellman ephemeral key exchange
- I want to use RSA authentication…

https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

`TLS_ECDHE_RSA_`**`WITH_CHAHA20_POLY1305`**`_SHA256`

- I speak the TLS protocol
- I want to use elliptic curve Diffie-Hellman ephemeral key exchange
- I want to use RSA authentication…
  - with a ChaCha stream cipher for encryption and Poly1305 authenticator for MACs

https://tls12.xargs.org/

Cipher Suites

- How clients and servers communicate about what cryptography they can functionally do.

`TLS_ECDHE_RSA_WITH_CHAHA20_POLY1305_`**`SHA256`**

- I speak the TLS protocol
- I want to use elliptic curve Diffie-Hellman ephemeral key exchange
- I want to use RSA authentication…
  - with a ChaCha stream cipher for encryption and Poly1305 authenticator for MACs
- And SHA256 as my hash function!s

https://tls12.xargs.org/

Cipher Suites are encoded by the protocol

Cipher Suites

The client provides an ordered list of which cryptographic methods it will support for key exchange, encryption with that exchanged key, and message authentication. The list is in the order preferred by the client, with highest preference first.

- `00 20` - 0x20 (32) bytes of cipher suite data
- `cc a8` - assigned value for `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`
- `cc a9` - assigned value for `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`
- `c0 2f` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `c0 30` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `c0 2b` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `c0 2c` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `c0 13` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
- `c0 09` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
- `c0 14` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `c0 0a` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `00 9c` - assigned value for `TLS_RSA_WITH_AES_128_GCM_SHA256`
- `00 9d` - assigned value for `TLS_RSA_WITH_AES_256_GCM_SHA384`
- `00 2f` - assigned value for `TLS_RSA_WITH_AES_128_CBC_SHA`
- `00 35` - assigned value for `TLS_RSA_WITH_AES_256_CBC_SHA`
- `c0 12` - assigned value for `TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `00 0a` - assigned value for `TLS_RSA_WITH_3DES_EDE_CBC_SHA`

https://tls12.xargs.org/

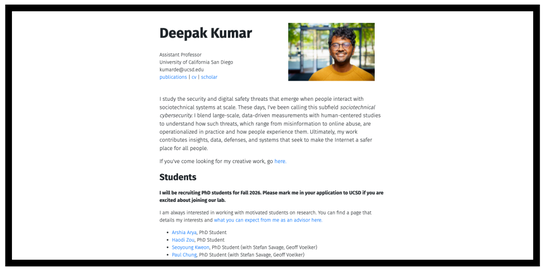
TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 2: The server says hello back

client hello: client random

[list of cipher suites]

server hello: server random, [cipher suite]



Alice

kumarde.com

- Server picks the version, and cryptography used in all downstream communication

https://tls12.xargs.org/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 3: The server sends a certificate

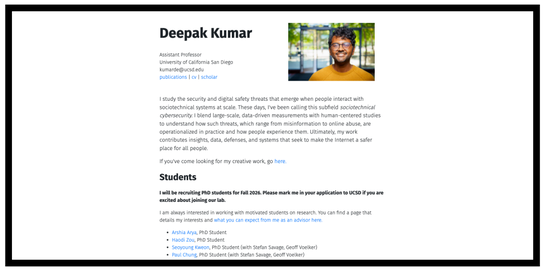
Alice

client hello: client random

[list of cipher suites]

server hello: server random, [cipher suite]

server certificate, verifying authenticity



kumarde.com

https://tls12.xargs.org/

Certificates and Certificate Authorities

- How do Alice **actually know** that she's talking to kumarde.com?
  - Enabled through what's called a TLS (or HTTPS) *certificate*
- **What is a certificate?**
  - Basically, it's an encoding of the server's **public key**, *signed* by a **trusted authority** called a Certificate Authority (CA)

Certificates and Certificate Authorities

- How do Alice **actually know** that she's talking to kumarde.com?
  - Enabled through what's called a TLS (or HTTPS) *certificate*
- **What is a certificate?**
  - Basically, it's an encoding of the server's **public key**, *signed* by a **trusted authority** called a Certificate Authority (CA)
- On the web, root CAs are *hardcoded into the browser* and signatures are verified by the browser
- On your computer, root CAs are *hardcoded into the OS* and signatures are verified in the kernel

What does a CA do?

- Basically one job: **validate the identity of an entity (website, individual, device) and issue digital certificates attesting to that identity.**
  - Certificates bind a *public key* to an *entity*, enabling authentication!

What does a CA do?

- Basically one job: **validate the identity of an entity (website, individual, device) and issue digital certificates attesting to that identity.**
  - Certificates bind a *public key* to an *entity*, enabling authentication!

Cert for kumarde.com please

What does a CA do?

- Basically one job: **validate the identity of an entity (website, individual, device) and issue digital certificates attesting to that identity.**
  - Certificates bind a *public key* to an *entity*, enabling authentication!

Cert for kumarde.com please

Please verify you own kumarde.com

Let's Encrypt

How does a CA verify you own a webpage?

- Used to be:
  - Call someone, fax them your ID / ownership details, pay them some amount of money… costly, and meant most people didn’t have HTTPS

How does a CA verify you own a webpage?

- Used to be:
  - Call someone, fax them your ID / ownership details, pay them some amount of money
- Nowadays:
  - **Domain ownership verification**

If you really own kumarde.com…

Add a TXT record w/ <random_val>

Add an HTTP page w/ <random_val>

https://letsencrypt.org/docs/challenge-types/

What does a CA do?

- Basically one job: **validate the identity of an entity (website, individual, device) and issue digital certificates attesting to that identity.**
  - Certificates bind a *public key* to an *entity*, enabling authentication!

Cert for kumarde.com please →

← Please verify you own kumarde.com

Here is my proof + public key →

Let's Encrypt

What does a CA do?

- Basically one job: **validate the identity of an entity (website, individual, device) and issue digital certificates attesting to that identity.**
  - Certificates bind a *public key* to an *entity*, enabling authentication!

Cert for kumarde.com please →

← Please verify you own kumarde.com

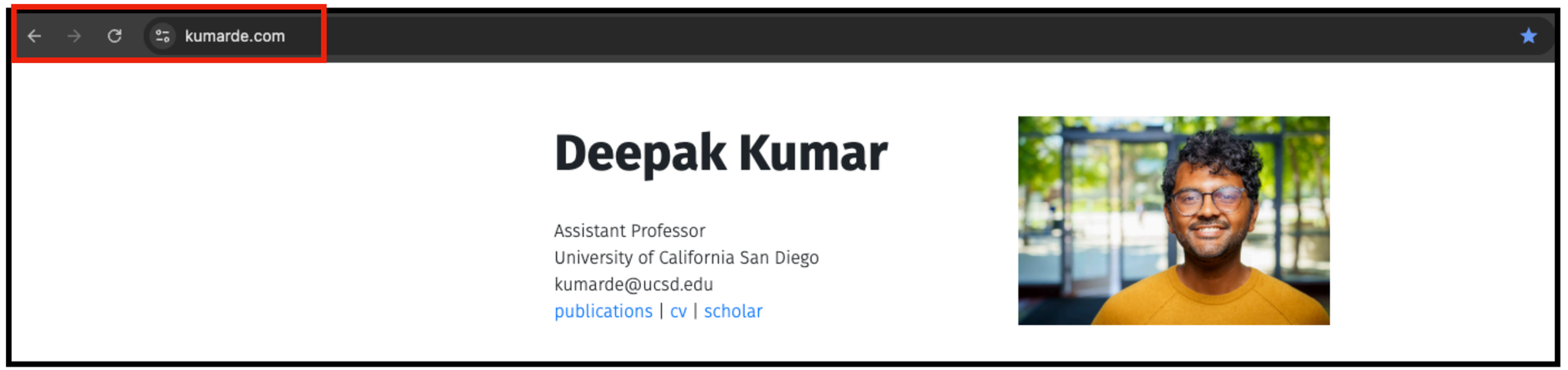
Here is my proof + public key →

← Confirmed, here's a cert **signed by us**

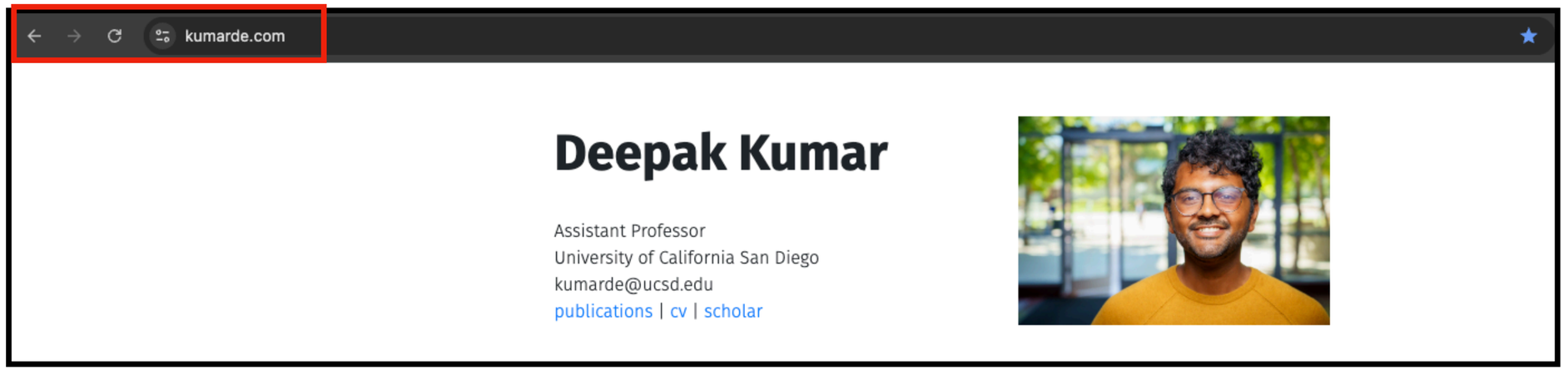
Let's Encrypt

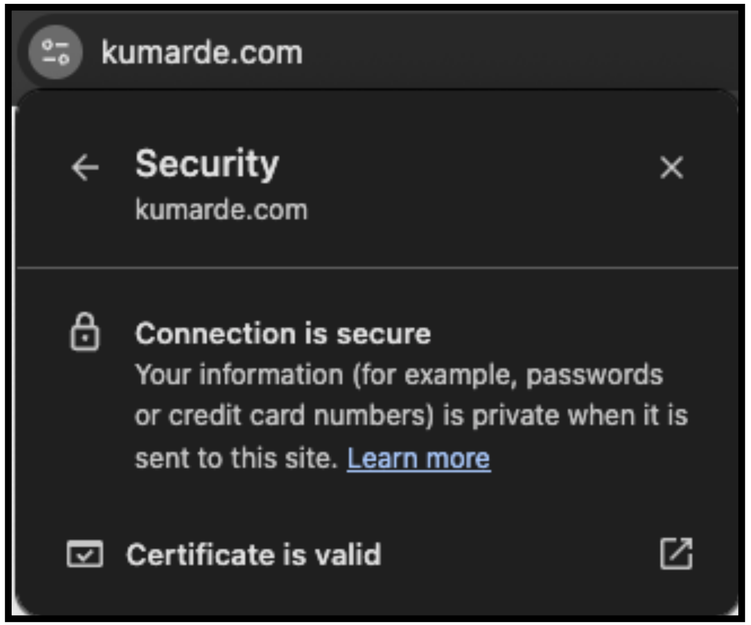
Nowadays, all this happens "automatically"

Certificate Deep Dive

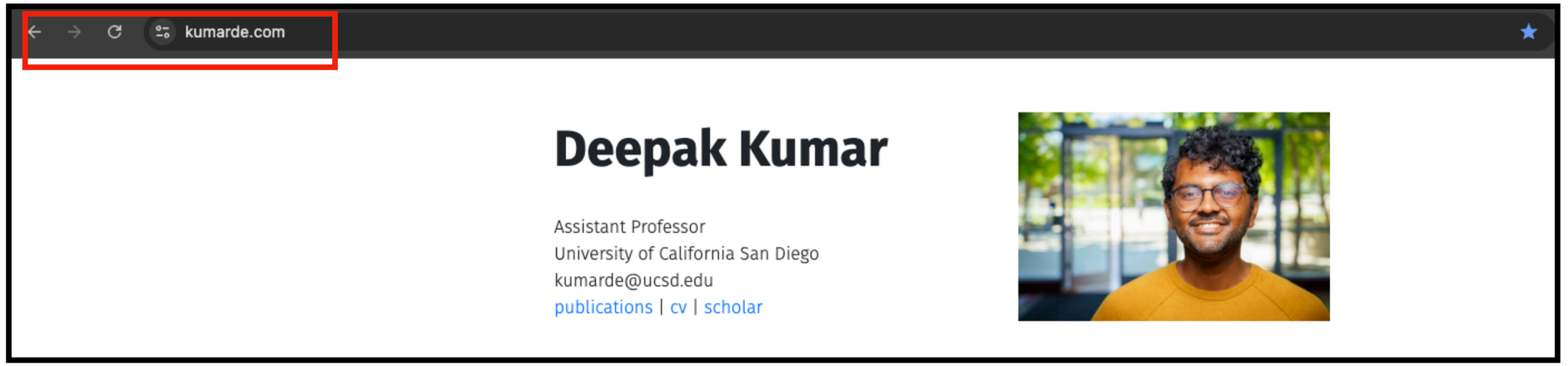


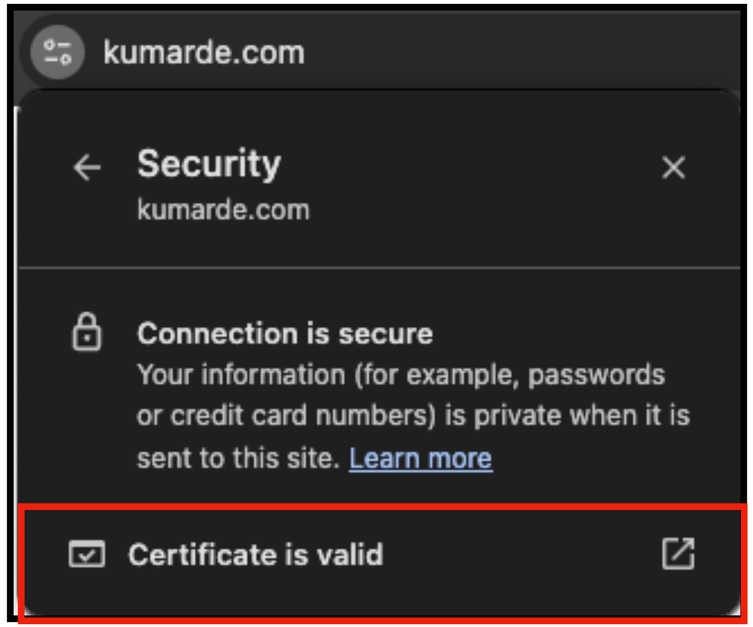
Certificate Deep Dive



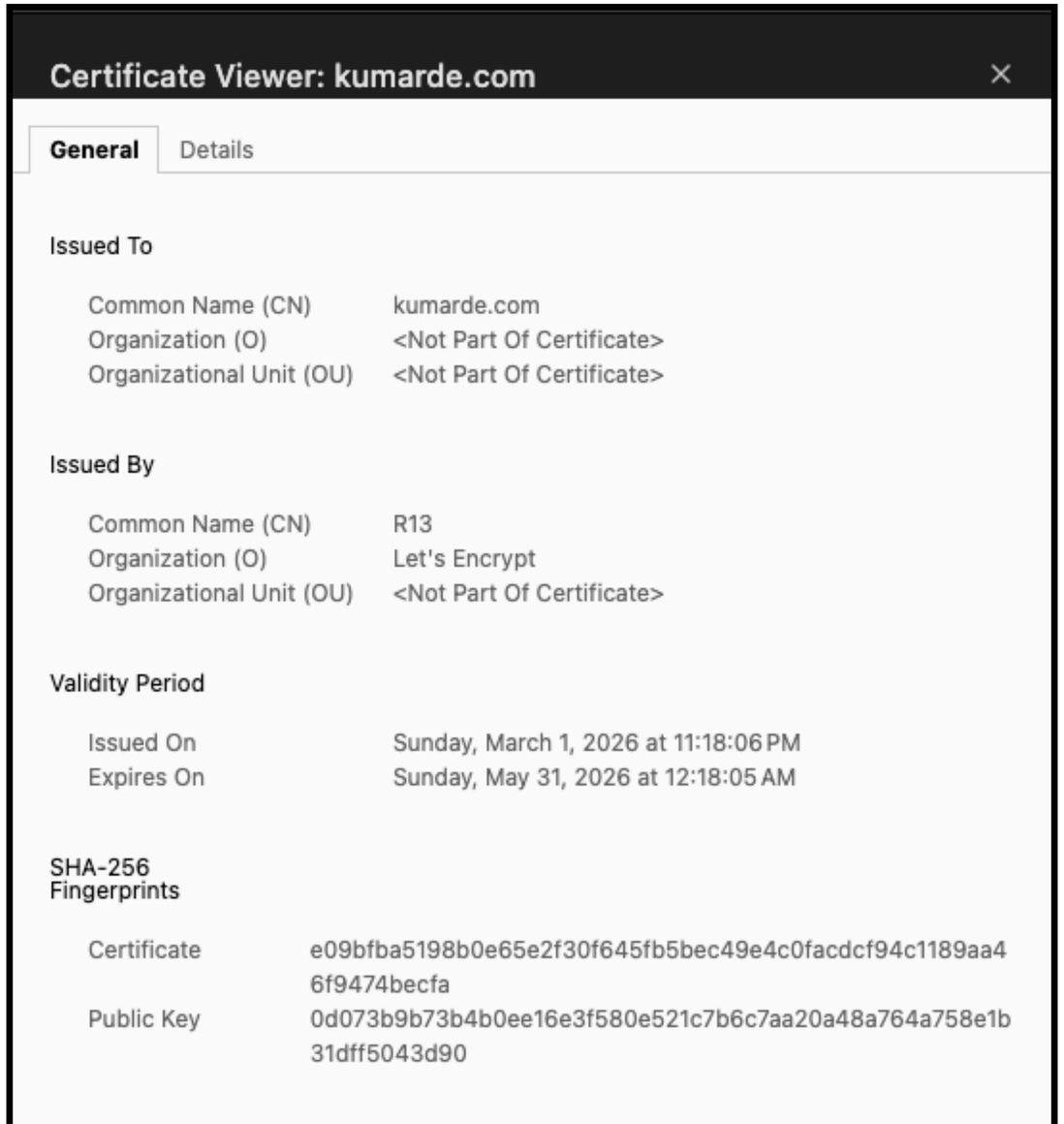


Certificate Deep Dive

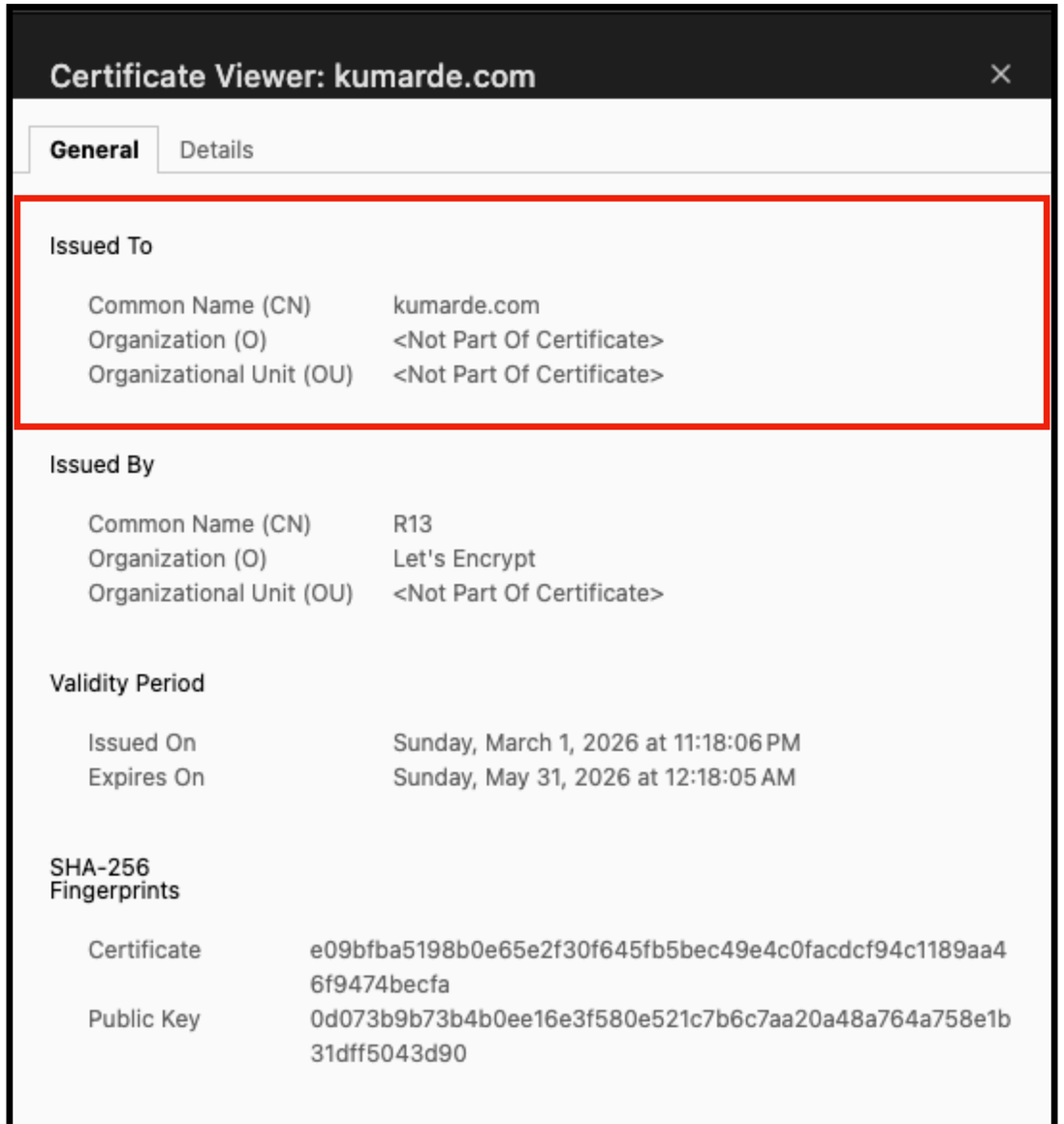




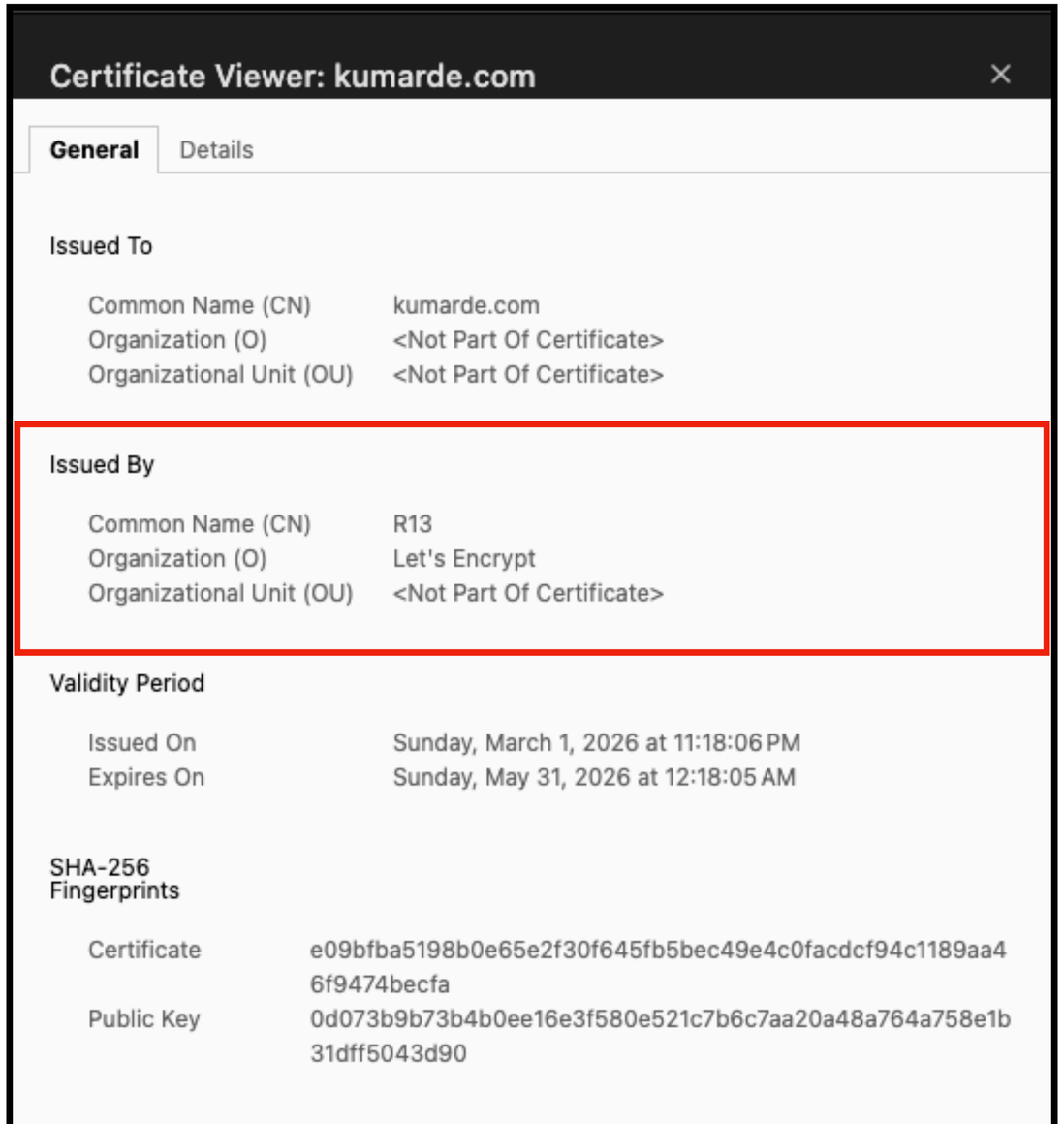
Certificate Deep Dive



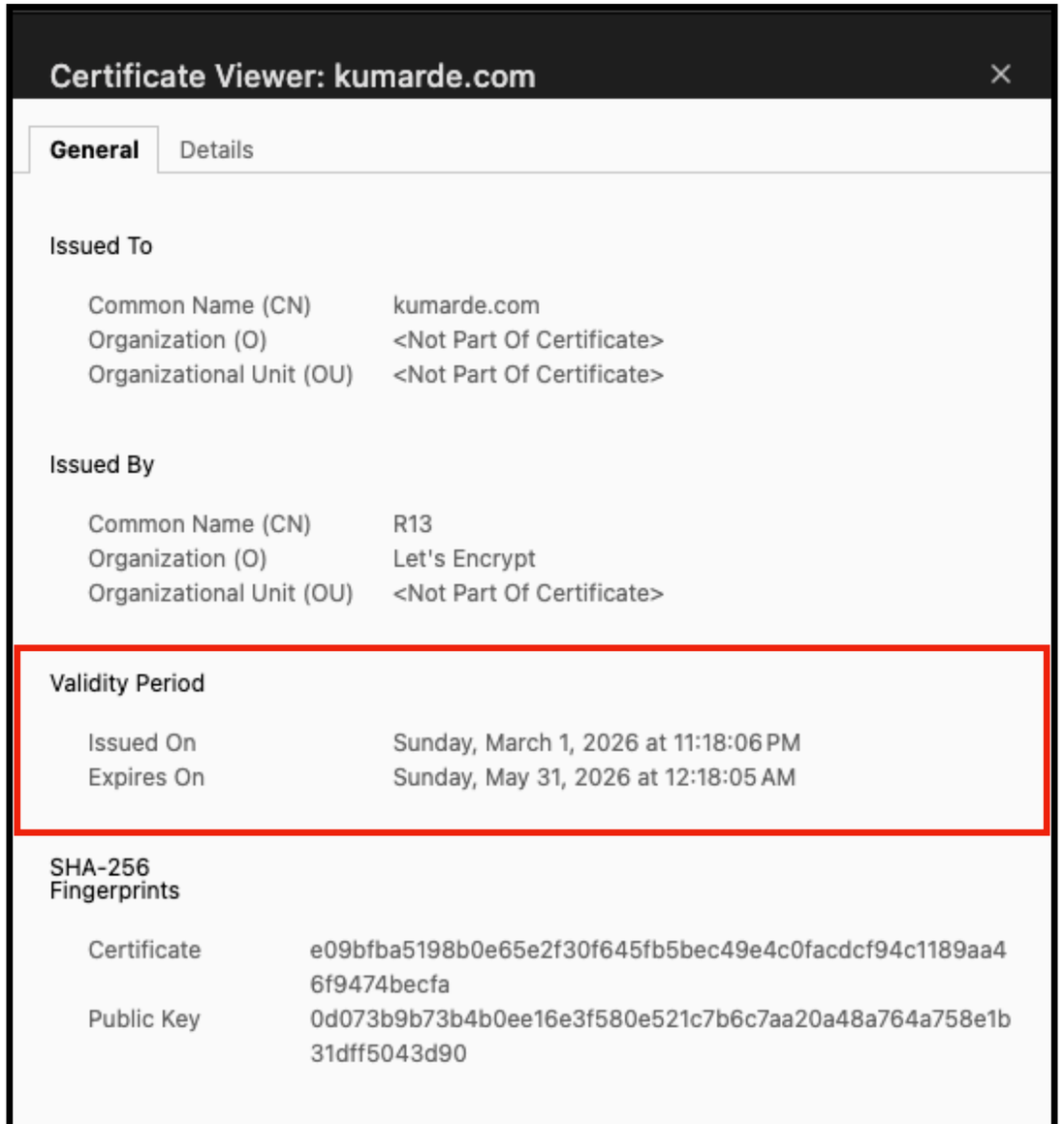
Certificate Deep Dive



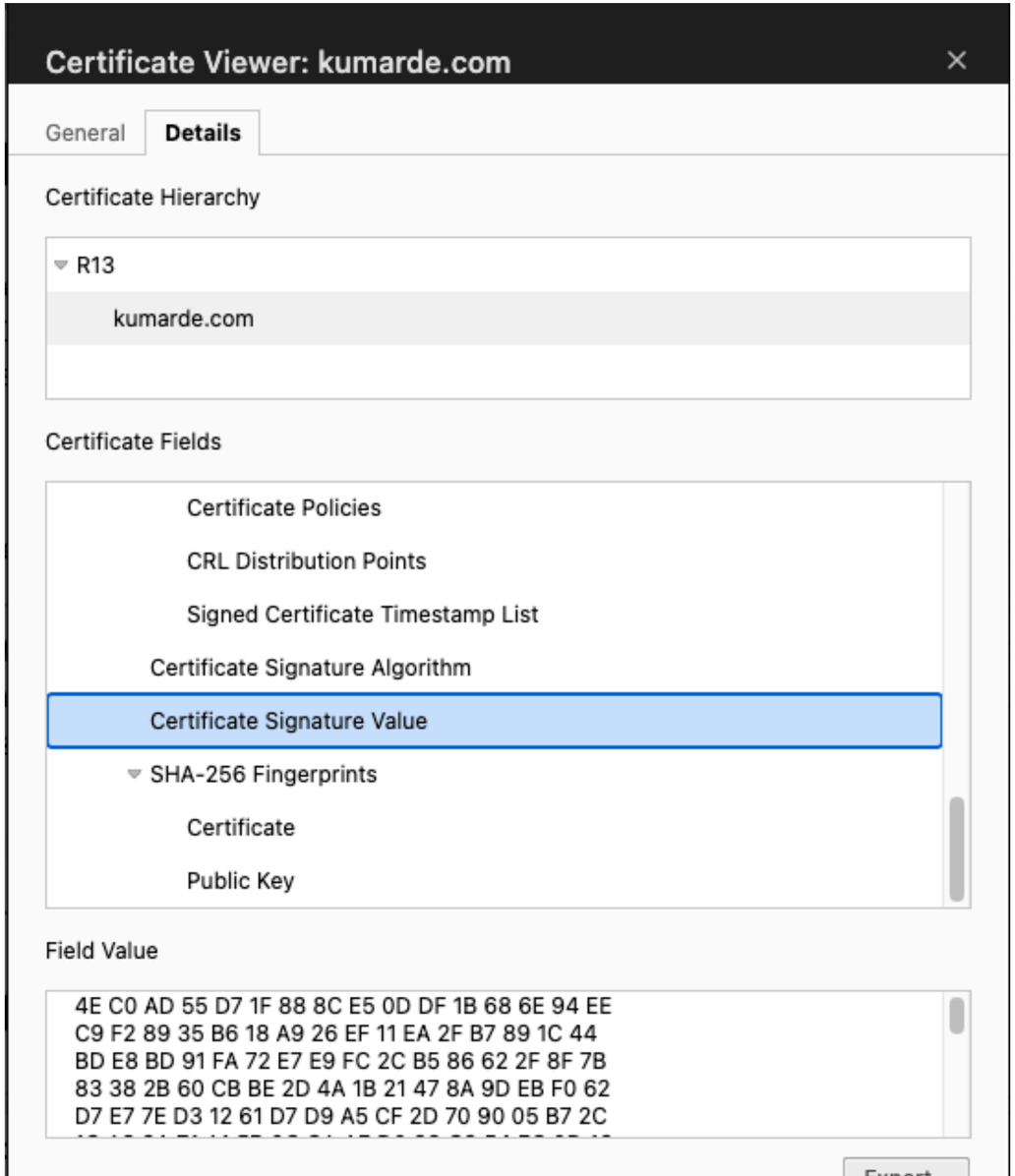
Certificate Deep Dive



Certificate Deep Dive



Certificate Deep Dive



OK, but how do we know which CAs to trust?

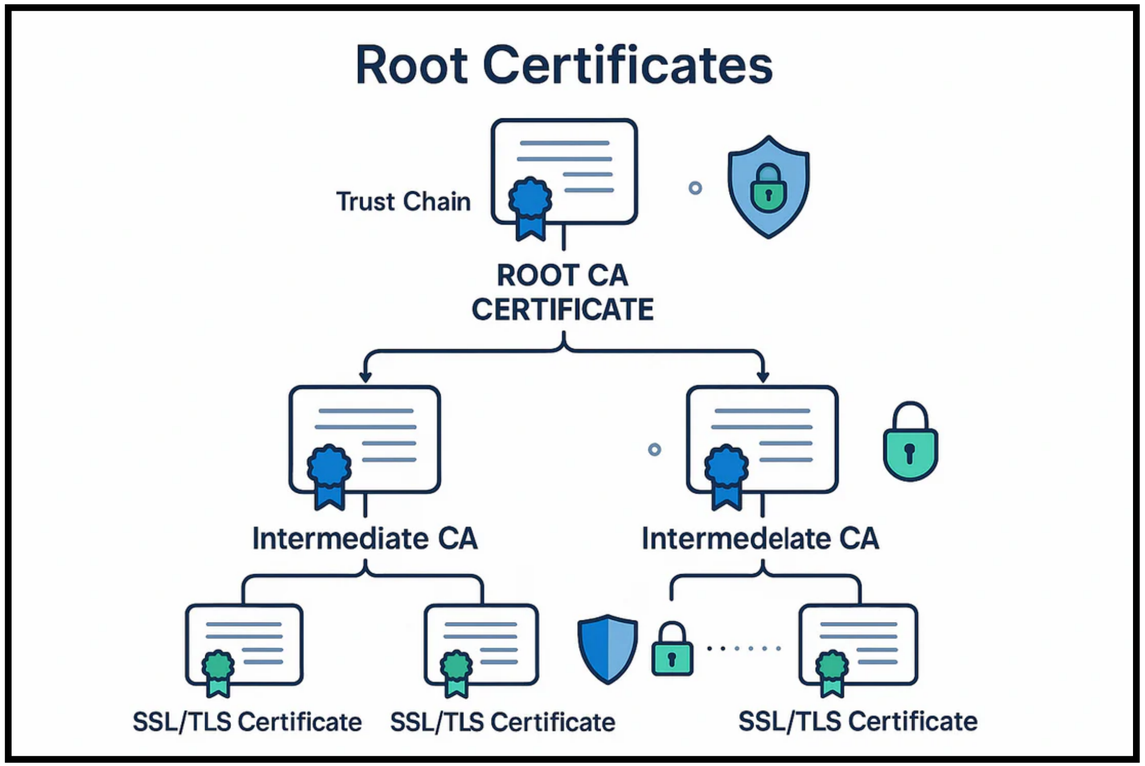
- CAs hold tremendous power: they are the arbiters of all identity and authenticity on the Internet
  - Who decides which CAs to trust? How do they get populated? Who are these CAs, even?
- Largely, CAs are decided by a coordination between **browsers, OS vendors,** and **CA vendors**
  - Open process of negotiation primarily between Google, Microsoft, Mozilla, and CA vendors
- There are **hundreds of CAs** — Let's Encrypt, Google, Microsoft, Digicert, Sectigo, GlobalSign, GoDaddy…. the list goes on

Certificate Authorities in Browsers

- Interestingly, each browser / OS has *slight* variations in who they trust…
- Mozilla
  - 181 root certificates
- Chrome
  - 99 root certificates
- iOS
  - 157 root certificates
- Microsoft
  - > 400 root certificates

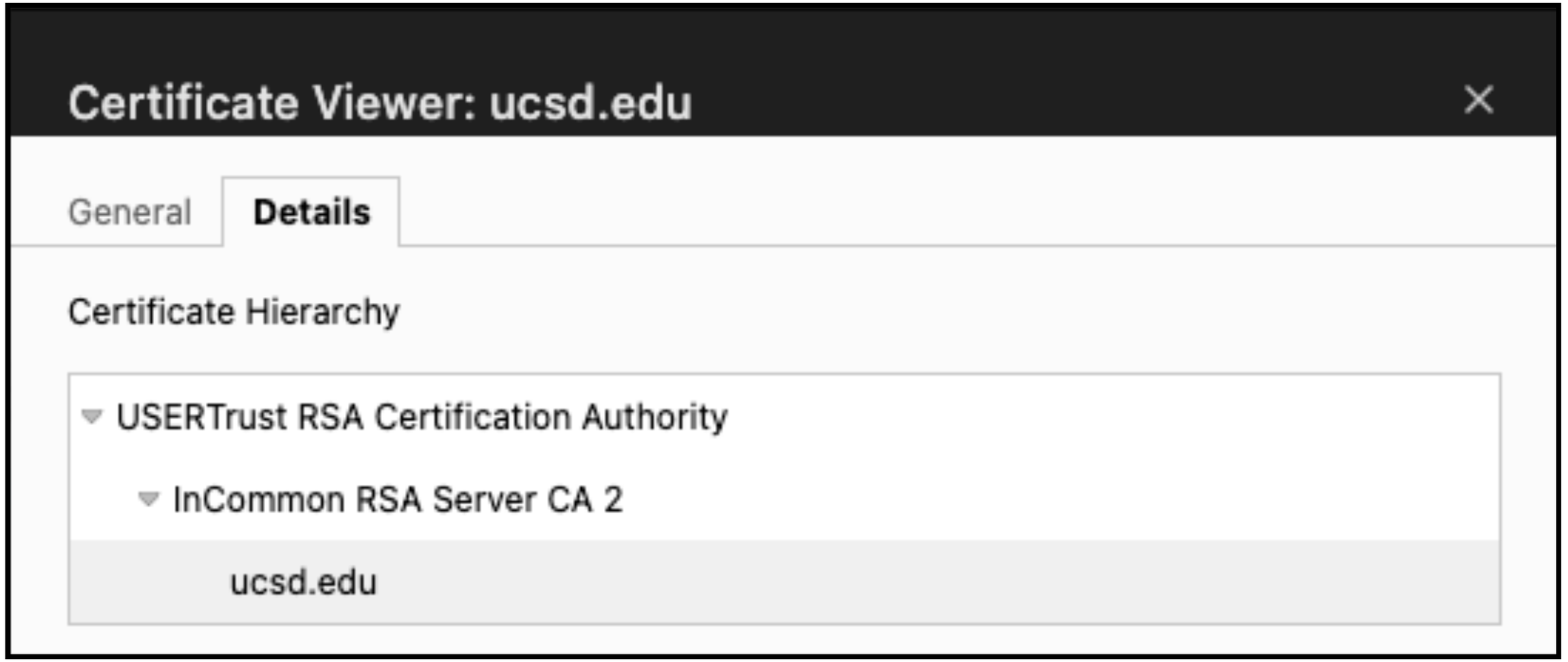
Verifying trust through a Chain-of-Trust

- **Root CAs** form what is called a *root of trust* — the foundational, inherently trusted entities that all trust goes back to (e.g., trust base)
  - **Root CAs** can delegate "signing authority" to what are called **intermediate CAs**
  - Mostly, this was a scalability thing… nowadays, enables **signing authority isolation**
- When verifying certificate, the client verifies that a certificate *chains back to the root CA in the root store*



https://shop.trustico.com/blogs/stories/about-root-certificates-in-ssl-tls

Verifying trust through a Chain-of-Trust

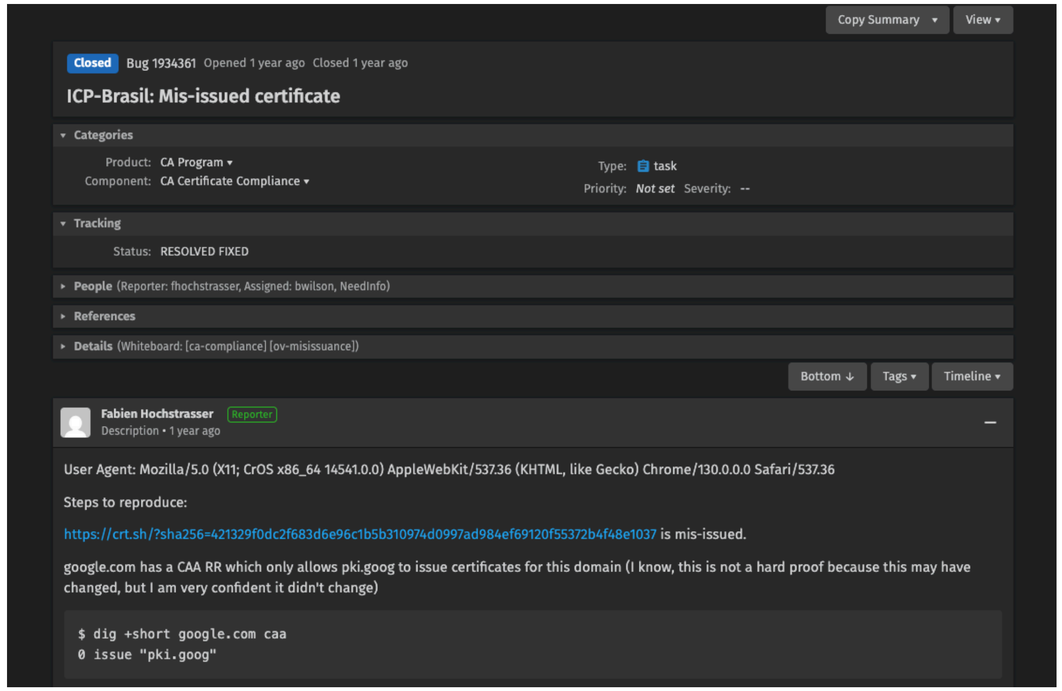


CAs have enormous power and responsibility

- Which CAs can sign for fbi.gov?

CAs have enormous power and responsibility

- Which CAs can sign for fbi.gov?
  - **Any CA that chains to a trusted root…**
- CAs occasionally “misissue” certificates for the wrong domains…. leading to potentially catastrophic outcomes
- Just last year, a trusted Brazilian CA misissued a valid certificate for google.com



Misissued CAs enable MiTM attacks

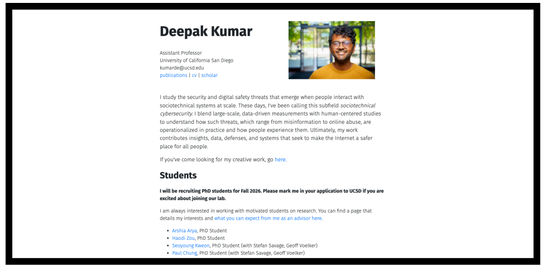
- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

Alice

bad kumarde.com
self signed



kumarde.com
cert issued by Let's Encrypt

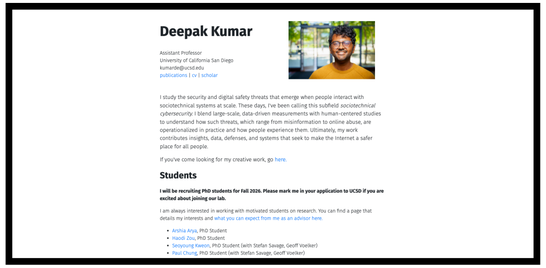
Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

185.199.111.153

Alice

bad kumarde.com
self signed



kumarde.com
cert issued by Let's Encrypt

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

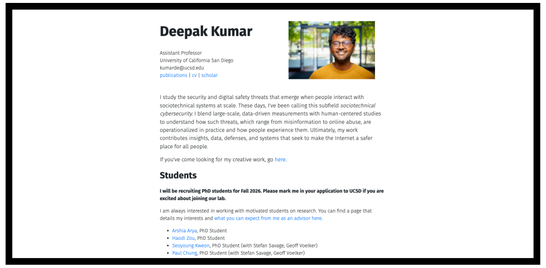
Alice

kumarde.com?

66.66.66.66

GoGo DNS

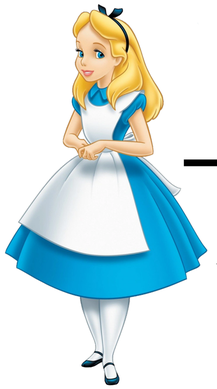
bad kumarde.com
self signed



kumarde.com
cert issued by Let's Encrypt

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

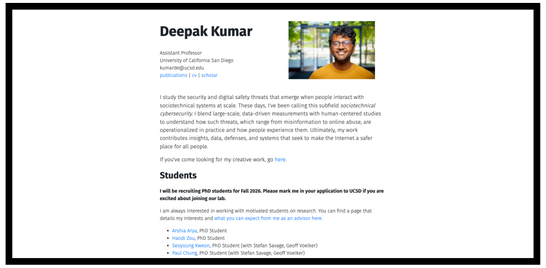


Alice

TLS client hello

certificate

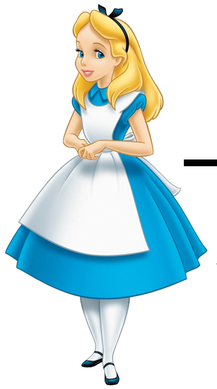
bad kumarde.com
self signed



kumarde.com
cert issued by Let's Encrypt

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

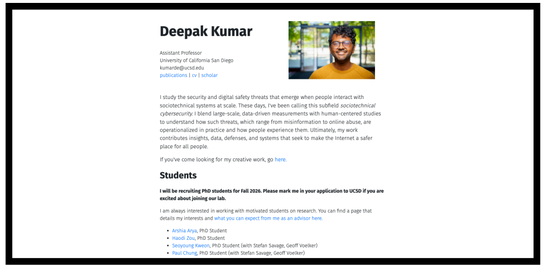


Alice

TLS client hello

certificate

bad kumarde.com
self signed

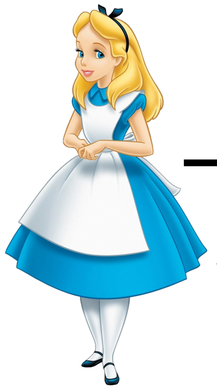


kumarde.com
cert issued by Let's Encrypt

Check will fail because browser doesn't trust self-signed!

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

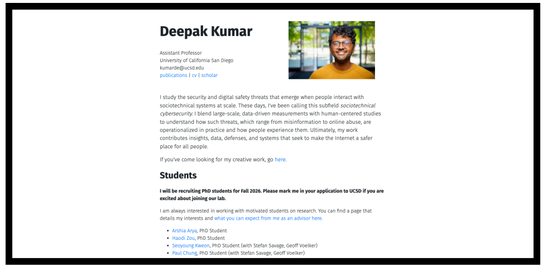


Alice

TLS client hello

certificate

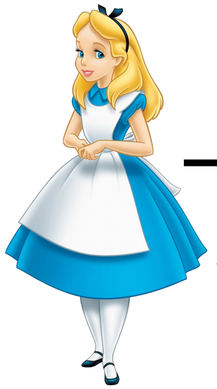
bad kumarde.com
signed by Gogo



kumarde.com
cert issued by Let's Encrypt

Misissued CAs enable MiTM attacks

- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**

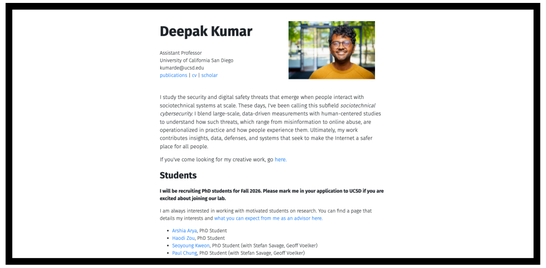


Alice

TLS client hello

certificate

bad kumarde.com
signed by Gogo

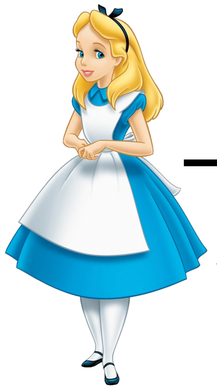


kumarde.com
cert issued by Let's Encrypt

Check will pass browser trusts GoGo!

Misissued CAs enable MiTM attacks

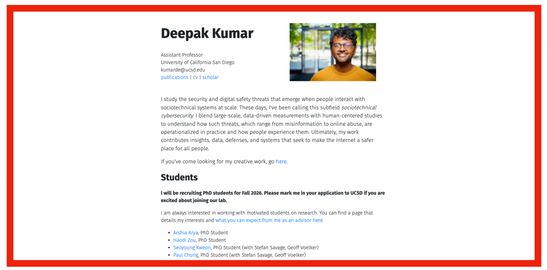
- If the certificate is misissued, **anyone with that certificate can impersonate the entity it is missisued for.**



Alice

TLS client hello

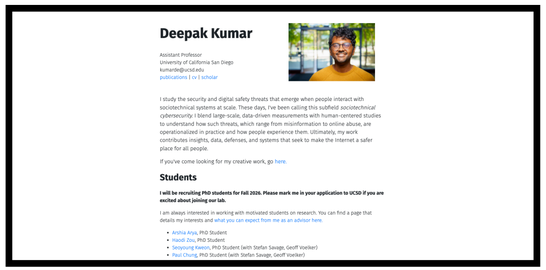
certificate



bad kumarde.com
signed by Gogo

hello

certificate



kumarde.com
cert issued by Let's Encrypt

- Misissued certificates **break authentication and trust** on the web… huge deal!

Misissued certificate measurements + fixing the ecosystem

Tracking Certificate Misissuance in the Wild

Deepak Kumar*, Zhengping Wang*, Matthew Hyder*, Joseph Dickinson*, Gabrielle Beck†,
David Adrian†, Joshua Mason*, Zakir Durumeric*†‡, J. Alex Halderman†, Michael Bailey*

* University of Illinois Urbana-Champaign † University of Michigan ‡ Stanford University

README Contributing Apache-2.0 license

ZLint

Go passing integration-test passing golangci-lint passing go report A+

ZLint is a X.509 certificate linter written in Go that checks for consistency with standards (e.g. RFC 5280) and other relevant PKI requirements (e.g. CA/Browser Forum Baseline Requirements).

It can be used as a command line tool or as a library integrated into CA software.

Requirements

ZLint requires Go 1.16.x or newer be installed. The command line setup instructions assume the `go` command is in your `$PATH`.

Zlint Users/Integrations

Pre-issuance linting is **strongly recommended** by the Mozilla root program. Here are some projects/CAs known to integrate with ZLint in some fashion:

- Actalis
- ANF AC
- Camerfirma
- CFSSL
- Digicert
- EJBCA
- Entrust
- Globalsign
- GoDaddy
- Google Trust Services
- Government of Spain, FNMT
- Izenpe
- Let's Encrypt and Boulder
- Microsec
- Microsoft
- Nexus Certificate Manager
- QuoVadis
- Sectigo, crt.sh, and pkimetal
- Siemens
- SSL.com
- PKI Insights
- NETLOCK
- Disig

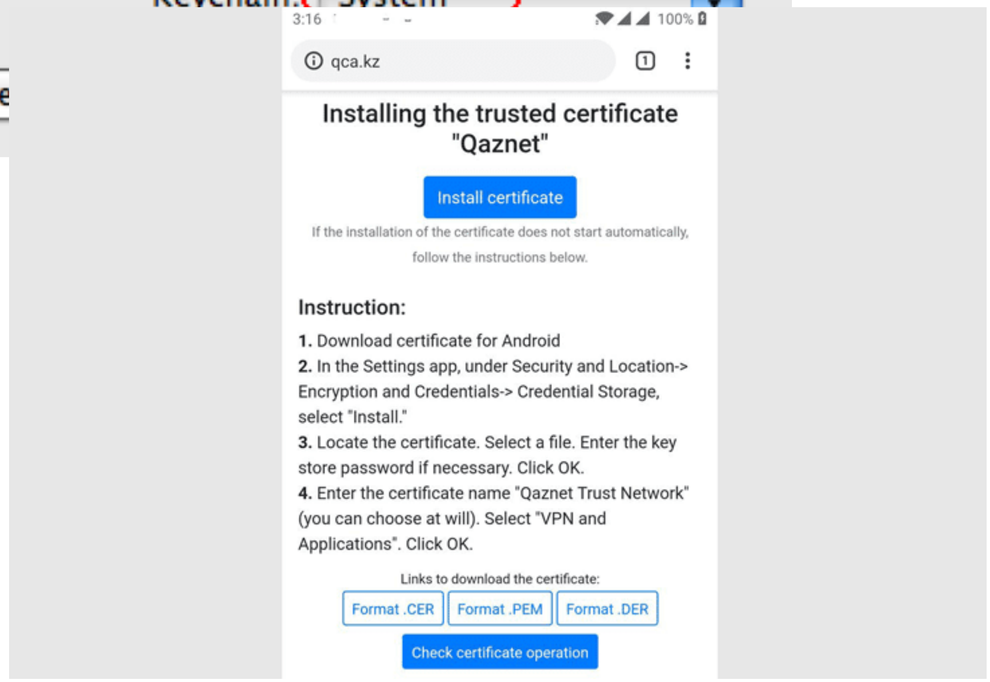
Certificate Revocation

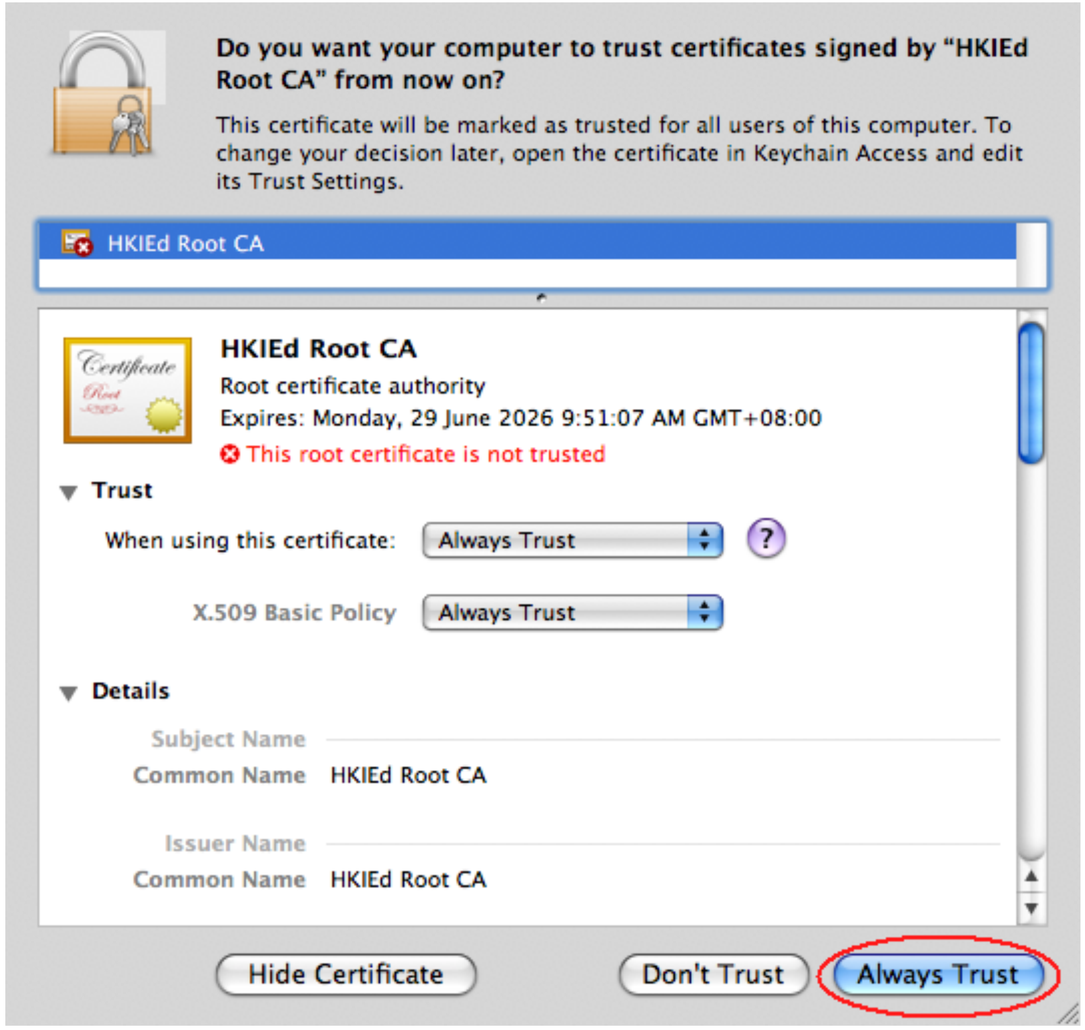
- So… let's say a certificate is misissued (or someone's private key is stolen. Or a CA goes rogue in other ways…) **What do we do?**
- **Certificates can be revoked:** basically, untrusted by the entirety of the ecosystem
- Two mechanisms for revocation: Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP)
  - CAs have an **obligation** to keep revocation systems online…. but they rarely do
  - Browsers tried many things to get them to do better, but nowadays, **revocation is largely unchecked online. <u>Why?</u>**

Failing Open vs. Failing Closed

- Browsers have an impossible choice w.r.t. revocation
  - Fail-open: If revocation check is not possible, just let the user connect anyway
    - Problem: Opens up users to MiTM attacks!
  - Fail-closed: If revocation check is not possible, close the connection
    - Problem: Erroneously early closes millions of connections on valid certs
- Mozilla, Google, all have different versions of how to do this…. CRLite, CRLSets… kind of a wild west atm

Installing root certs also opens you up to attacks…





TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 4: Server initiates DHKE

Alice

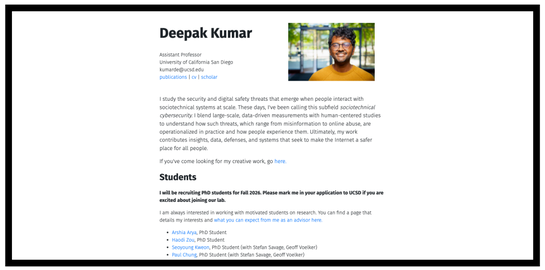
client hello: client random

[list of cipher suites]

server hello: server random, [cipher suite]

server certificate, verifying authenticity

server kex: p, g, g^a, Sign(p, g, g^a)



kumarde.com

https://tls12.xargs.org/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 5: Client continues DHKE

Alice

client hello: client random

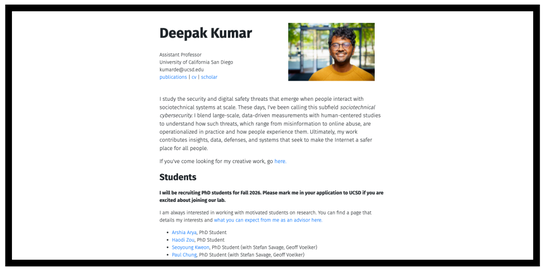
[list of cipher suites]

server hello: server random, [cipher suite]

server certificate, verifying authenticity

server kex: p, g, g^a, Sign(p, g, g^a)

client kex: g^b



kumarde.com

https://tls12.xargs.org/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 6: Client + Server compute keys

Alice

client hello: client random

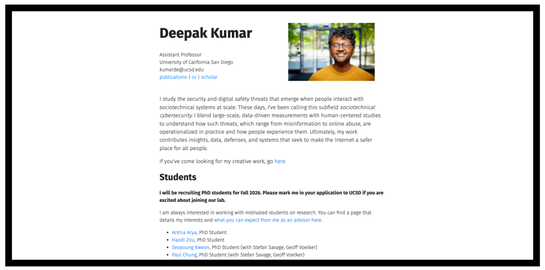
[list of cipher suites]

server hello: server random, [cipher suite]

server certificate, verifying authenticity

server kex: p, g, g^a, Sign(p, g, g^a)

client kex: g^b



kumarde.com

KGF(g^{ab}, random) —> encryption, signing keys

https://tls12.xargs.org/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 7: Verify integrity of conversation

Alice

client hello: client random

[list of cipher suites]

server hello: server random, [cipher suite]

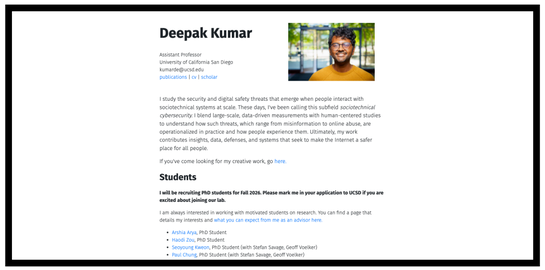
server certificate, verifying authenticity

server kex: p, g, g^a, Sign(p, g, g^a)

client kex: g^b

client finished: *MAC(dialog)*

server finished: *MAC(dialog)*



kumarde.com

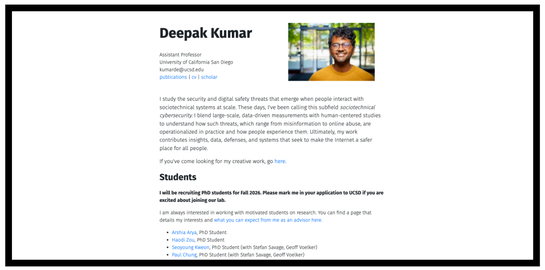
https://tls12.xargs.org/

TLS 1.2 w/ Diffie-Hellman Key Exchange (DHKE)

Step 8: Encrypt and send data!

Alice

Send encrypted application data!



kumarde.com

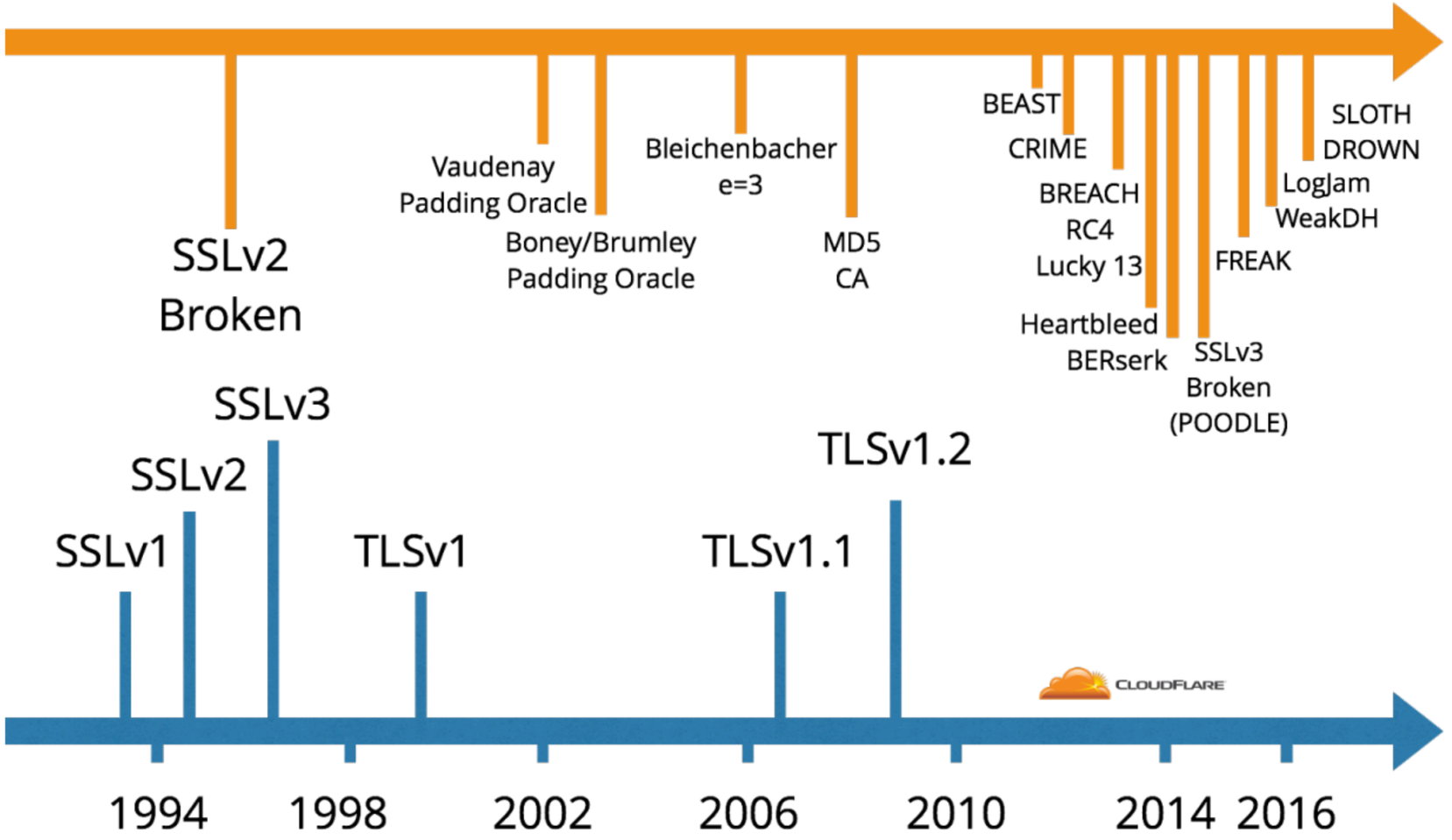
Decades of research into networks, cryptography, web, security…. now in your fingertips every single day!

https://tls12.xargs.org/

Some practical benefits of TLS

- Coffee shop WiFi
  - Even if attacker spoofs ARP, IP… your data is **still protected because of TLS level encryption**
- DNS cache poisoning
  - Even if attacker gets you to go to a different IP… your connection is **still protected because of TLS authentication guarantees**
- Offers high level protection against most network threats!

But, not without its problems!



TLS 1.3 is the new standard

- Developed over several years as a collaboration between cryptographers from industry and academia
- Some major differences from TLS 1.2:
  - Drops significantly vulnerable or newly vulnerable cipher suites
  - **Handshake encrypted immediately after key exchange**; certificate metadata is sent under encryption and limits passive eavesdropping
  - Protocol downgrade protection
    - Protects against protocol being downgraded to prior insecure version
  - …and much more, especially on performance side

TLS 1.3 is the new standard, long way for deployment

- Why?
  - HTTPS proxies super common in industry; they are all designed for TLS 1.2 and so they cannot upgrade until those boxes are upgraded
  - MiTM hardware is very common… e.g., how your network spies on you
  - Some hardware literally has TLS 1.2 etched inside of it and has become mission critical
- But overall, deployment is increasing, and we're all seeing the benefits

Check your understanding

- In groups, answer the following questions…
  - What can a passive eavesdropper learn from observing a TLS 1.2 handshake?
  - If an attacker obtains a TLS server's private key, what can they do?

Check your understanding

- In groups, answer the following questions…
  - What can a passive eavesdropper learn from observing a TLS 1.2 handshake?
    - Can see all negotiation details in plaintext, server certificate, supported cipher suites, and all TCP / IP information (e.g., metadata)
  - If an attacker obtains a TLS server's private key, what can they do?

Check your understanding

- In groups, answer the following questions…
  - What can a passive eavesdropper learn from observing a TLS 1.2 handshake?
    - Can see all negotiation details in plaintext, server certificate, supported cipher suites, and all TCP / IP information (e.g., metadata)
  - If an attacker obtains a TLS server's private key, what can they do?
    - Completely impersonate a TLS server. Keys are typically stored in HSM (hardware security modules) and kept under high security

Side note: other trust schemes exist too…

- SSH is another secure protocol (not TLS) that uses asymmetric cryptography for authentication
  - E.g., id_rsa, id_rsa.pub
- How does this work?
  - Strategy: Trust-on-first-use (TOFU)
- No trusted authorities, basically, once you connect to a server for the first time you "trust" it… any future connections with changes to configs will yell at you

```
sovereign:~ jharvard$ ssh login.rc.fas.harvard.edu
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: POSSIBLE DNS SPOOFING DETECTED!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for login.rc.fas.harvard.edu has changed,
and the key for the corresponding IP address 10.242.104.144
has a different value. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /Users/jharvard/.ssh/known_hosts:64
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
61:51:30:e0:e0:28:4d:df:df:65:79:ec:ec:ec:60:59.
Please contact your system administrator.
Add correct host key in /Users/jharvard/.ssh/known_hosts to get rid of this message.
Offending RSA key in /Users/jharvard/.ssh/known_hosts:8
RSA host key for login.rc.fas.harvard.edu has changed and you have requested strict checking.
Host key verification failed.
sovereign:~ jharvard$
```

https://docs.rc.fas.harvard.edu/kb/ssh-key-error/

In sum

- TLS is the most widely deployed cryptographic protocol in the world. It combines advances in networks, cryptography, and the web to secure **> 80% of all HTTP connections daily.**
- TLS offers confidentiality, integrity, *and* authenticity (via CAs), making it a one-stop shop for the security properties we care about the most
  - …but TLS **does not solve trust**
- **Our trust system on the Internet is entirely based on delegation to a small handful of players who claim to serve our best interests.**
  - This, IMO, is where a lot of the cryptography action is these days… big push towards *trustless systems*

Next time

- No class! Don’t come! I won’t be here!
- Good luck on PA5
- Final lecture will focus on sociotechnical security (my research area), and I’ll give you a flavor of some of the recent work to come out of my research group!
  - Will also impart some final words on applying security principles in your life
  - We’ll also spend ~10 mins for SETs, so please bring a laptop
- **Almost done!**