HSTS and HPKP in practice

These slides: https://goo.gl/tI6zOf

Research paper

Joseph Bonneau

(based on research w/Michael Kranch)

IETF 92

March 26 2015

HTTPS: where web-sec meets TLS

HTTP (≈ web browsing)

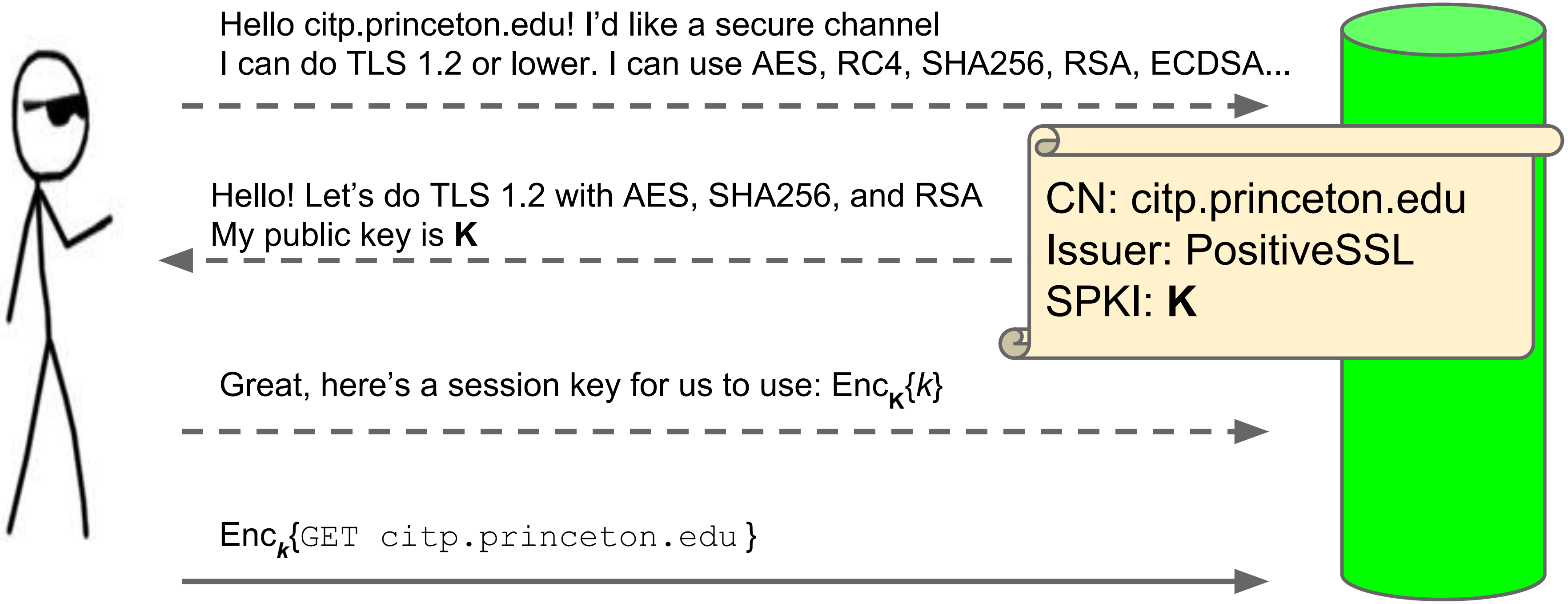
over

Secure **S**ockets **L**ayer (SSL)

or

Transport **L**ayer **S**ecurity

TLS in one slide



Hello citp.princeton.edu! I’d like a secure channel
I can do TLS 1.2 or lower. I can use AES, RC4, SHA256, RSA, ECDSA...

Hello! Let’s do TLS 1.2 with AES, SHA256, and RSA
My public key is **K**

CN: citp.princeton.edu
Issuer: PositiveSSL
SPKI: **K**

Great, here’s a session key for us to use: $\text{Enc}_{\mathbf{K}}\{k\}$

$\text{Enc}_{k}\{\texttt{GET citp.princeton.edu}\}$

Cryptographic flaws in TLS

- RSA timing leaks
- CBC padding oracle attacks
  - BEAST attack
- Compression leaks
  - CRIME attack
  - Lucky 13 attack
- RC4 statistical leakage
- Downgrade to SSL v3
- Session resumption attacks

See Clark & van Oorschot [IEEE SP '13]

The goal of HTTPS is a padlock

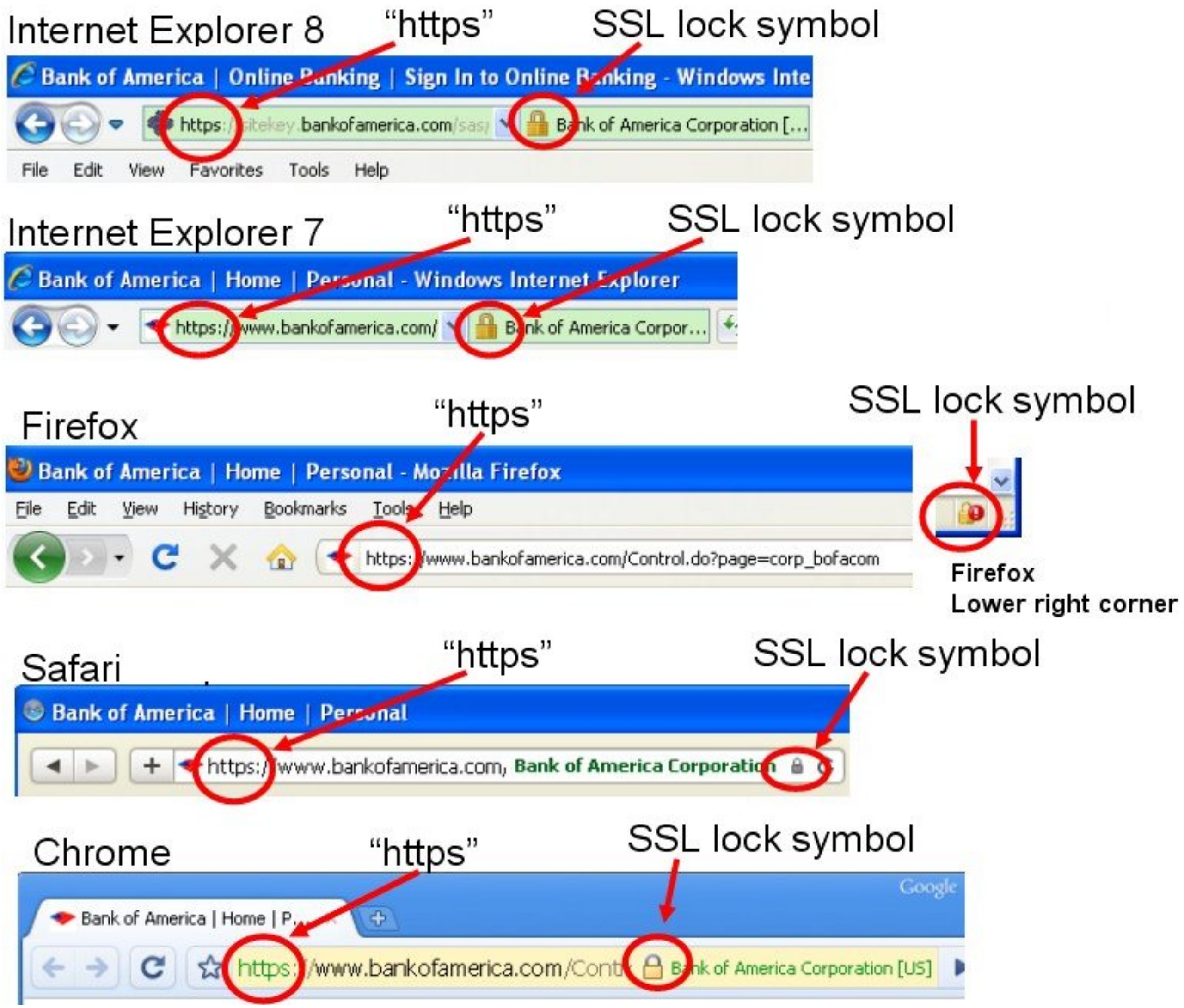

Internet Explorer 8
"https"
SSL lock symbol
Bank of America | Online Banking | Sign In to Online Banking - Windows Inte
https://sitekey.bankofamerica.com/sas/
Bank of America Corporation [...
File Edit View Favorites Tools Help
Internet Explorer 7
"https"
SSL lock symbol
Bank of America | Home | Personal - Windows Internet Explorer
https://www.bankofamerica.com/
Bank of America Corpor...
Firefox
"https"
SSL lock symbol
Bank of America | Home | Personal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://www.bankofamerica.com/Control.do?page=corp_bofacom
Firefox
Lower right corner
Safari
"https"
SSL lock symbol
Bank of America | Home | Personal
https://www.bankofamerica.com, Bank of America Corporation
Chrome
"https"
SSL lock symbol
Google
Bank of America | Home | P...
https://www.bankofamerica.com/Cont
Bank of America Corporation [US]

Image credit:
Will Bradley

HTTPS attacks in practice

- Inconsistent and incomplete deployment
  - *stripping attacks*
- Failures by Certificate Authorities
  - *rogue certificates*

HTTPS-level

- Lack of forward secrecy
  - *Subpoena of private keys*
  - *Compromise of keys*

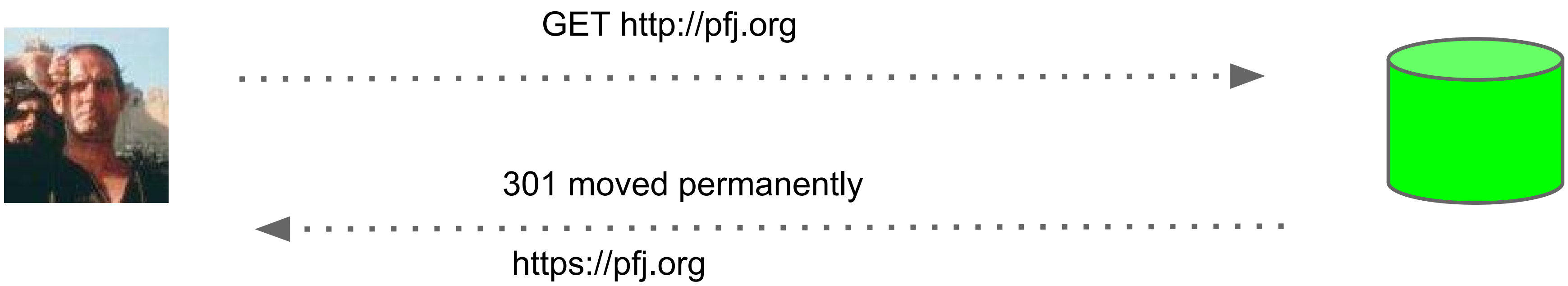
TLS-level

This talk will survey HSTS & pinning

- Overview of 2 big problems & solutions
  - HTTPS stripping, strict transport security
  - Rogue certificates, pinning
- Deployment overview
- Bugs!
  - Poorly configured HSTS
  - Mixed-content issues
  - Cookie leaking
  - Insecure links
- Design lessons

Problem 1: HTTPS stripping

HTTPS stripping

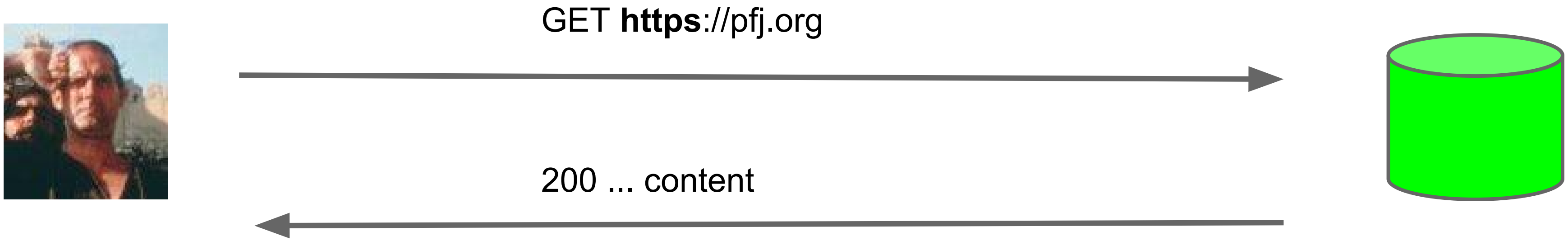


GET http://pfj.org

301 moved permanently

https://pfj.org

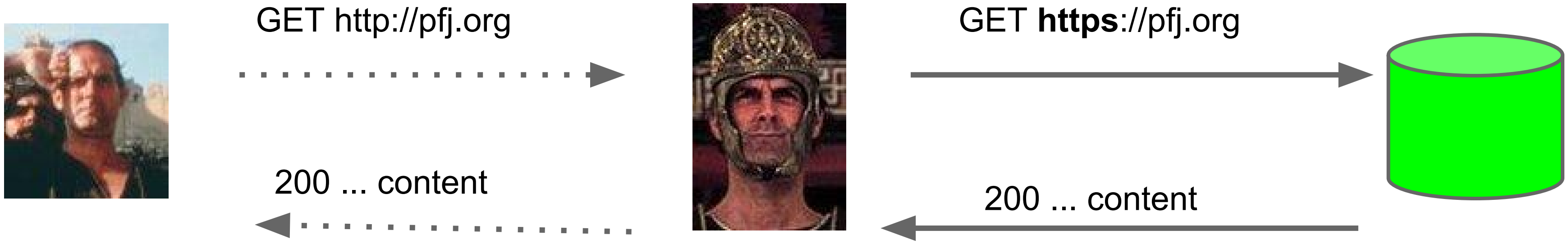
HTTPS stripping



GET **https**://pfj.org

200 ... content

HTTPS stripping



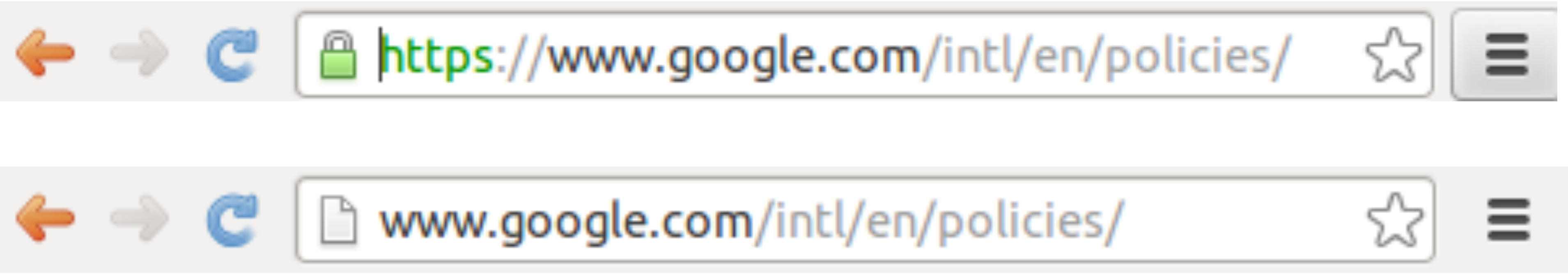
GET http://pfj.org

GET **https**://pfj.org

200 ... content

200 ... content

Will users detect HTTPS stripping?



https://www.google.com/intl/en/policies/

www.google.com/intl/en/policies/

<10% notice [Schechter et al. 2007] and others

Solution #1: HSTS (Strict Transport Security)

- *Mandatory* HTTPS at "HSTS domains"
  - Also: convert soft errors into hard errors
- **preloaded** by browsers
- **continuity** (*explicit*) via HTTP headers
- **introduction** via HTTPS links

HSTS Preload

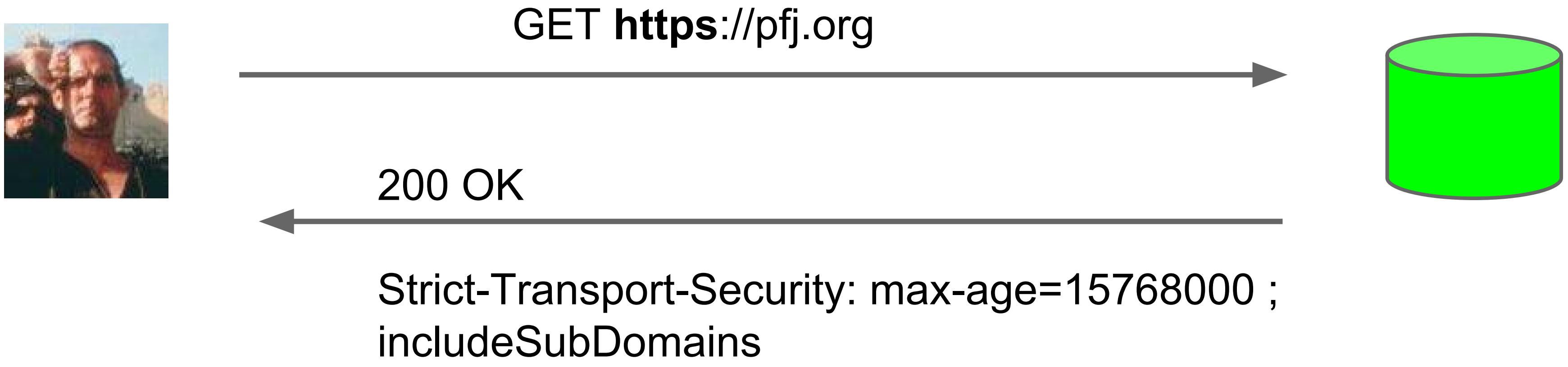
```
{ "name": "www.paypal.com", "mode": "force-https" },
{ "name": "www.elanex.biz", "mode": "force-https" },
{ "name": "jottit.com", "include_subdomains": true,
"mode": "force-https" },
{ "name": "sunshinepress.org", "include_subdomains":
true, "mode": "force-https" },
{ "name": "www.noisebridge.net", "mode": "force-https" },
...
```

transport_security_static.json (Chromium project)

Want more?

HTTPS Everywhere

Continuity: HSTS headers

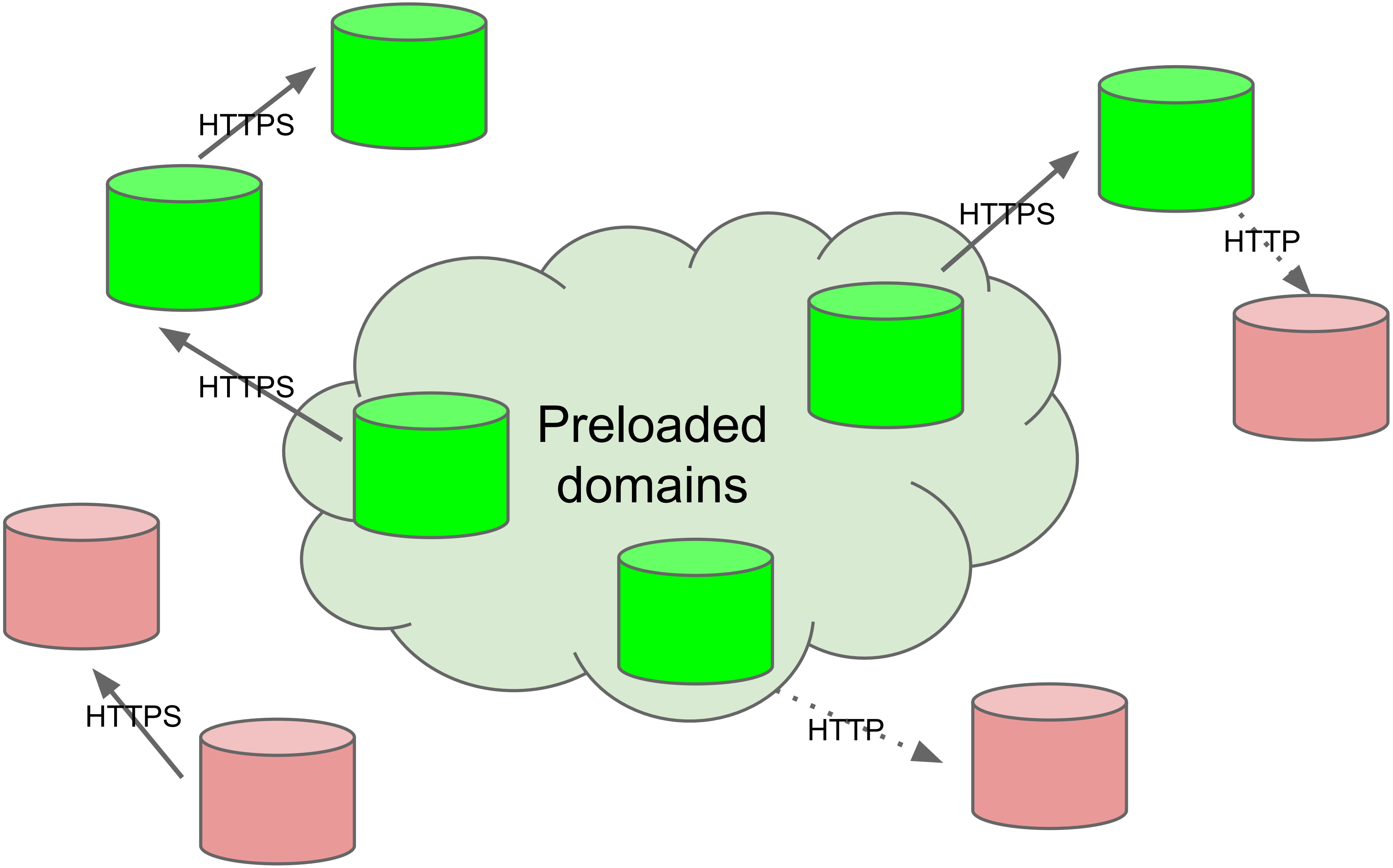


GET **https**://pfj.org

200 OK

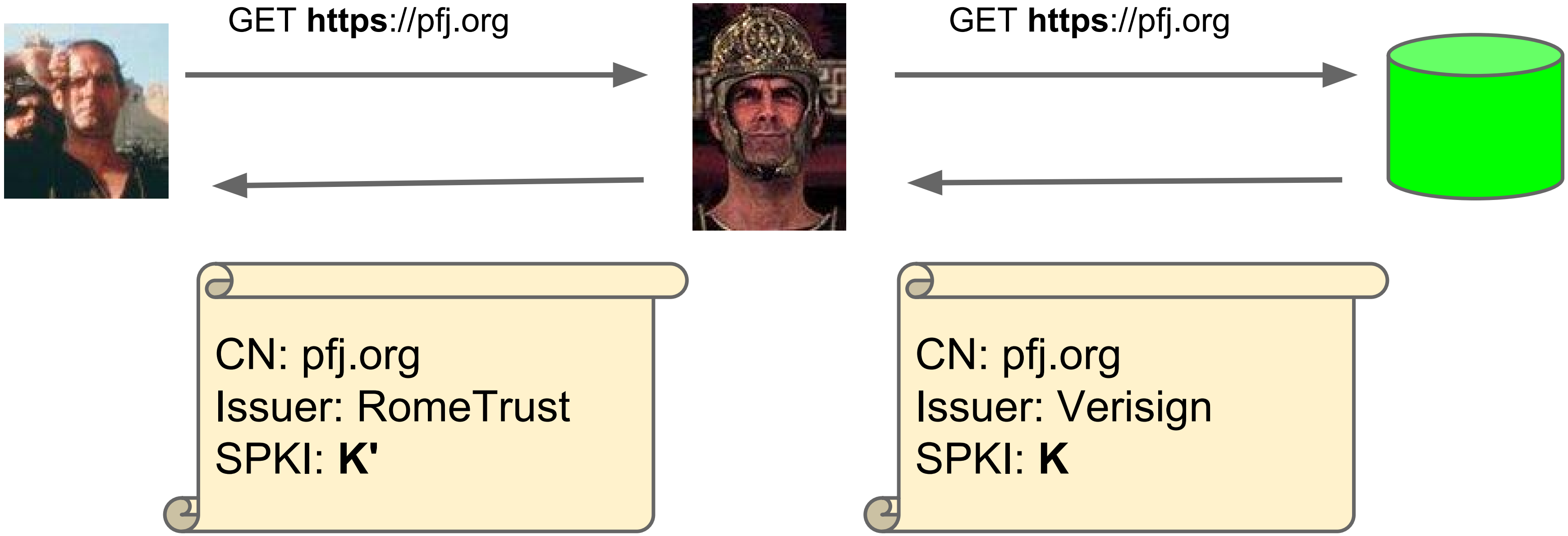
Strict-Transport-Security: max-age=15768000 ;
includeSubDomains

End-to-end HSTS security



Problem 2: Rogue certificates

Rogue certificates



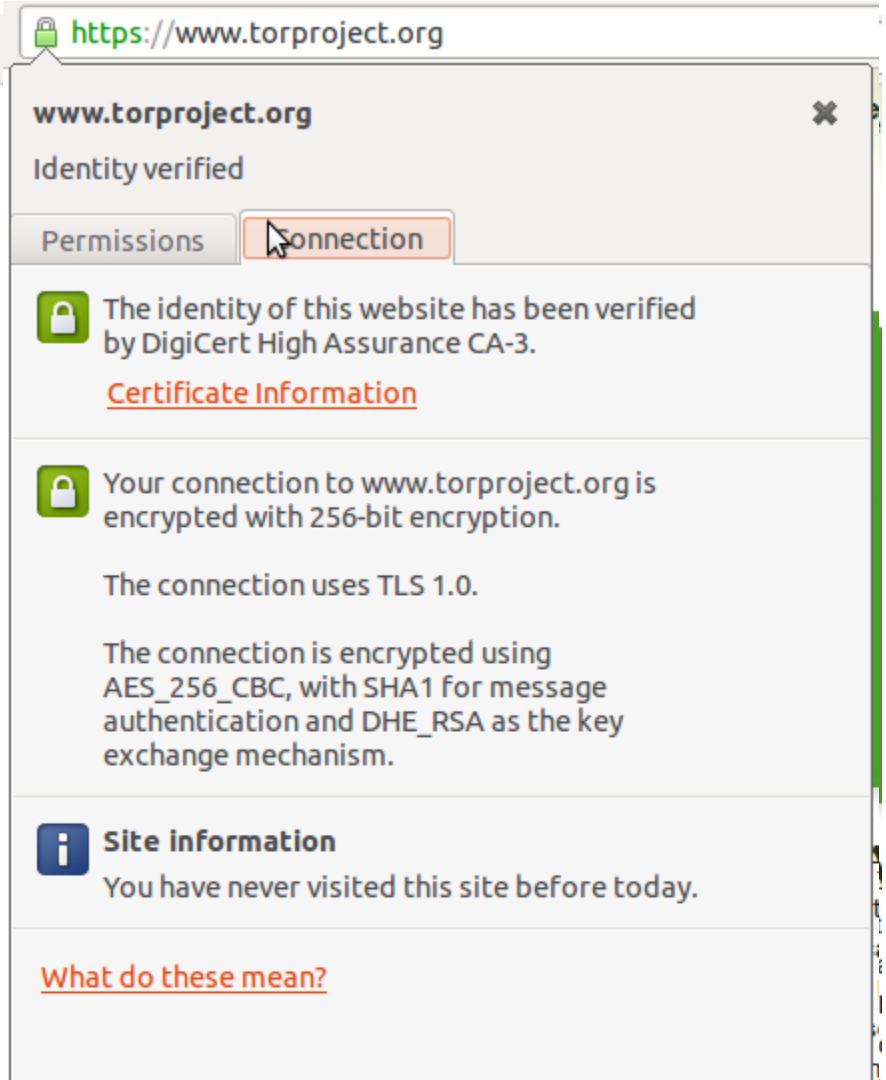
GET **https**://pfj.org

GET **https**://pfj.org

CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

CN: pfj.org
Issuer: Verisign
SPKI: **K**

Will users detect a rogue certificate?



https://www.torproject.org

www.torproject.org

Identity verified

The identity of this website has been verified by DigiCert High Assurance CA-3.

Certificate Information

Your connection to www.torproject.org is encrypted with 256-bit encryption.

The connection uses TLS 1.0.

The connection is encrypted using AES_256_CBC, with SHA1 for message authentication and DHE_RSA as the key exchange mechanism.

Site information

You have never visited this site before today.

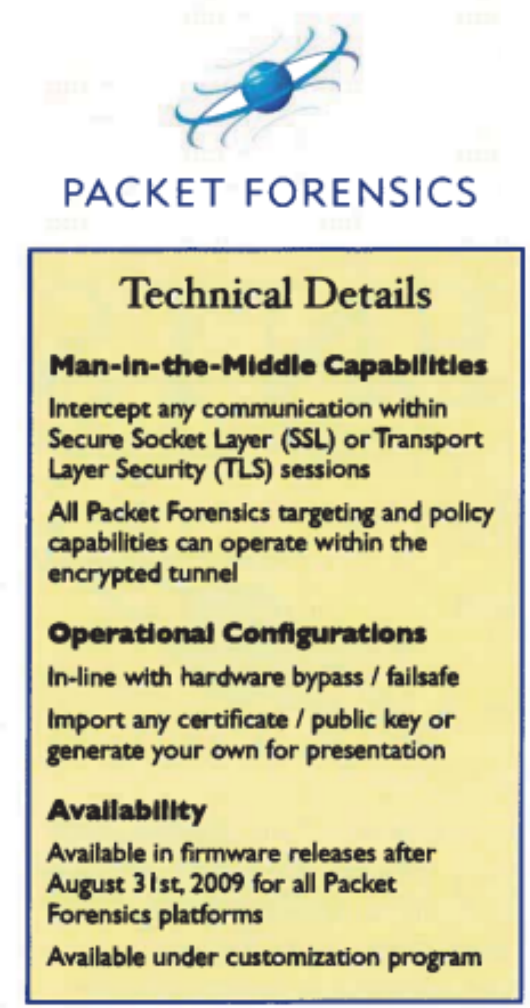
What do these mean?

Rogue certificates in the wild

- March 2011: Comodo registrar hacked
    - 9 certs: mail.google.com, login.live.com, www.google.com, login.yahoo.com, login.skype.com, addons.mozilla.org
- July 2011: DigiNotar hacked
    - 531+ certs issued: *.google.com detected first
- ~2011: TürkTrust issues 2 intermediate CAs
    - One returned, one used in 2012 to proxy traffic...

Survey: Niemann, Brendel 2014

Compelled certificates



PACKET FORENSICS

Technical Details

Man-in-the-Middle Capabilities

Intercept any communication within Secure Socket Layer (SSL) or Transport Layer Security (TLS) sessions

All Packet Forensics targeting and policy capabilities can operate within the encrypted tunnel

Operational Configurations

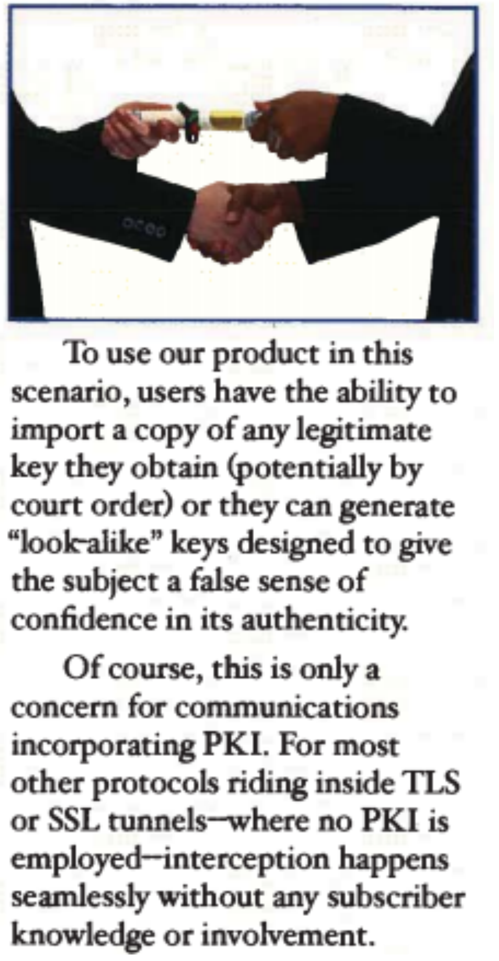
In-line with hardware bypass / failsafe

Import any certificate / public key or generate your own for presentation

Availability

Available in firmware releases after August 31st, 2009 for all Packet Forensics platforms

Available under customization program



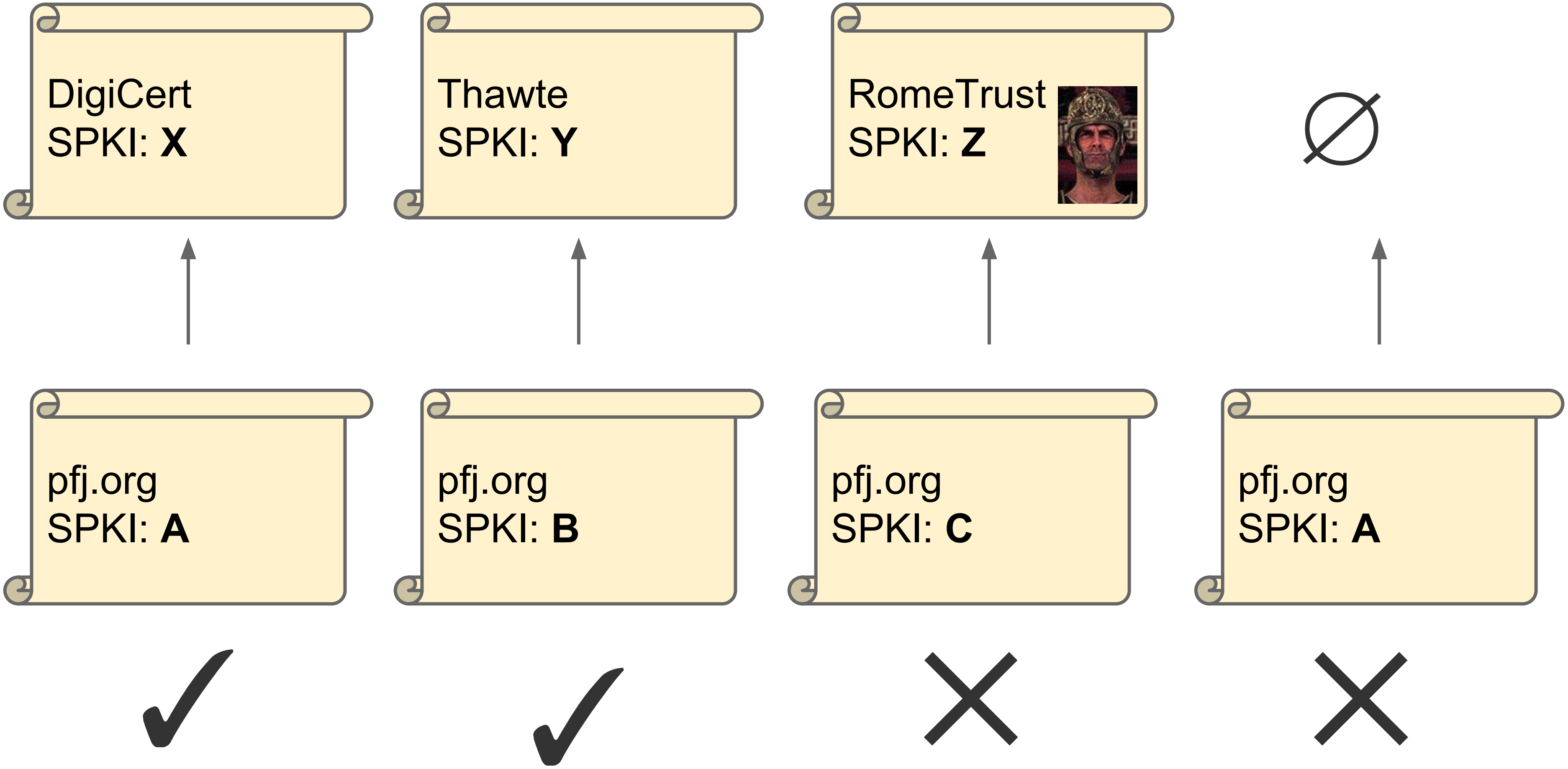
To use our product in this scenario, users have the ability to import a copy of any legitimate key they obtain (potentially by court order) or they can generate "look-alike" keys designed to give the subject a false sense of confidence in its authenticity.

Of course, this is only a concern for communications incorporating PKI. For most other protocols riding inside TLS or SSL tunnels—where no PKI is employed—interception happens seamlessly without any subscriber knowledge or involvement.

Soghoian, Stamm 2010

Solution #2: Key pinning

Pinset: {A, Y}



DigiCert
SPKI: **X**

pfj.org
SPKI: **A**

✓

Thawte
SPKI: **Y**

pfj.org
SPKI: **B**

✓

RomeTrust
SPKI: **Z**

pfj.org
SPKI: **C**

✗

∅

pfj.org
SPKI: **A**

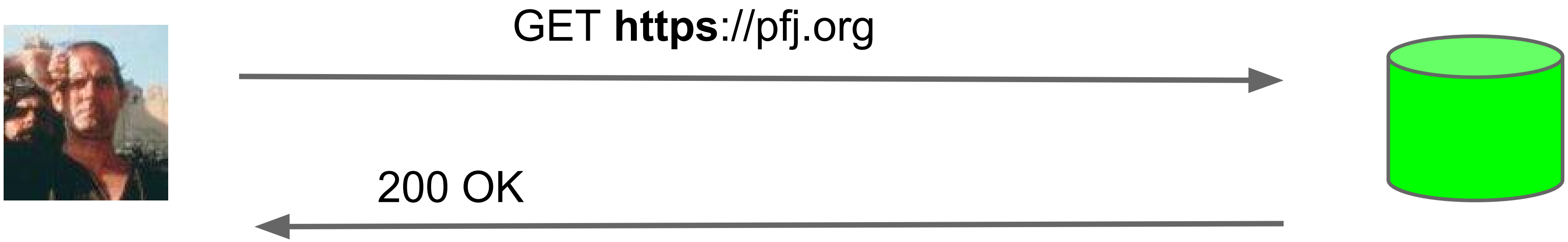
✗

Preloads: HPKP

```
{
  "pinsets": [
    {
      "name": "tor",
        "static_spki_hashes": [
          "RapidSSL",
          "DigiCertEVRoot",
          "Tor1",
          "Tor2",
          "Tor3"
      ]
    },
...
{ "name": "torproject.org", "mode": "force-https",
"pins": "tor" },
```

transport_security_static.json (Chromium project)

Continuity (explicit): HPKP headers



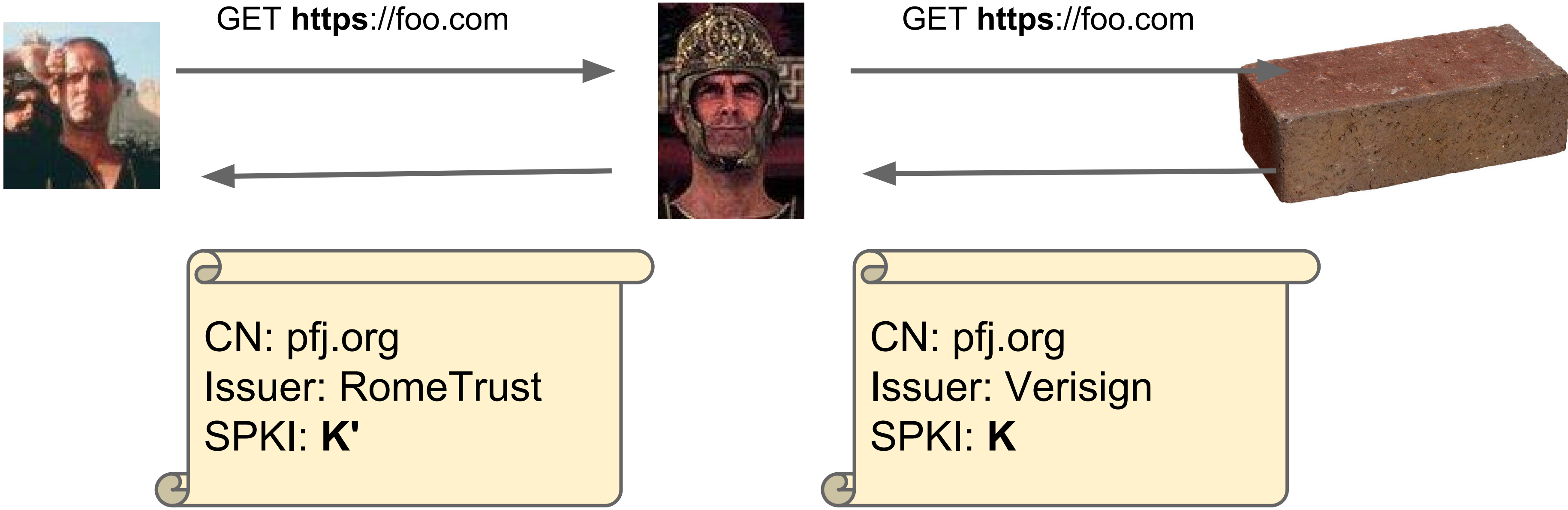
GET **https**://pfj.org

200 OK

Strict-Transport-Security: max-age=15768000 ; includeSubDomains

Public-Key-Pins: max-age=15768000; pin-sha1="4n972...baXc="; pin-sha256="LPJN... LmCQ="

Initial connections in HPKP



GET **https**://foo.com

GET **https**://foo.com

CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

CN: pfj.org
Issuer: Verisign
SPKI: **K**

Public-Key-Pins: max-age=15768000;
pin-sha1=H(**K'**); pin-sha256=H(X)

Public-Key-Pins: max-age=15768000;
pin-sha1=H(**K**); pin-sha256=H(X)

Current deployment

HSTS deployment so far

- proposed 2008 [Jackson/Barth W2SP paper]
- RFC 6797 standardized 2012
- support in Chrome, FF, Opera, Safari
  - No support in Internet Explorer ☹

As of November 2014:

- ~12,500 domains setting or trying HSTS
- 80% setting long-term HSTS

HPKP (aka PKP, web pinning)

- Evans, Palmer, Sleevi 2011
  - Proposed Standard, IETF Web Security working group
- Remaining issues
  - Domain bricking
  - Report-only mode
- ~20 early adopters!
  - No browser support

Growth of preloads in Chrome

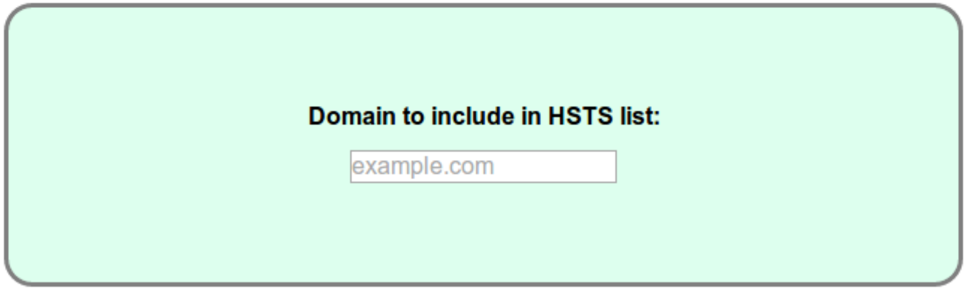
How do I get preloaded?

- 2012 to mid 2014:

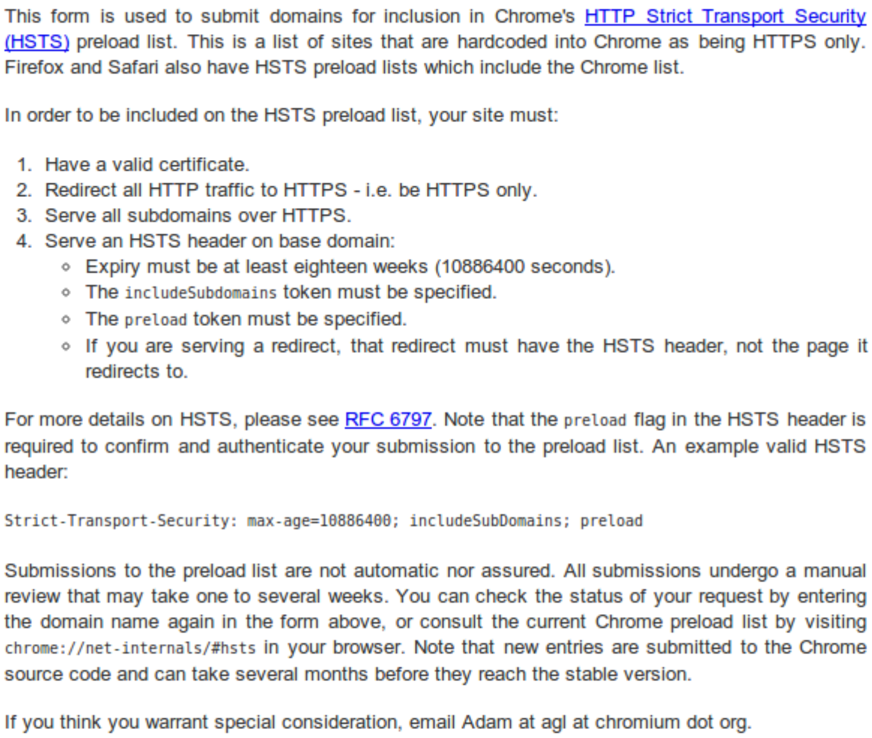
-via email, informal

- Now:

hstspreload.appspot.com



Domain to include in HSTS list:



This form is used to submit domains for inclusion in Chrome's HTTP Strict Transport Security (HSTS) preload list. This is a list of sites that are hardcoded into Chrome as being HTTPS only. Firefox and Safari also have HSTS preload lists which include the Chrome list.

In order to be included on the HSTS preload list, your site must:

1. Have a valid certificate.
2. Redirect all HTTP traffic to HTTPS - i.e. be HTTPS only.
3. Serve all subdomains over HTTPS.
4. Serve an HSTS header on base domain:
   - Expiry must be at least eighteen weeks (10886400 seconds).
   - The `includeSubdomains` token must be specified.
   - The `preload` token must be specified.
   - If you are serving a redirect, that redirect must have the HSTS header, not the page it redirects to.

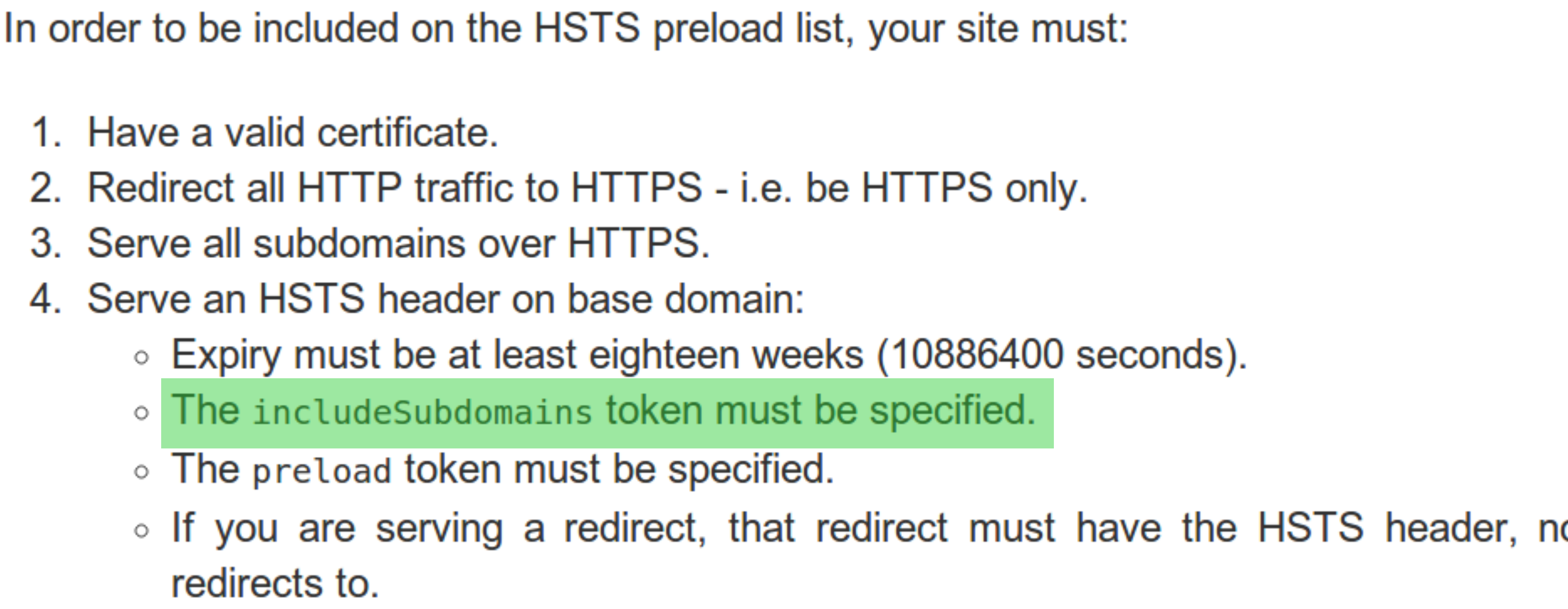
For more details on HSTS, please see RFC 6797. Note that the `preload` flag in the HSTS header is required to confirm and authenticate your submission to the preload list. An example valid HSTS header:

```
Strict-Transport-Security: max-age=10886400; includeSubDomains; preload
```

Submissions to the preload list are not automatic nor assured. All submissions undergo a manual review that may take one to several weeks. You can check the status of your request by entering the domain name again in the form above, or consult the current Chrome preload list by visiting `chrome://net-internals/#hsts` in your browser. Note that new entries are submitted to the Chrome source code and can take several months before they reach the stable version.

If you think you warrant special consideration, email Adam at agl at chromium dot org.

How do I get preloaded?

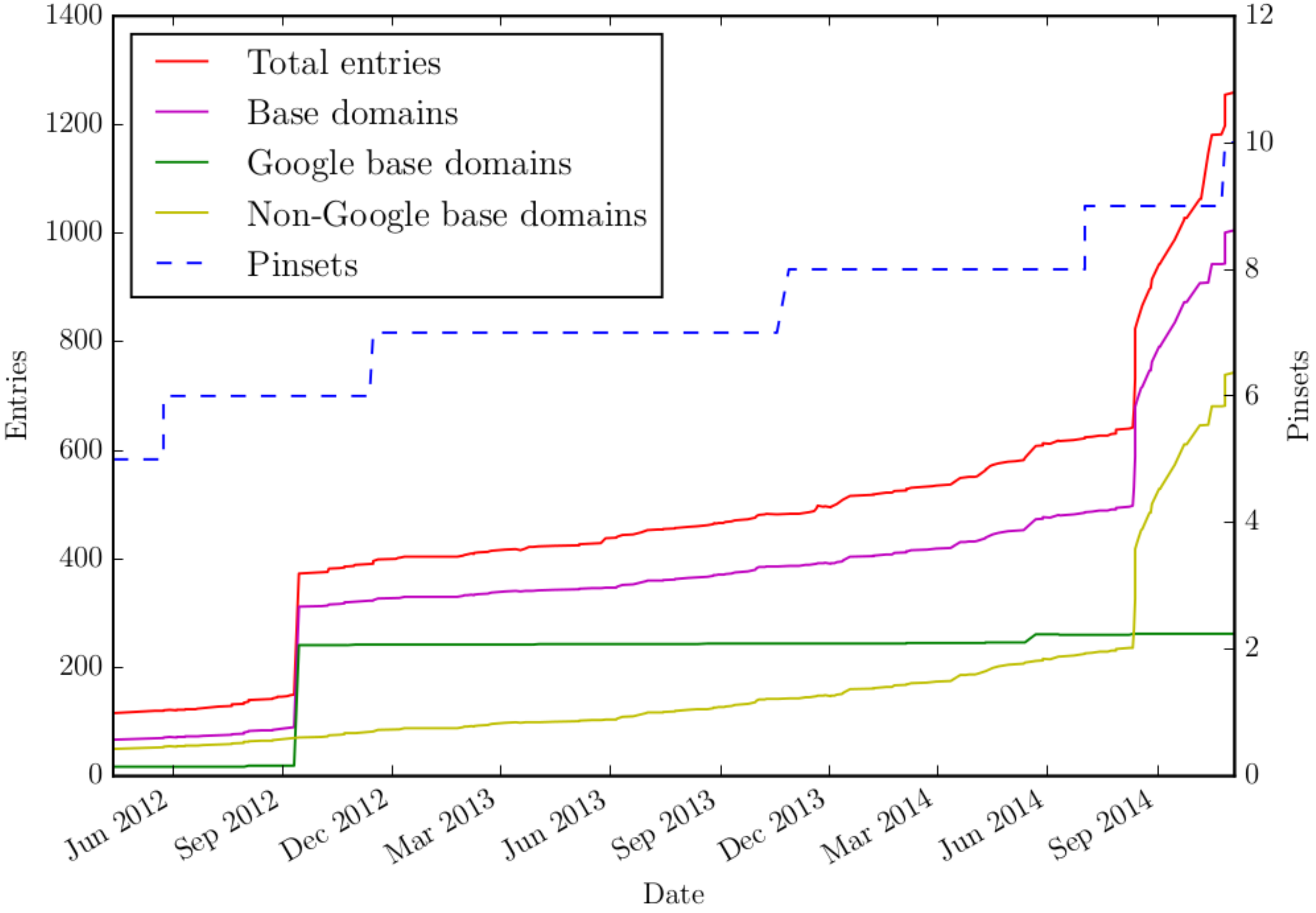


In order to be included on the HSTS preload list, your site must:

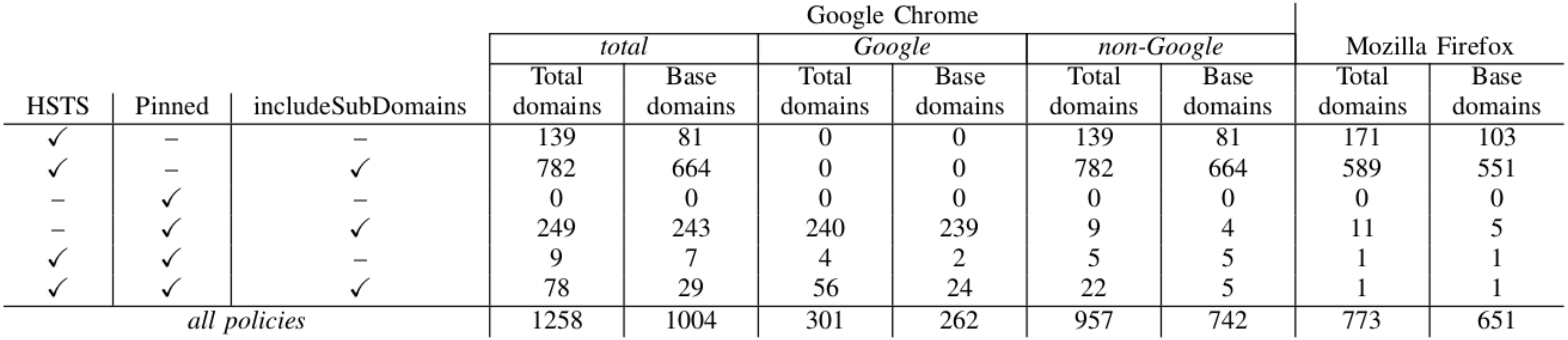
1. Have a valid certificate.
2. Redirect all HTTP traffic to HTTPS - i.e. be HTTPS only.
3. Serve all subdomains over HTTPS.
4. Serve an HSTS header on base domain:
   - Expiry must be at least eighteen weeks (10886400 seconds).
   - The `includeSubdomains` token must be specified.
   - The `preload` token must be specified.
   - If you are serving a redirect, that redirect must have the HSTS header, no redirects to.

(not retroactive)

Preloads growing in Chrome

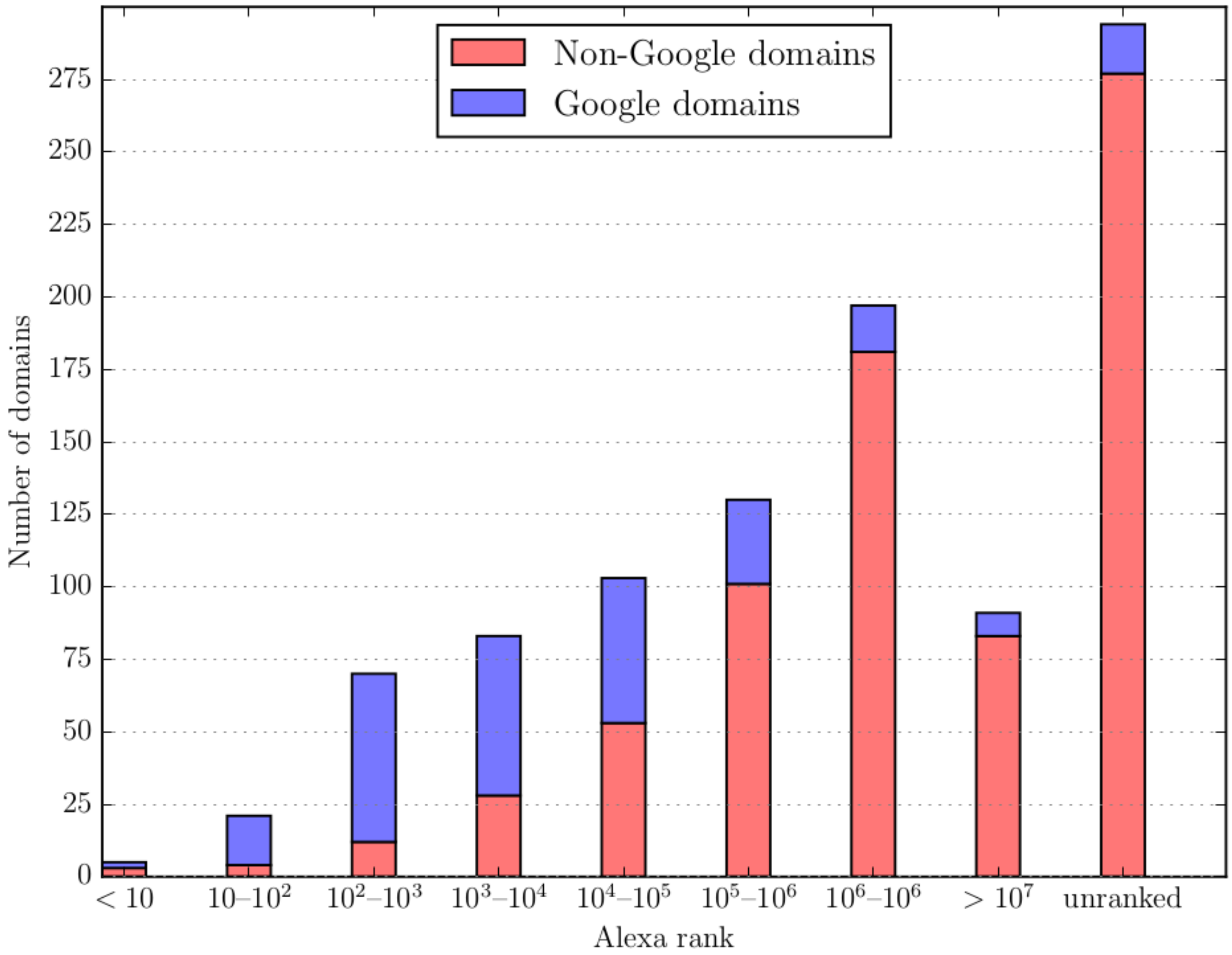


Policies vary considerably

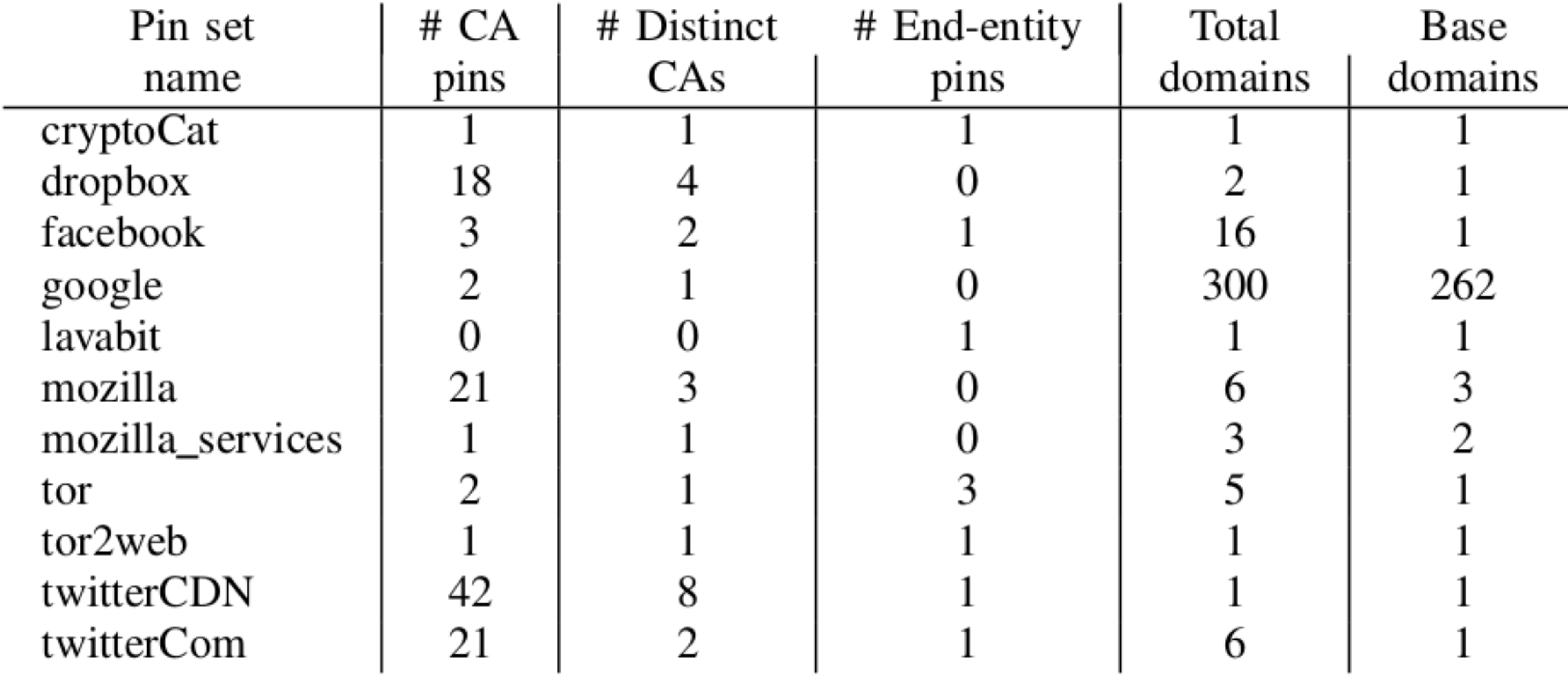


			Google Chrome							
			total		*Google*		*non-Google*		Mozilla Firefox	
HSTS	Pinned	includeSubDomains	Total domains	Base domains	Total domains	Base domains	Total domains	Base domains	Total domains	Base domains
✓	–	–	139	81	0	0	139	81	171	103
✓	–	✓	782	664	0	0	782	664	589	551
–	✓	–	0	0	0	0	0	0	0	0
–	✓	✓	249	243	240	239	9	4	11	5
✓	✓	–	9	7	4	2	5	5	1	1
✓	✓	✓	78	29	56	24	22	5	1	1
all policies			1258	1004	301	262	957	742	773	651

Many low-traffic sites preloaded



Few domains pinned, many big pin sets



Pin set name	# CA pins	# Distinct CAs	# End-entity pins	Total domains	Base domains
cryptoCat	1	1	1	1	1
dropbox	18	4	0	2	1
facebook	3	2	1	16	1
google	2	1	0	300	262
lavabit	0	0	1	1	1
mozilla	21	3	0	6	3
mozilla_services	1	1	0	3	2
tor	2	1	3	5	1
tor2web	1	1	1	1	1
twitterCDN	42	8	1	1	1
twitterCom	21	2	1	6	1

List is often stale

- Of 742 non-Google HSTS domains
    - 77 returned 404
    - 23 permanently redirected to HTTP
    - > 10% stale!
    - Lavabit dead, still pinned

- Some stale Google domains too
    - 4 permanent HTTP redirects

Firefox policy

- Must be included in Chrome
- Must respond over HTTPS
- Must set a dynamic HSTS header
  - Must set an age > 18 weeks

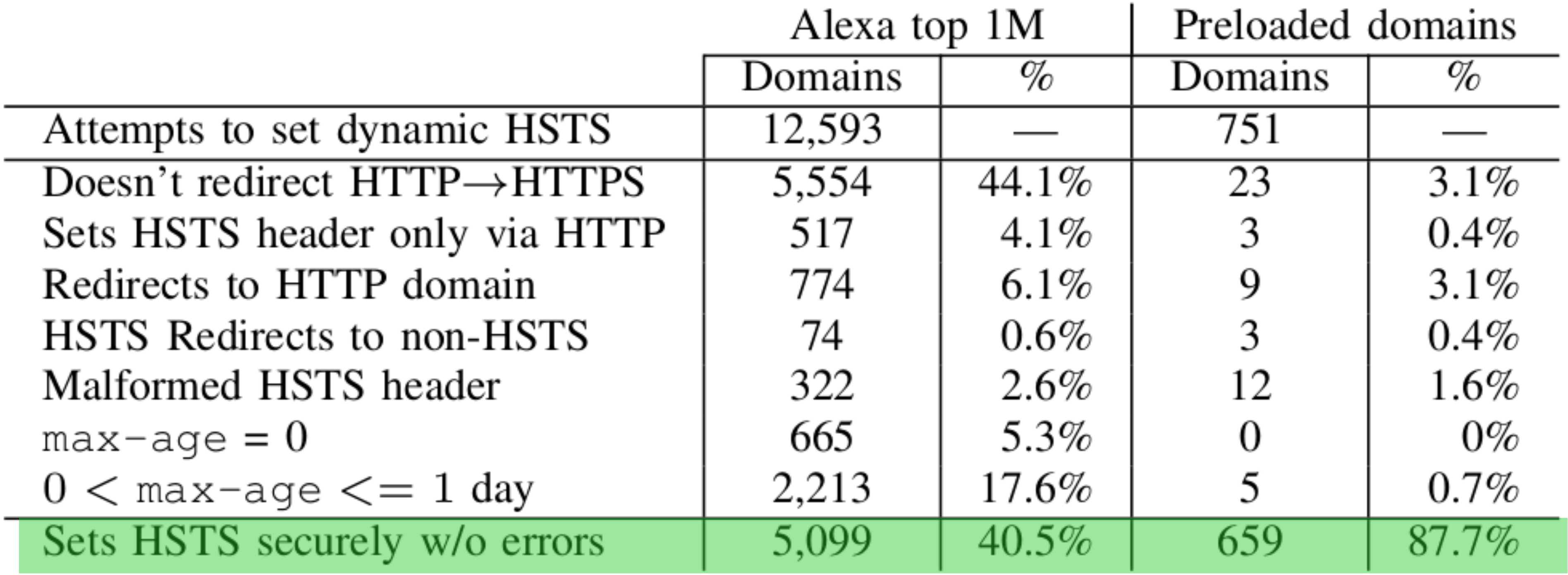
Few domains setting HSTS headers

- **1.1%** of the top **1M** domains (Alexa rank)
  - 5.2% of those have max-age=0

- Many non-HSTS domains redirect to HTTPS
  - **5.8%** of the top **1M** domains

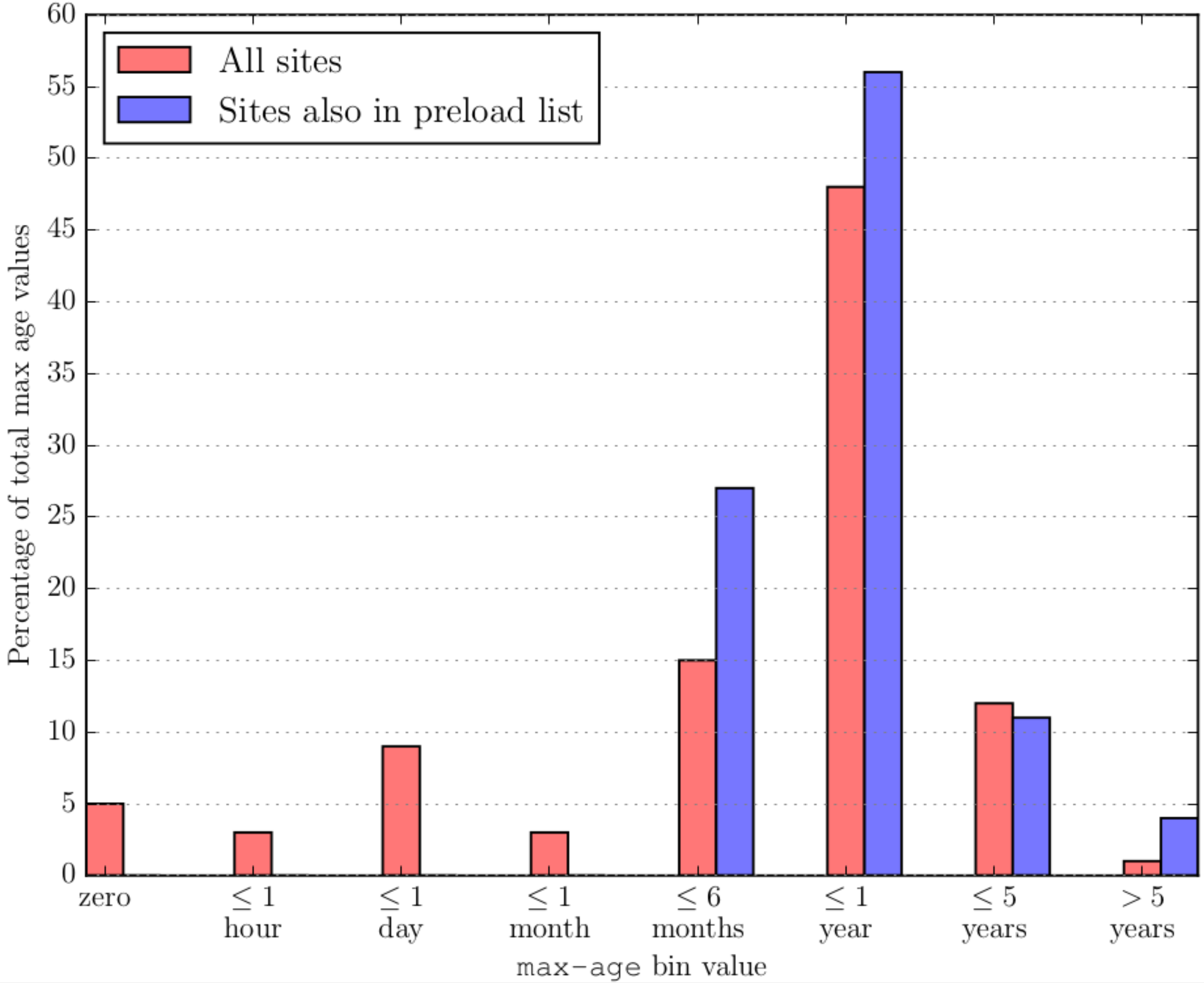
- **34%** of preloaded domains not setting headers
  - **65%** of preloaded Google domains

Many domains set HSTS incorrectly



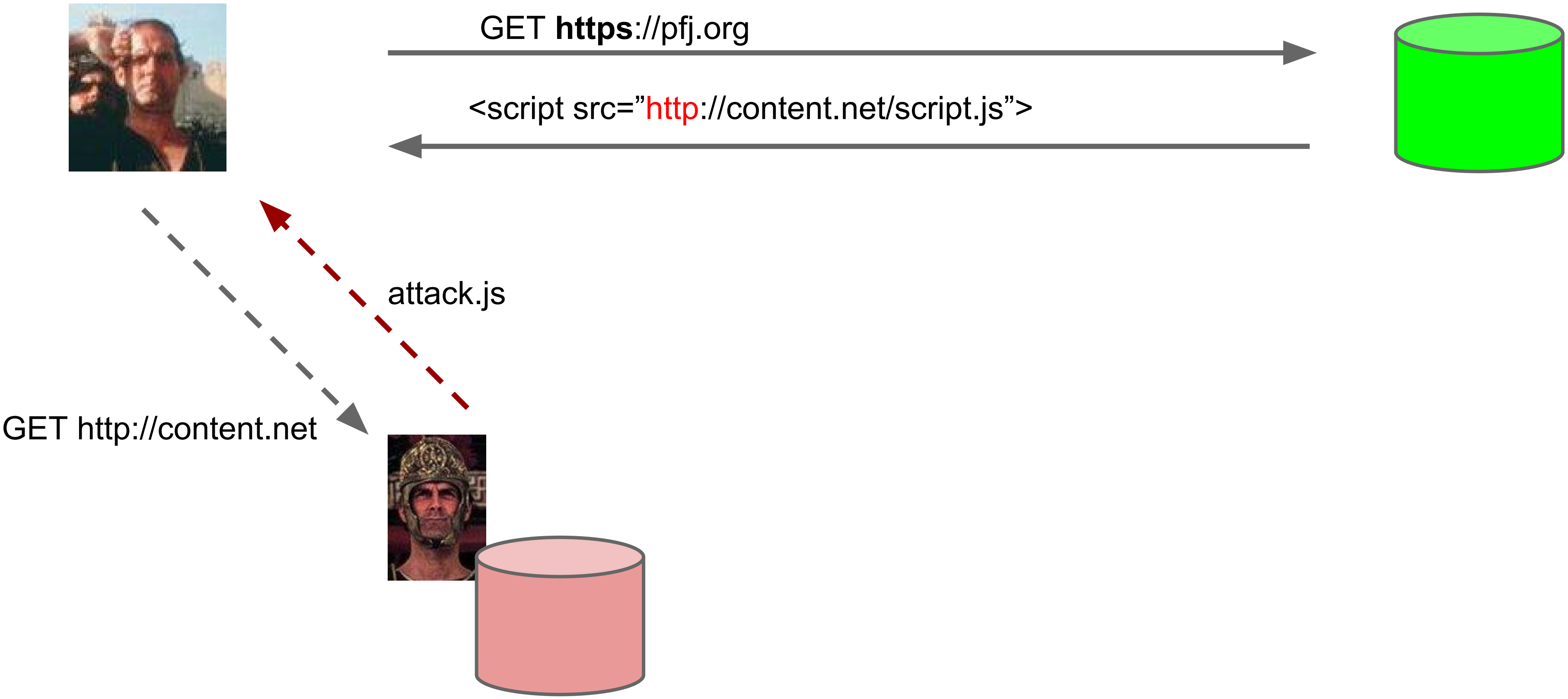
	Alexa top 1M		Preloaded domains	
	Domains	%	Domains	%
Attempts to set dynamic HSTS	12,593	—	751	—
Doesn't redirect HTTP→HTTPS	5,554	44.1%	23	3.1%
Sets HSTS header only via HTTP	517	4.1%	3	0.4%
Redirects to HTTP domain	774	6.1%	9	3.1%
HSTS Redirects to non-HSTS	74	0.6%	3	0.4%
Malformed HSTS header	322	2.6%	12	1.6%
`max-age` = 0	665	5.3%	0	0%
0 < `max-age` <= 1 day	2,213	17.6%	5	0.7%
Sets HSTS securely w/o errors	5,099	40.5%	659	87.7%

Max-age values vary significantly



Mixed content

Classic mixed content



GET **https**://pfj.org

<script src=”http://content.net/script.js”>

attack.js

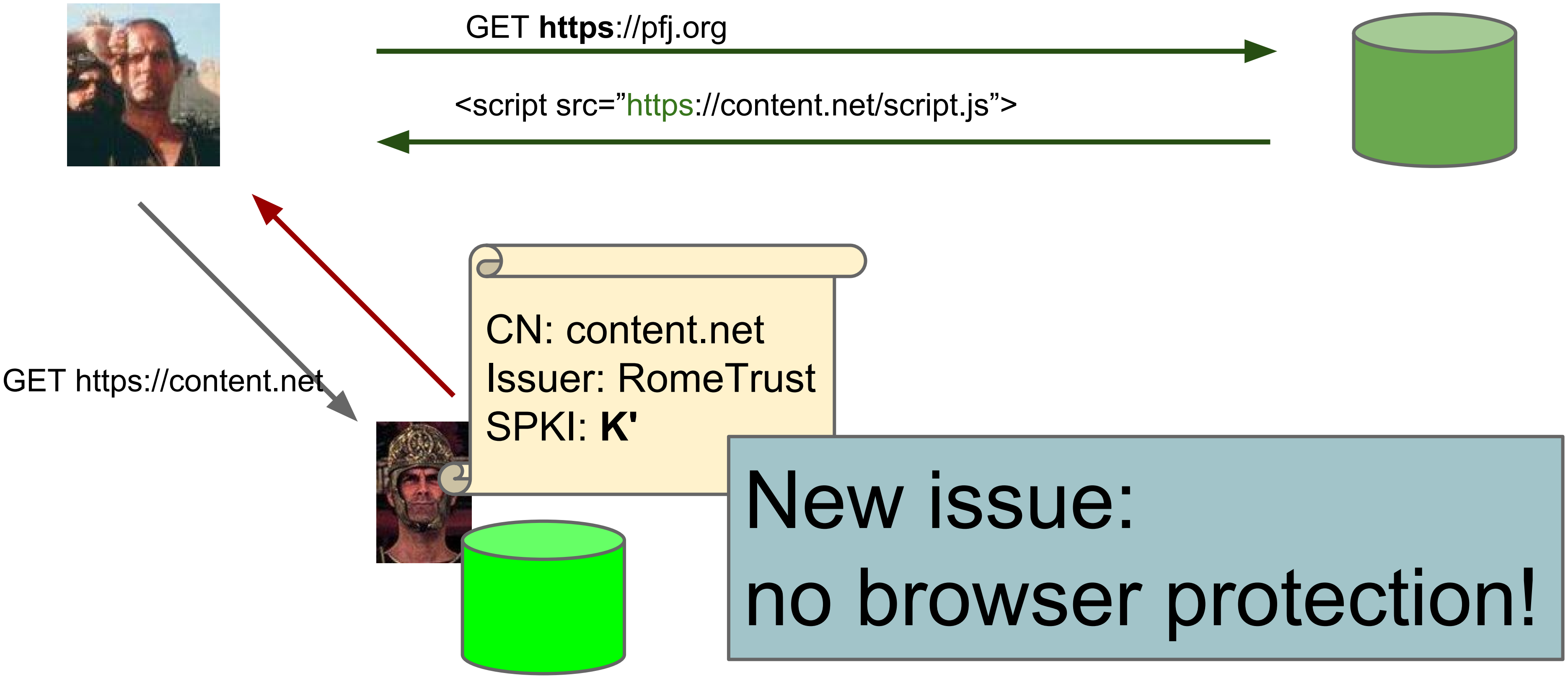
GET http://content.net

Mixed content now (mostly) blocked

- Active content (blocked as of 2012)
  - scripts
  - stylesheets
  - iframes
  - Flash
  - fonts

- Passive content (allowed)
  - images
  - video
  - audio

Mixed pinning content



GET **https**://pfj.org

<script src=”https://content.net/script.js”>

GET https://content.net

CN: content.net

Issuer: RomeTrust

SPKI: **K'**

New issue:

no browser protection!

Passive mixed content is common

- Every pinset affected
  - Over **66,000** passive resources
  - **99%** images

Active mixed content also common!

- 5/10 pinsets, **24,477** resources
    - Twitter, Dropbox, Cryptocat, Tor, DoubleClick

resource type	**#**
script	15,540
stylesheet	7,195
xmlhttprequest	1,515
subdocument	170
font	49

Causes of mixed content

- Twitter
    - scripts from Akamai, Facebook
- Tor
    - Videos-from www.youtube-nocookie.com
- DoubleClick
    - various advertising scripts

- Unpinned subdomains
    - syndication.twitter.com
    - blog.cryptocat.com
    - forum.dropbox.com

Expanded-pinset mixed content

- Twitter
  - scripts from twitterCDN (intentional)
- Various domains
  - ssl.google-analytics.com

Plain mixed content ☹

- 30,000 observations
  - More than mixed pinning!
- Only one active
  - doubleclick.net

Interaction with cookies

RFC2965: Same-origin policy for cookies

Domain Defaults to the effective request-host. (Note that because there is no dot at the beginning of effective request-host, the default Domain can only domain-match itself.)

Domain=value
OPTIONAL. The value of the Domain attribute specifies the domain for which the cookie is valid. If an explicitly specified value does not start with a dot, the user agent supplies a leading dot.

Host names can be specified either as an IP address or a HDN string. Sometimes we compare one host name with another. (Such comparisons SHALL be case-insensitive.) Host A's name domain-matches host B's if

* their host name strings string-compare equal; or

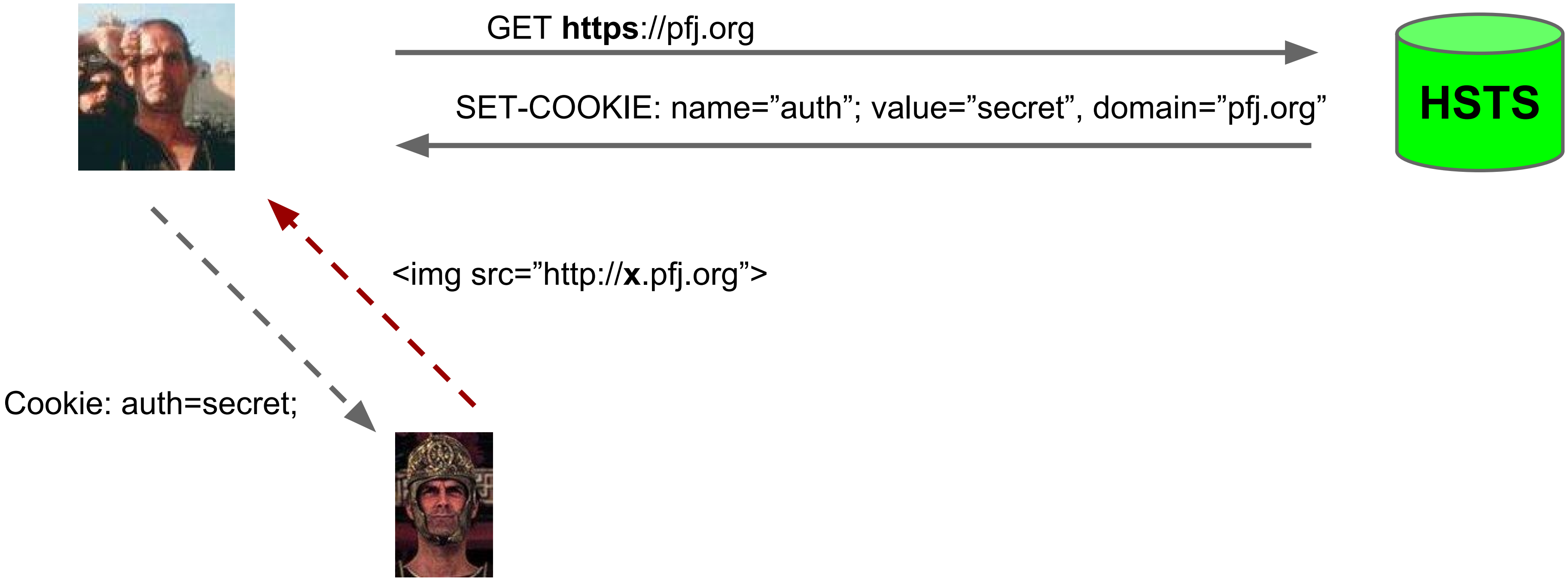
* A is a HDN string and has the form NB, where N is a non-empty name string, B has the form .B', and B' is a HDN string. (So, x.y.com domain-matches .Y.com but not Y.com.)

Note that domain-match is not a commutative operation: a.b.c.com domain-matches .c.com, but not the reverse.

RFC2965 in plain English

- If you supply a domain=parameter, it’s a wildcard
- If you omit the domain=parameter, it’s exact
  - Except on Internet Explorer, because ?

Cookie-stealing attack



GET **https**://pfj.org

SET-COOKIE: name=”auth”; value=”secret”, domain=”pfj.org”

HSTS

<img src=”http://**x**.pfj.org”>

Cookie: auth=secret;

Preventing cookie-stealing (HSTS)

- Set HSTS with **`includeSubdomains`**
- Mark cookies with **`secure`** attribute

Cookie-stealing in the wild

- **10,174** cookies at **2,460** domains not covered by HSTS

- **10,174** (98%) not marked as **`secure`**

- Several from large domains
  - PayPal, Lastpass, USAA

- Mostly tracking cookies and IDS
  - No auth tokens identified

Preventing cookie-stealing (Pinning)

- Set pins with `includeSubdomains`

No equivalent for pinning!

Cookie-stealing from pinned domains

- Every pinned domain vulnerable!
  - Excluding those setting **`includeSubdomains`**
  - 75 total cookies visible

- Several login cookies vulnerable
  - Facebook, Twitter
  - Known vulnerability

Google’s (now fixed) pinning hole

```
{ "name": "google.com", "include_subdomains": true, "pins": "google" }
```

```
// play.google.com doesn't have include_subdomains because of crbug.com/327834.
{ "name": "play.google.com", "mode": "force-https", "pins": "google" }
```

Insecure links also a problem

- Initial connections to HSTS not protected

Takeaways: web security is hard!

- Users don’t read specs

- Spec writers don’t know about real constraints

Takeaways: standards not holistic

- Different formats for headers, preloads

- Preload format not standardized, changing

- DANE has a different format as well

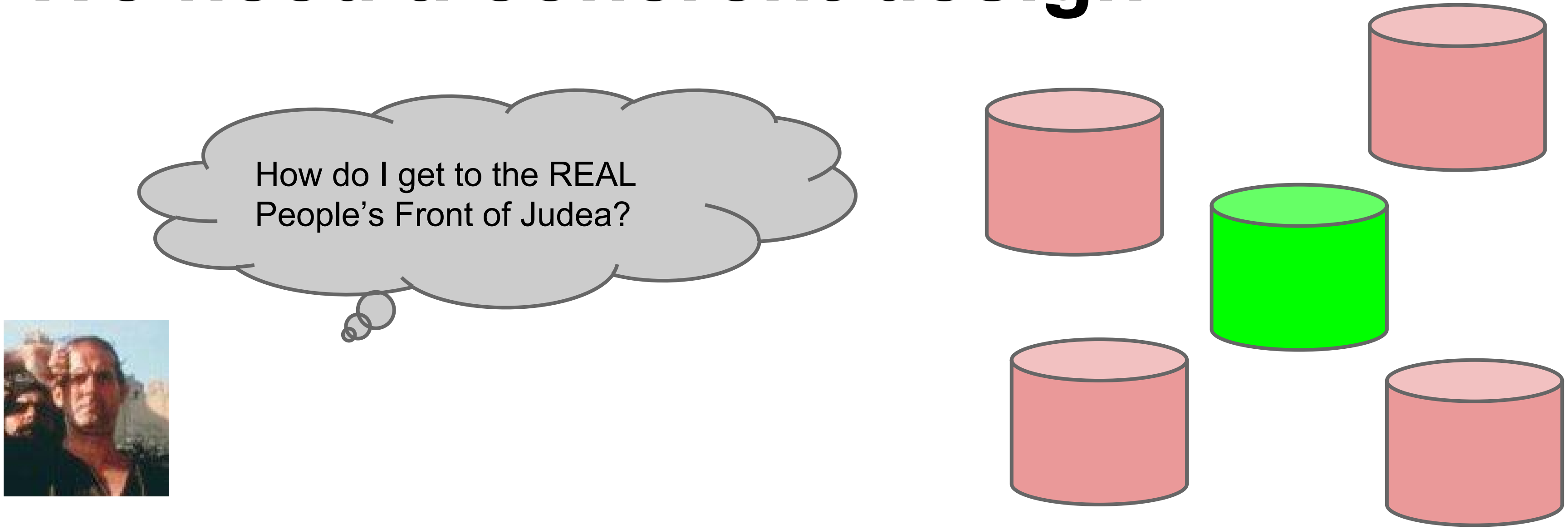
Better defaults may help

- Pinning, HSTS default should be **`includeSubdomain`**
- **`secure`** default should extend to cover pinning
- Cookies should require explicit wildcard notation!

Thank you

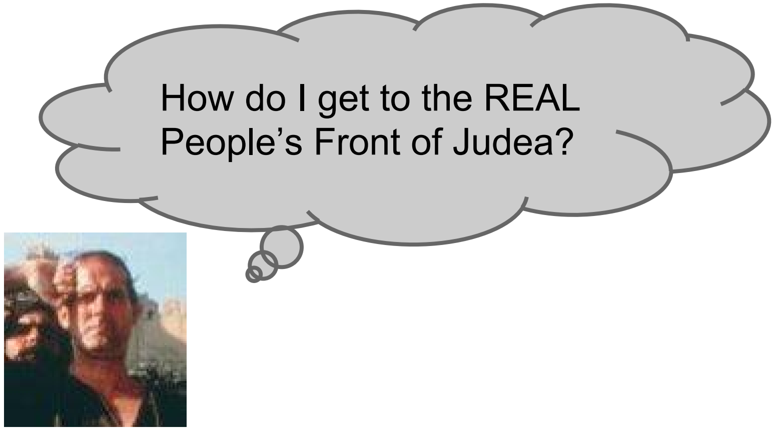
jbonneau@princeton.edu
mkranch@princeton.edu

We need a coherent design



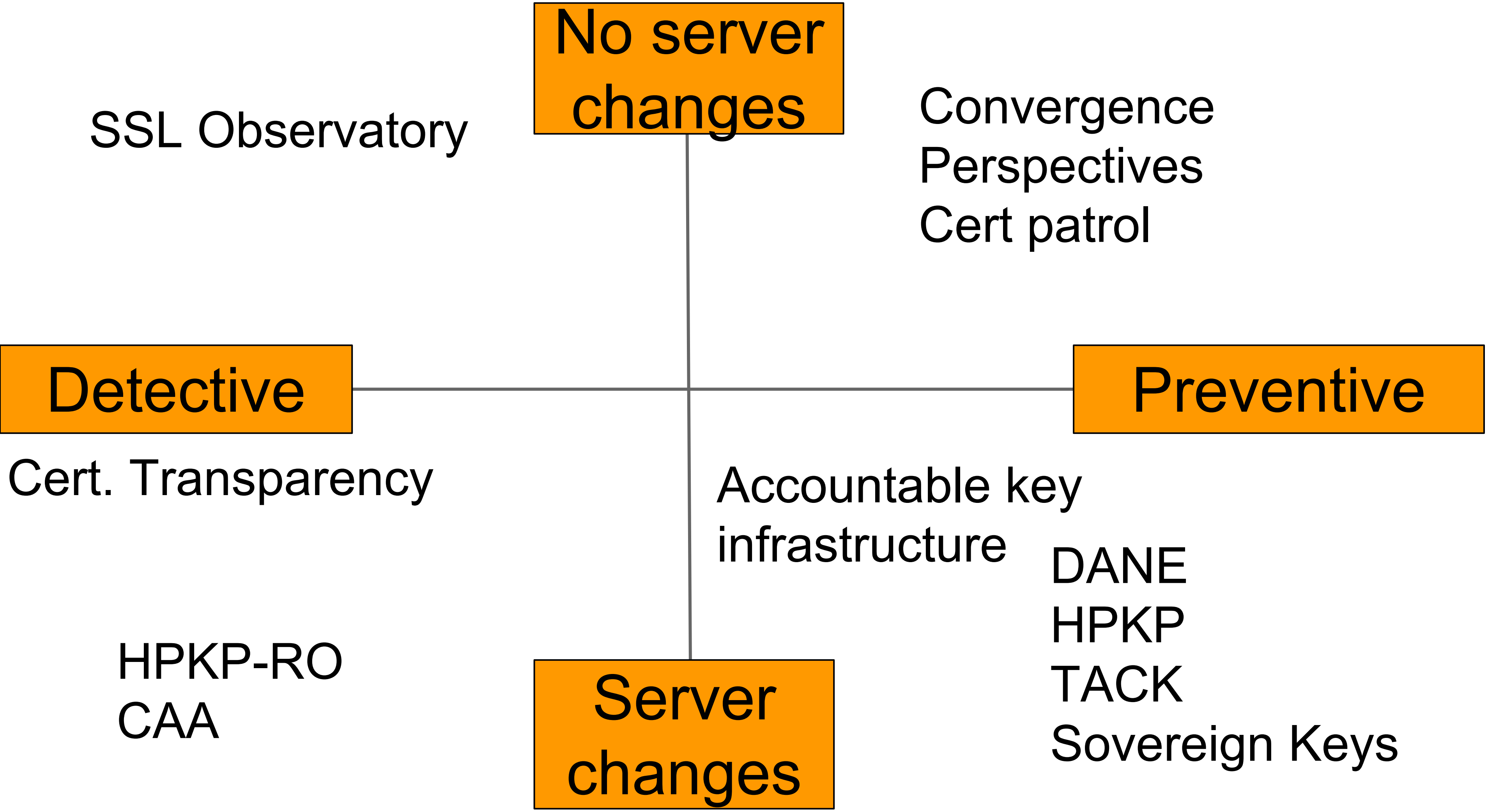
- Do they support HTTPS?
- Which public keys should I accept?
- What protocol version do they support?

Many ways to learn Transport Sec.Policy



- **Preloads** (hardcoded)
  - Browser or extensions
- **Authorities**
  - DNS, CAs, Notaries, crowdsource
- **Continuity**
  - What they've done before (*implicit*)
  - What they've promised to keep doing (*explicit*)
- **Introduction**
  - When following a hyperlink

Many proposals to upgrade HTTPS



No server changes

SSL Observatory

Convergence
Perspectives
Cert patrol

Detective

Preventive

Cert. Transparency

Accountable key infrastructure

DANE
HPKP
TACK
Sovereign Keys

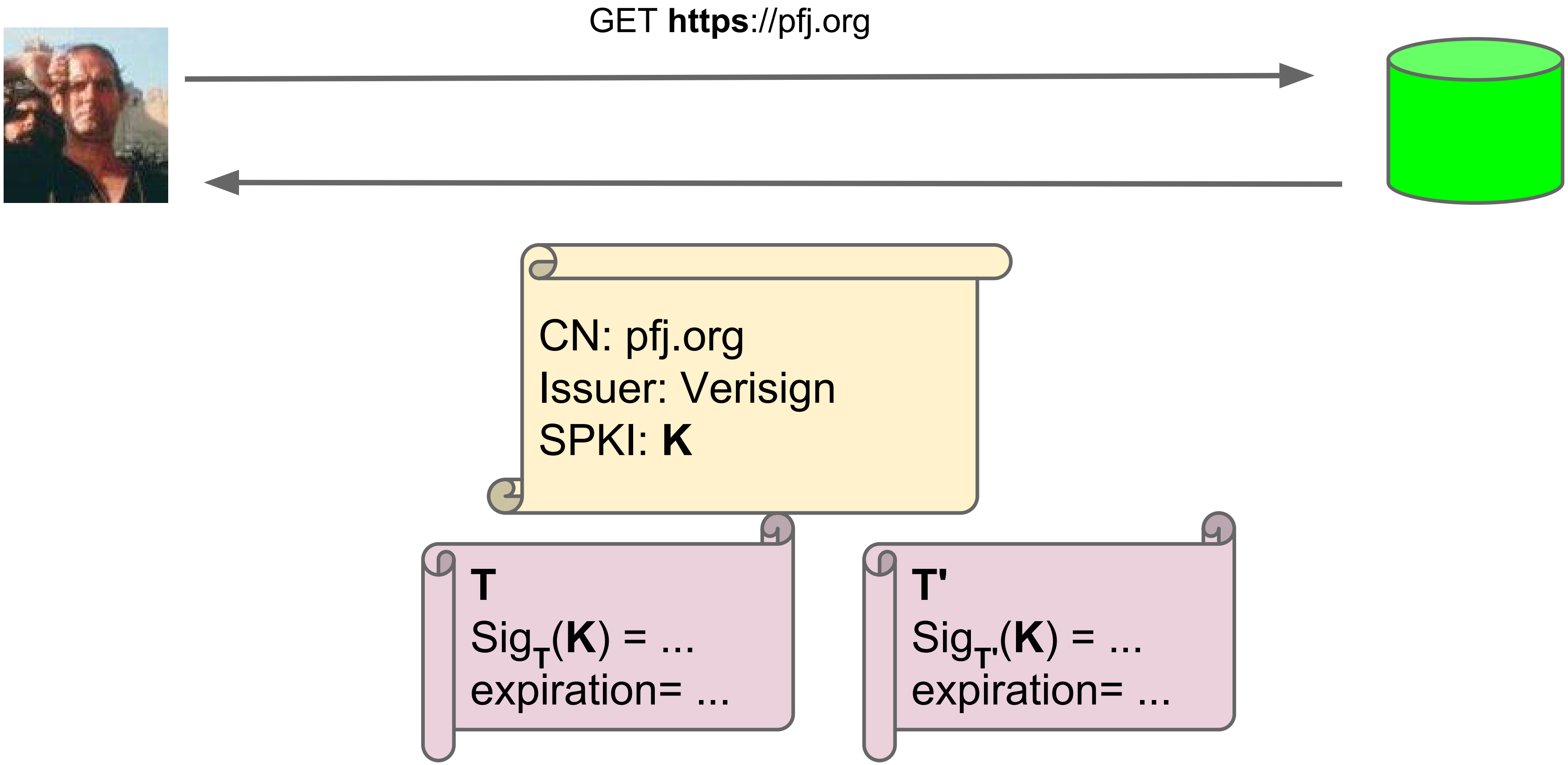
HPKP-RO
CAA

Server changes

Linked web navigation model

users only reach new domains via hyperlinks, beginning with a set of domains with preloaded security policies.

Discovering TACK keys



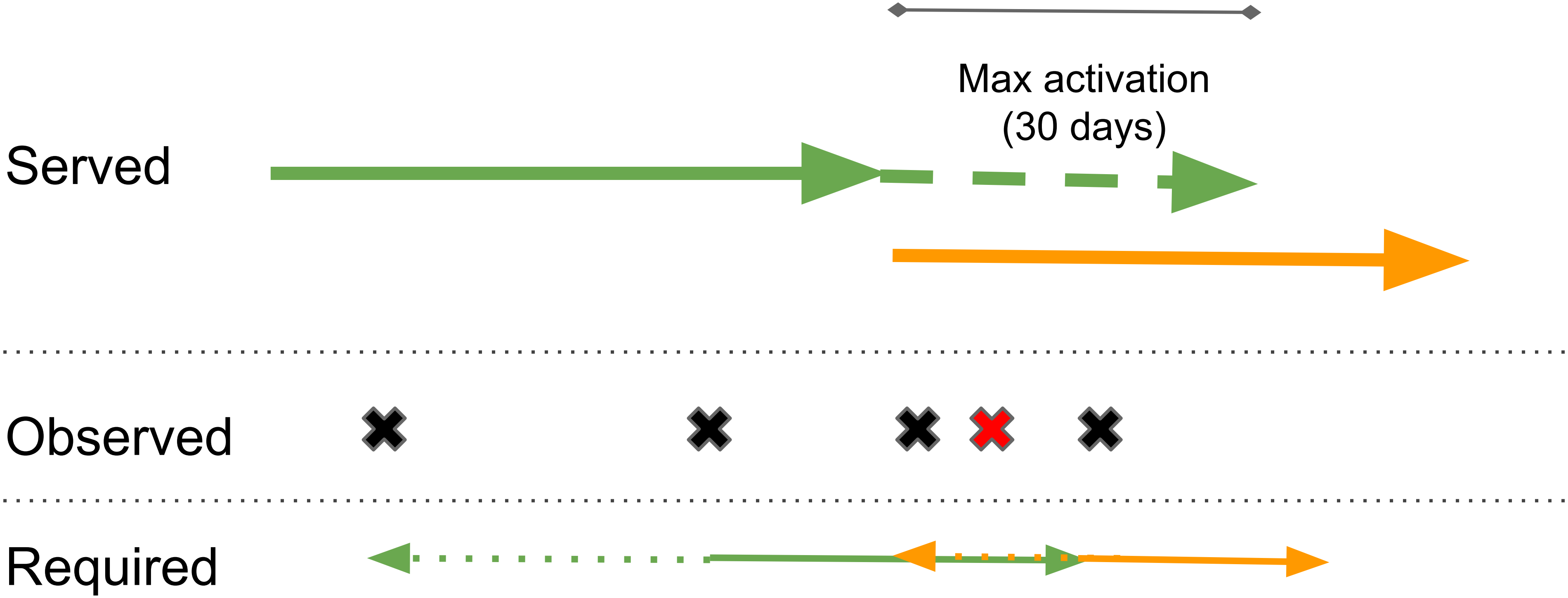
GET **https**://pfj.org

CN: pfj.org
Issuer: Verisign
SPKI: **K**

T
Sig$_{T}$(**K**) = ...
expiration= ...

T'
Sig$_{T'}$(**K**) = ...
expiration= ...

TACK activation (rollover)



Max activation (30 days)

Served

Observed

Required

Malicious s-links?

- Can only make security policy *stricter*
  - Can never undermine ambient policy

- No persistent effects
  - No domain bricking

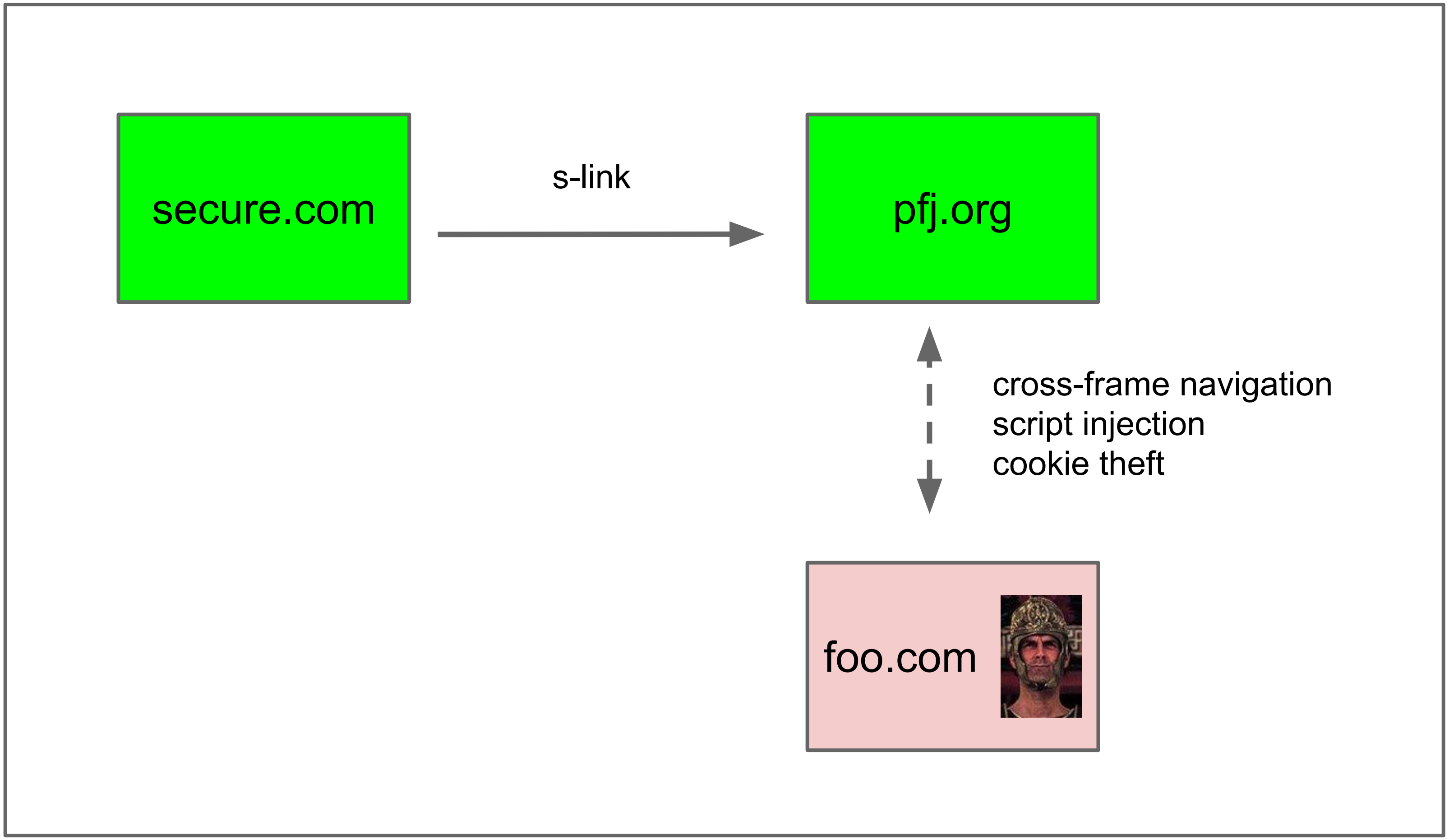
- UI ≈ 404 (not found)
  - Limit risk or "warning fatigue"

Stale s-links

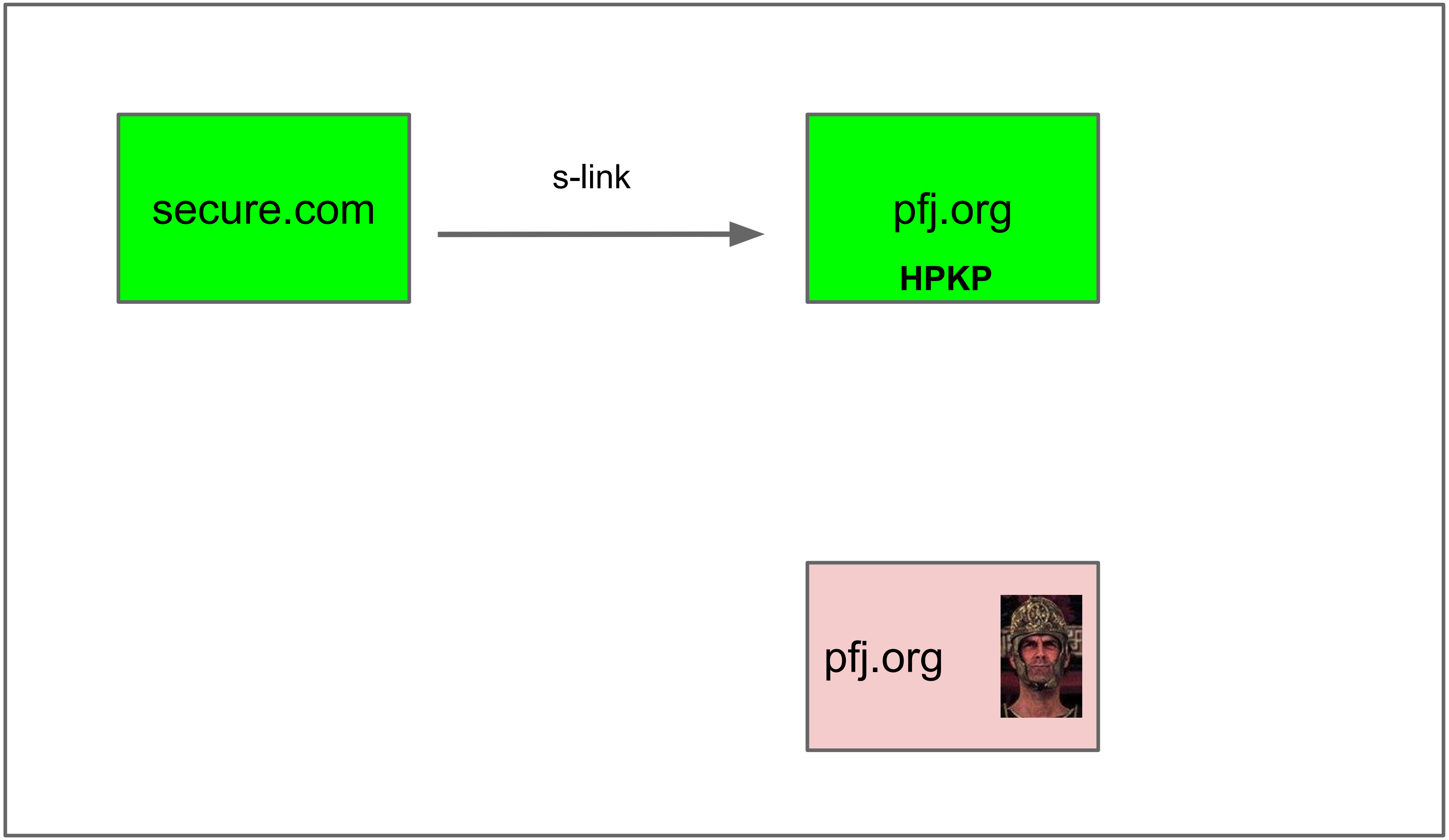
- Expiry is mandatory
  - In absolute time, to require constant changes

- Links can always go stale
  - Hopefully, existing user model is to blame introducer

S-links and the same origin policy



S-links and the same origin policy



Upgrading security policy

- Need to re-check ALL cached resources
  - HTTP cache
  - HTML5 localStorage/WebCache
  - TLS saved sessions
  - Cookies
  - etc.

- Need to do so atomically

- No issues for non-framed content
  - For example, script libraries

Case study: crawlers and HTTPS

- Redirects
- <link rel="canonical" href="...
- HSTS headers?

Secure introduction

- IDEA: for web navigation, linking website can indicate security policy in-band

- Already exists for HSTS!
- Effects of an HTTPS link:
  - mandatory
  - ephemeral
  - transparent to users
  - easy to deploy

An early attempt: YURLs

https**y**://*cl7h3f...mayi@pfj.org/

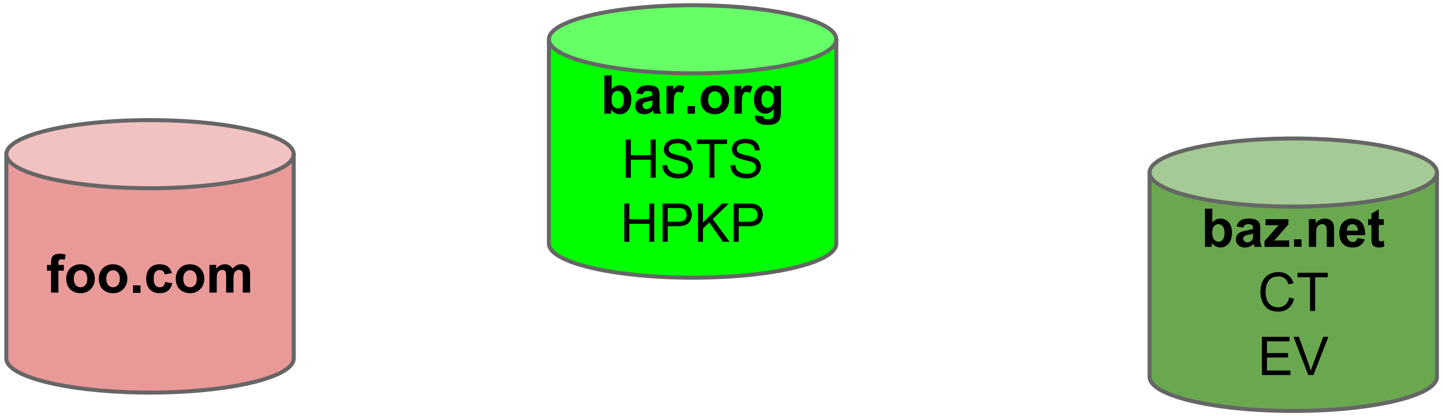
Why HTML?

- Extensible
- Backwards compatible
- Easy to deploy

Challenges:

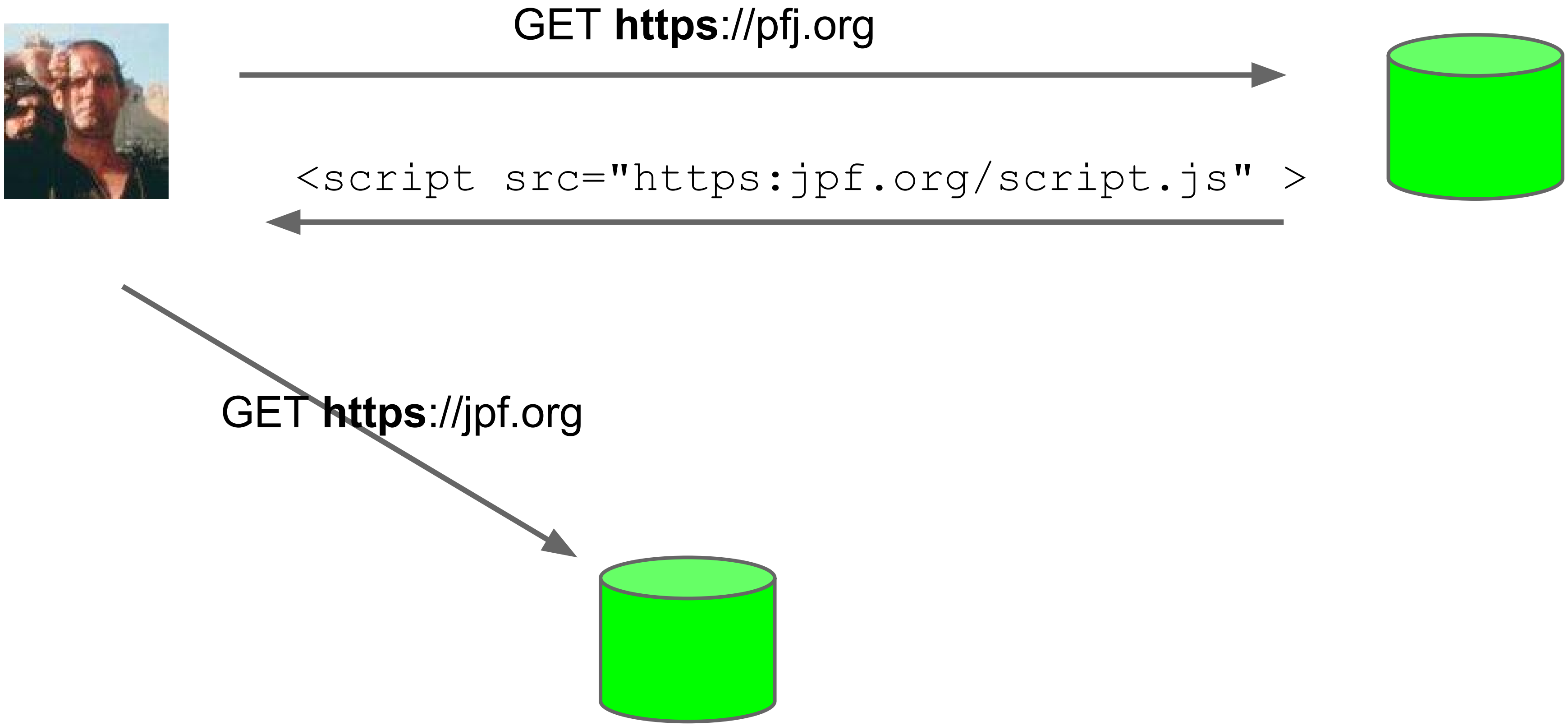
- Redirects
- Copy/paste

Major design constraint: compatibility



Browsers must know what to expect *prior to the initial connection*

Introduction: HTTPS links



GET **https**://pfj.org

```
<script src="https:jpf.org/script.js" >
```

GET **https**://jpf.org

Where did HSTS go right?

- Effective against HTTPS stripping

Security

- Incrementally deployable
- Relatively easy "off switch"

Deployability

- Transparent to end users
- High trust agility
- High trust affordance

Usability

Clean-slate designs

- QUIC
  - Google
- MinimaLT
  - Petullo, Zhang, Solworth, Bernstein, Lange 2014

HTTPS bugs

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer
signedParams, uint8_t *signature, UInt16 signatureLen)
{
  OSStatus err;
  ...

  if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
  if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
  ...

fail:
  SSLFreeBuffer(&signedHashes);
  SSLFreeBuffer(&hashCtx);
  return err;
}
```

HTTPS bugs

```
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYHOST, true);
```

PHP Manual Entry for `CURLOPT_SSL_VERIFYHOST`:

1 to check the existence of a common name in the SSL peer certificate. 2 to check the existence of a common name and also verify that it matches the hostname provided. In production environments the value of this option should be kept at 2 (default value).

from Georgiev et al. 2012 “The Most Dangerous Code in the World”

Core problems

- Flexibility at a protocol level
  - Ciphersuites
  - Choice of CA for domains
  - Choice of public key for each domain
  - Protocol version
  - Choice to deploy HTTPS at all!

- Inflexibility of implementations
  - Browsers must support *every* server
  - Middleware boxes block attempted improvements

Key players

- Certification Authorities (CAs)
  - Incentives vary, but mostly survival dominates

- Browser vendors
  - Security, but with zero false positives

- Webmasters
  - Mostly, low latency and no bricking

Threat model

Malicious government

Control a CA:

Rome**Trust**

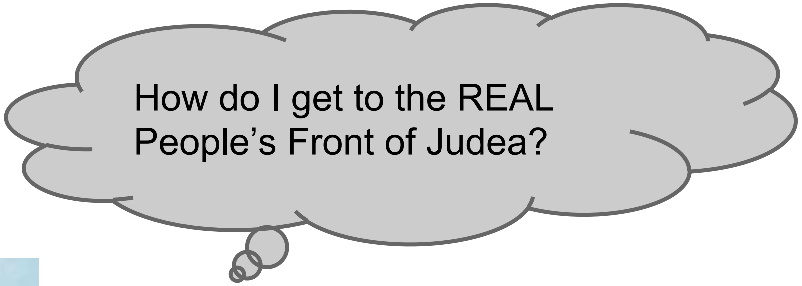
Control an ISP:

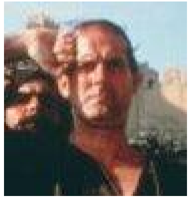
Rome**Cast**

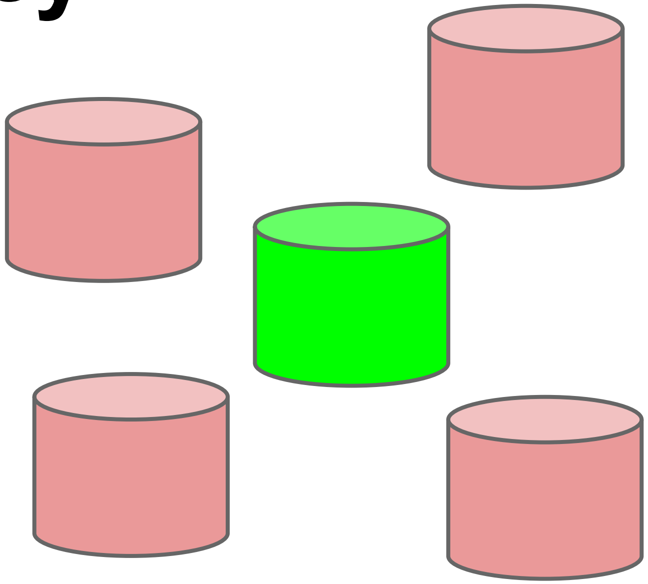
Limitations:

- Don't control all servers
- Don't control browser

Transport security policy

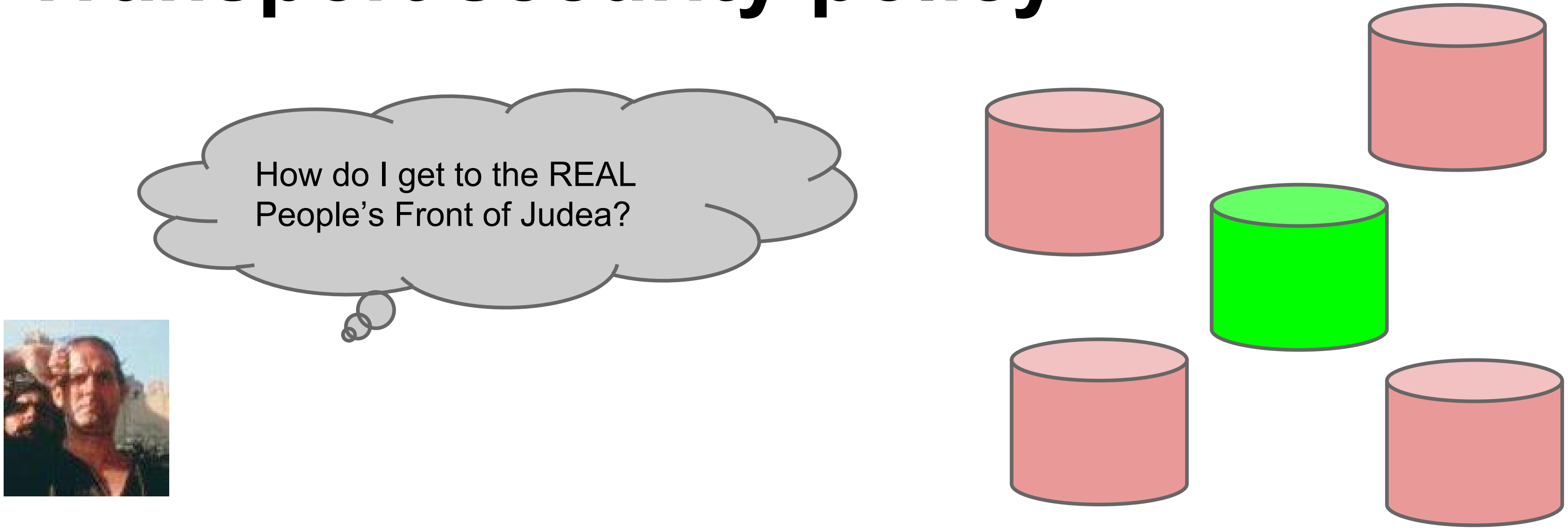






- Do they support HTTPS?
- What is their public key?
- What protocol version do they support?

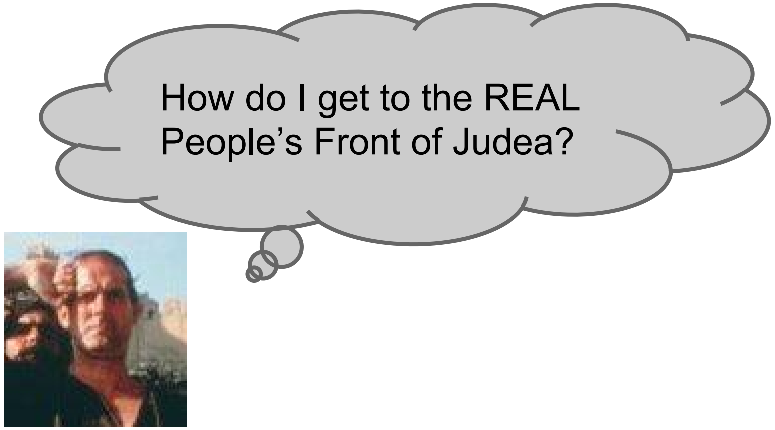
Transport security policy



How do I get to the REAL People’s Front of Judea?

- Do they support HTTPS?
- Which public keys should I accept?
- What protocol version do they support?

Ways to learn Transport Security Policy



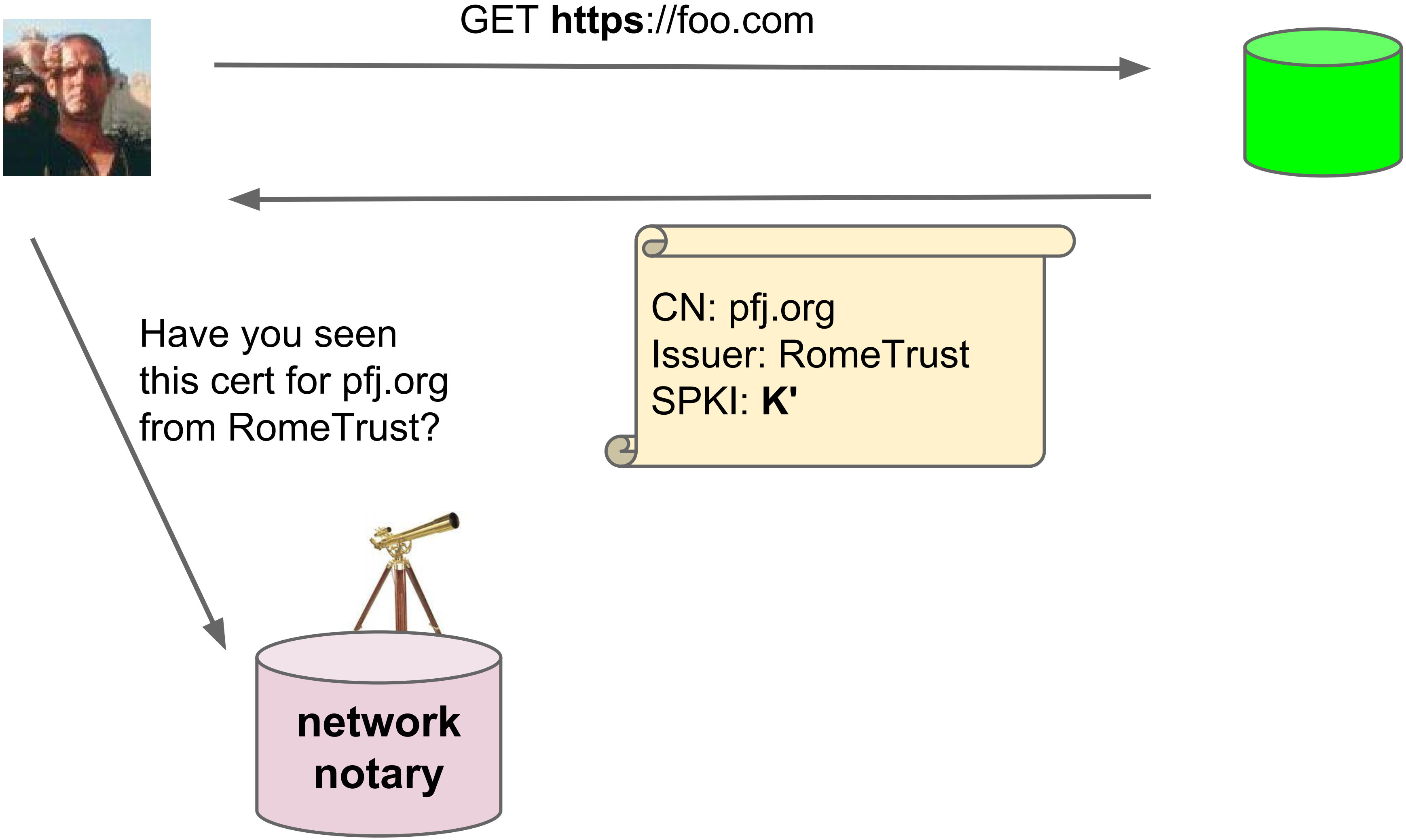
- **Preloads** (hardcoded)
  - Browser or extensions
- **Authorities**
  - DNS, CAs, Notaries, crowdsource
- **Continuity**
  - What they’ve done before (*implicit*)
  - What they’ve promised to keep doing (*explicit*)
- **Introduction**
  - When following a hyperlink

Authority: DNSSEC

- DANE
    - Hoffman, Schlyter 2012
    - Standards track RFC

- CAA
    - Hallam-Baker, Stradling 2013
    - Standards-track RFC

Authority: Network Perspectives



GET **https**://foo.com

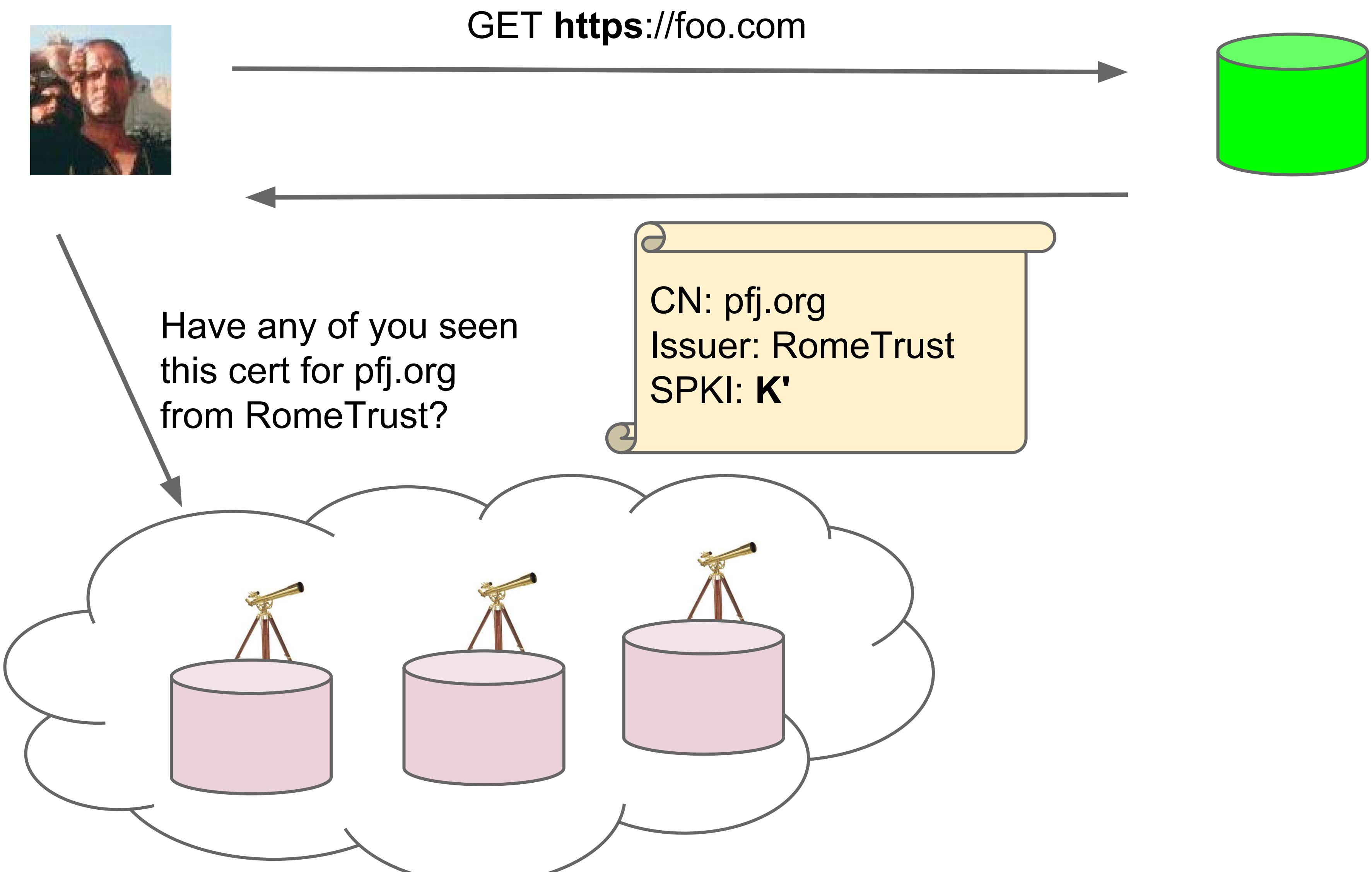
CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

Have you seen
this cert for pfj.org
from RomeTrust?

network
notary

Authority: Convergence

CONVERGENCE Beta

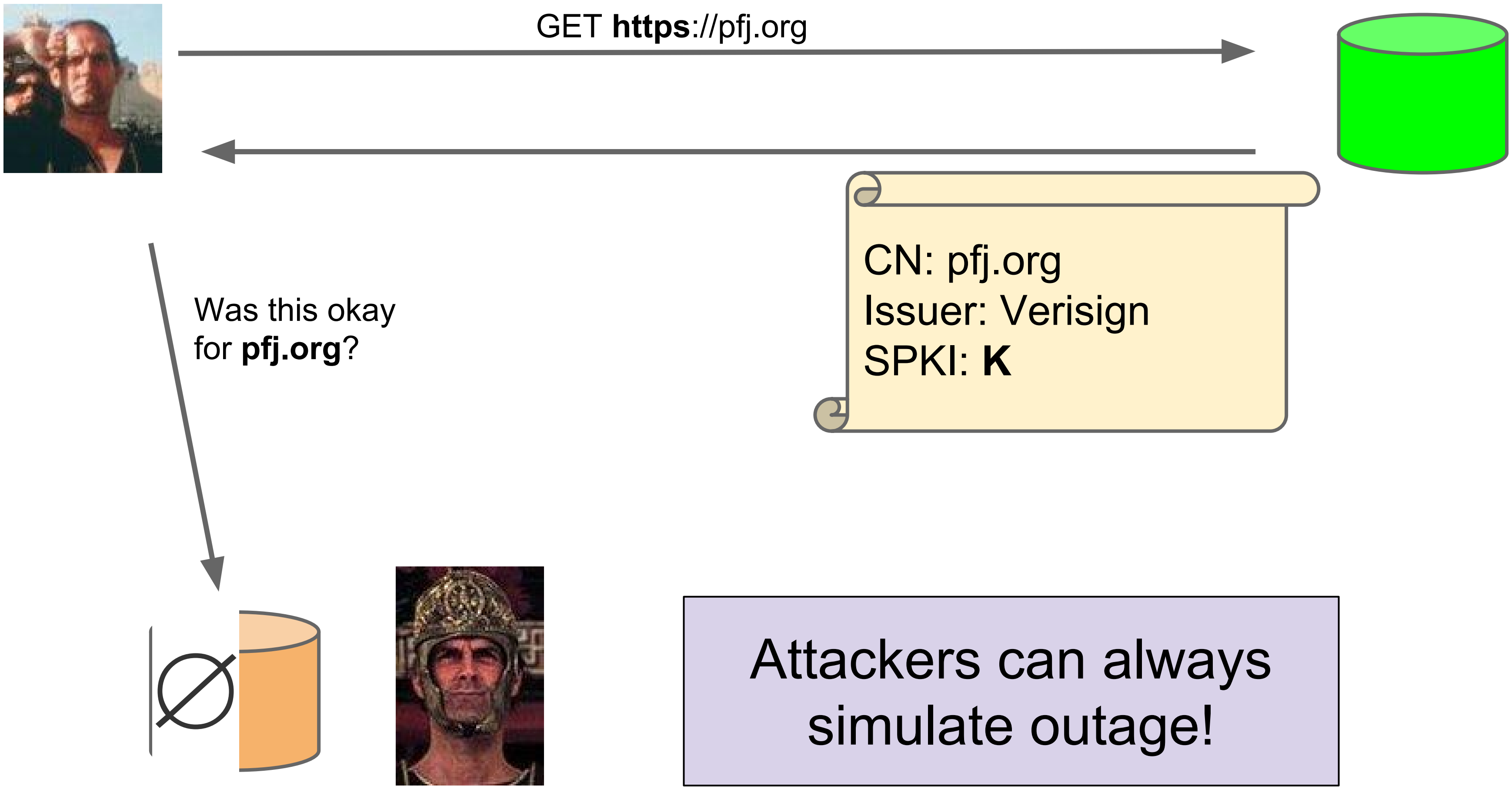


GET **https**://foo.com

CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

Have any of you seen
this cert for pfj.org
from RomeTrust?

Why out-of-band Authorities fail



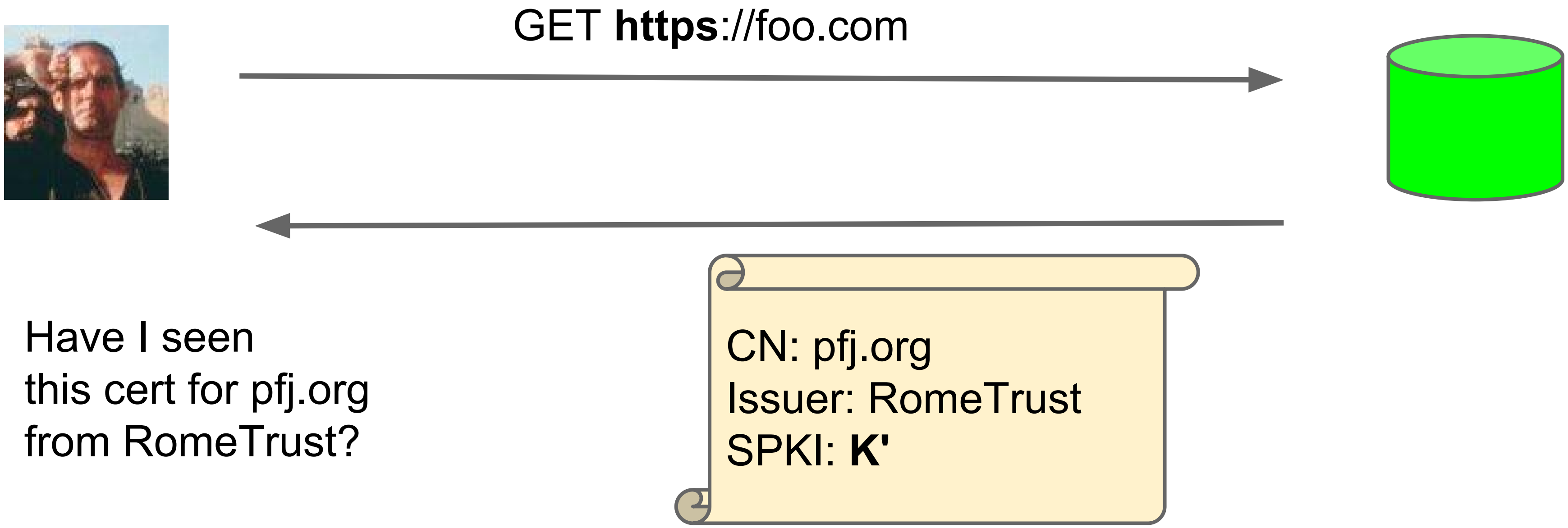
GET **https**://pfj.org

CN: pfj.org
Issuer: Verisign
SPKI: **K**

Was this okay
for **pfj.org**?

Attackers can always simulate outage!

Continuity (implicit)

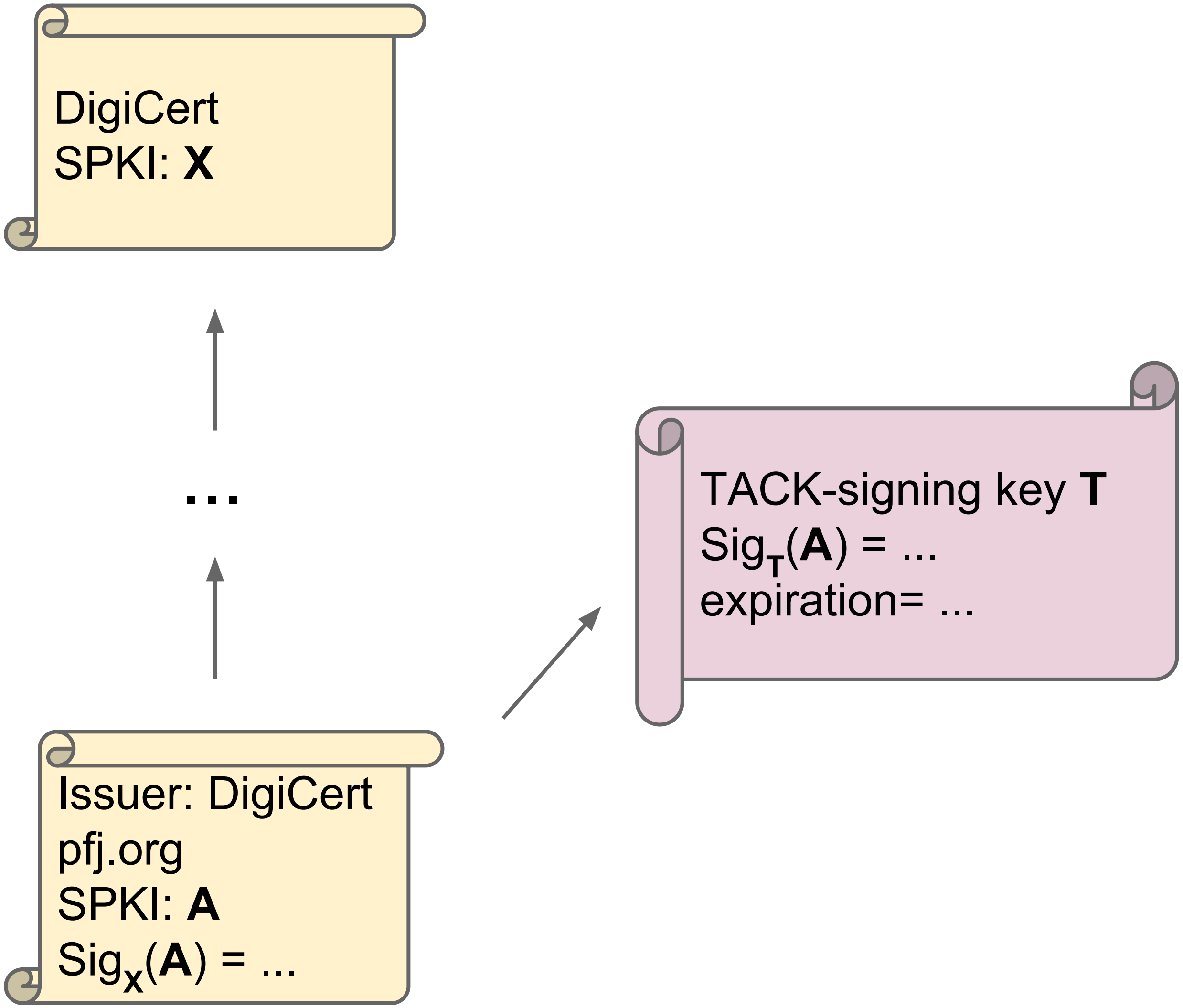


GET **https**://foo.com

CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

Have I seen
this cert for pfj.org
from RomeTrust?

Continuity (explicit): **TACK**



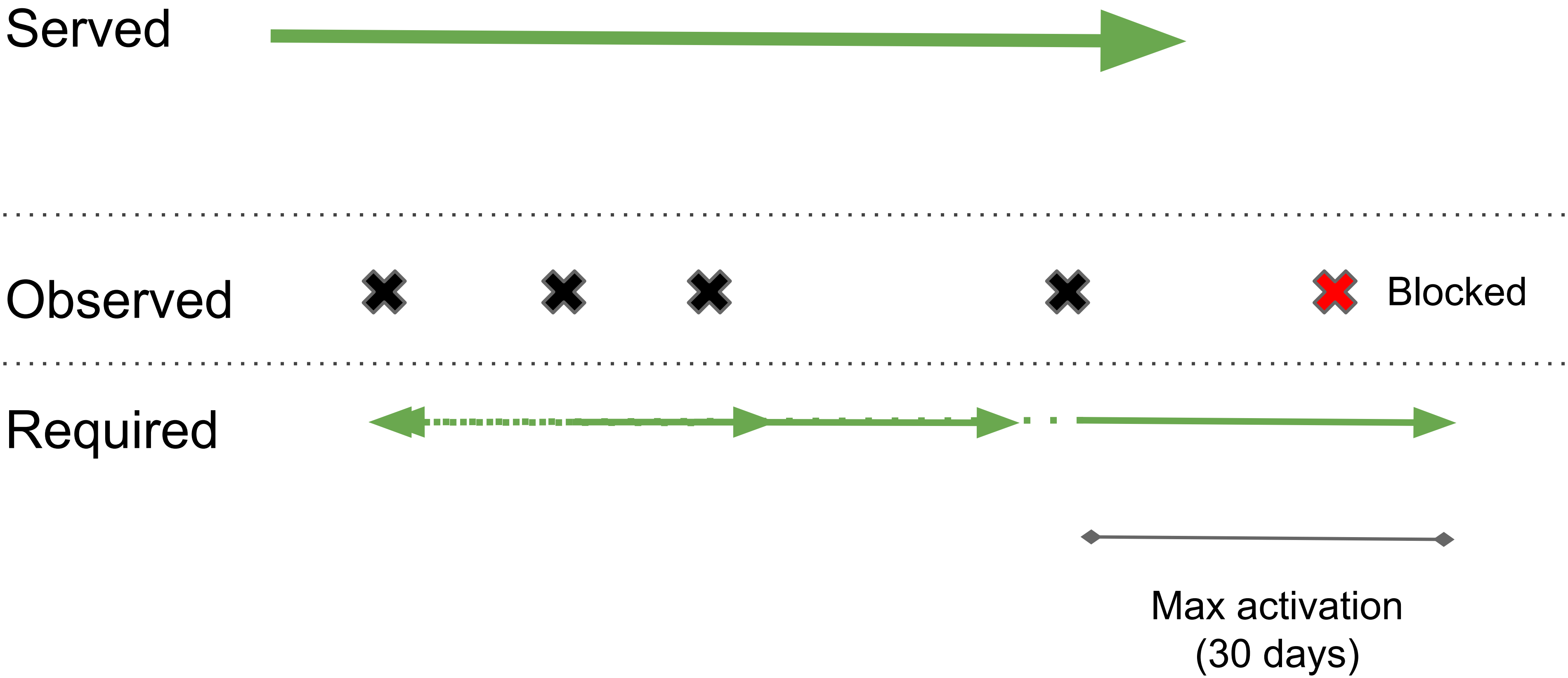
DigiCert
SPKI: **X**

...

TACK-signing key **T**
Sig_T(**A**) = ...
expiration= ...

Issuer: DigiCert
pfj.org
SPKI: **A**
Sig_X(**A**) = ...

TACK activation (simple case)



Served

Observed

Blocked

Required

Max activation
(30 days)

TACK

- Marlinspike, Perring 2012
    - Internet draft, TLS working group
- Compared to HPKP
    - Lower level
    - More flexible
    - More complex
    - Safer against domain bricking
- Rough equivalent: domain-bound CA
    - With HPKP pins

S-LINKS

Introduction: S-links

```
<a link-security="expiry=1357849989;
pin-sha256=YWRm...cnF=;
pin-sha256=LPJN...mCQ=;"
href="https://pfj.org">secure link!</a>
```

secure link!

S-links directives

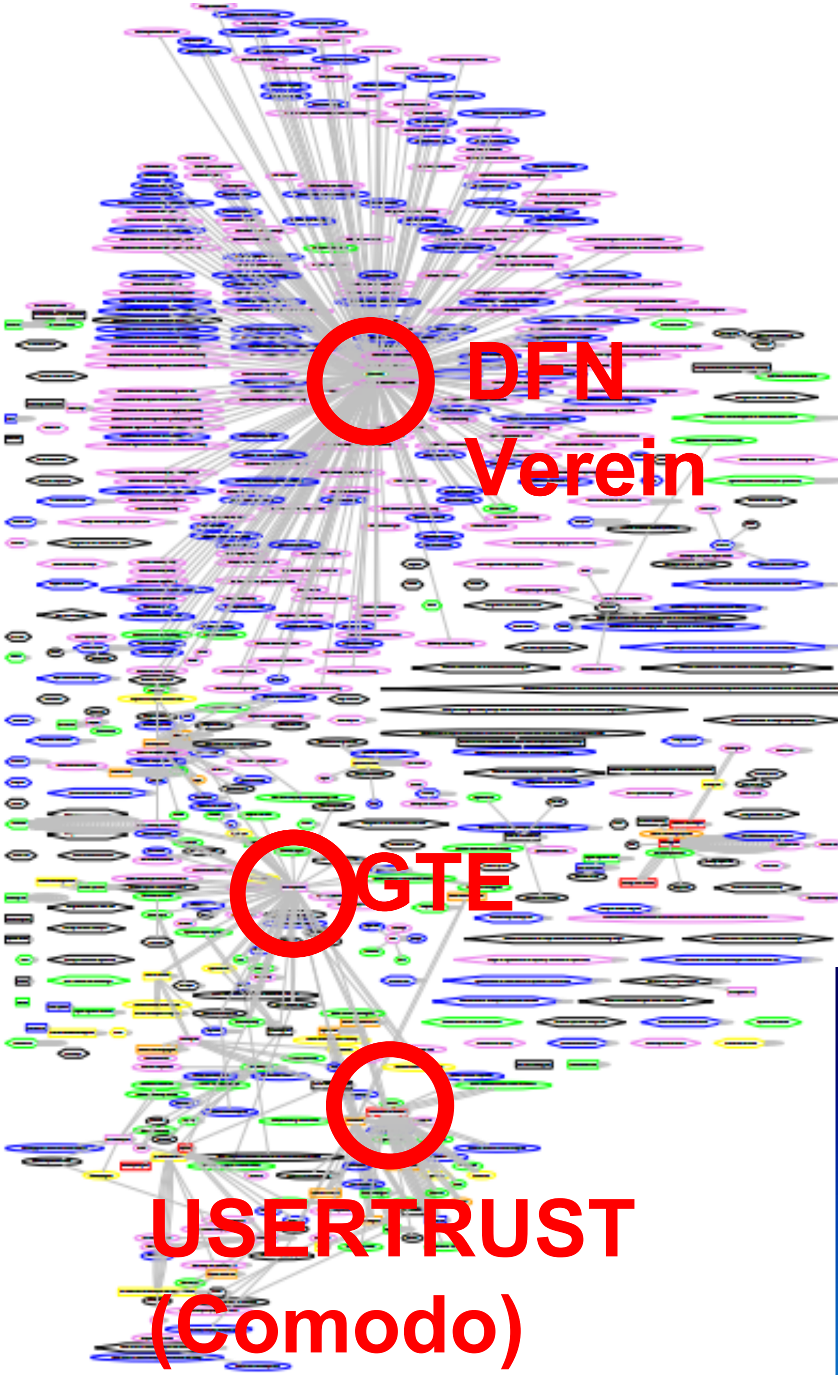
- Key pins
- CT mandatory
- EV mandatory
- Minimum TLS version
- ...

- Expiry

Who might set s-links?

- Search engines
- Social media sites
- Link aggregators

Detective/forensic approaches



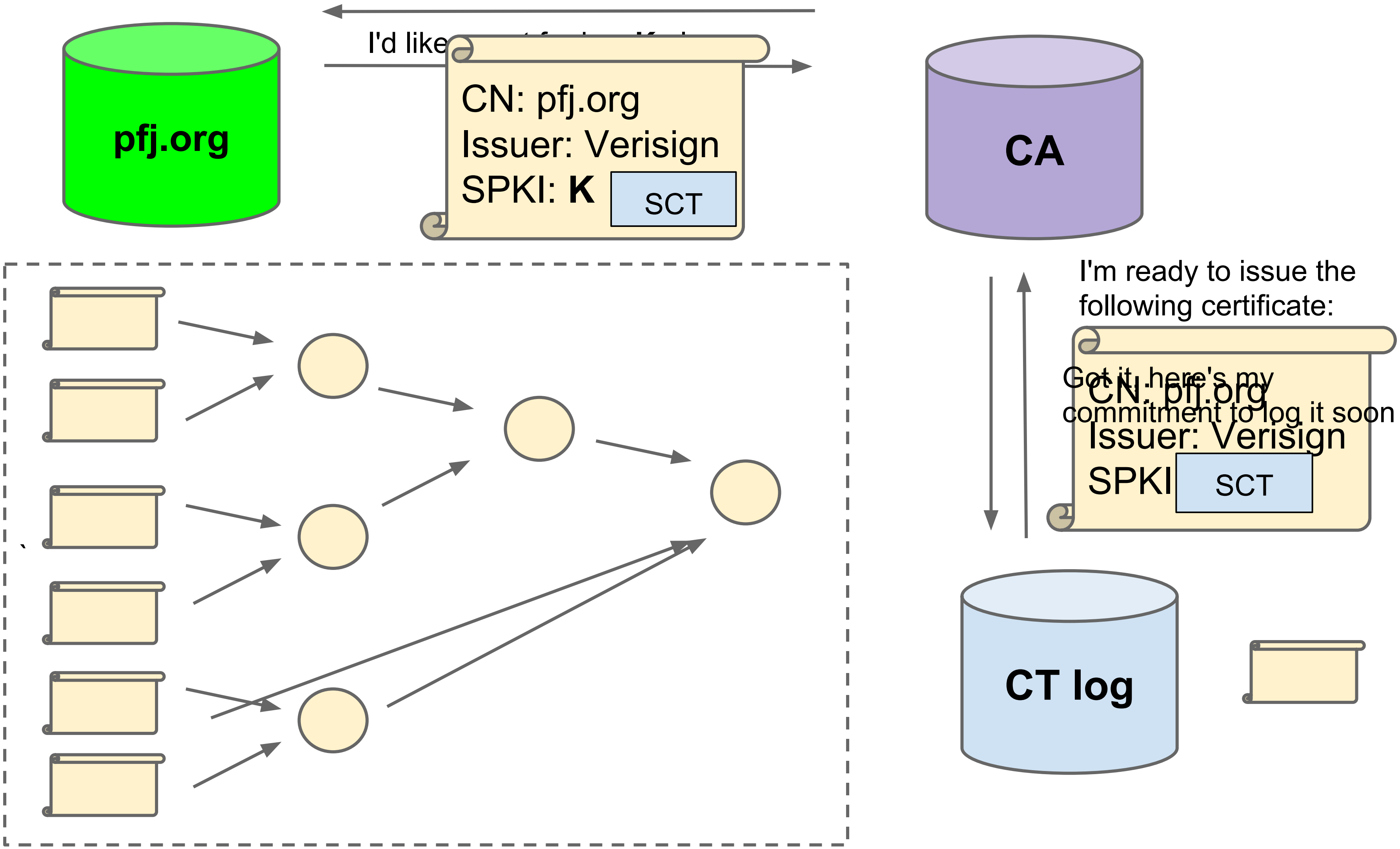
Oh my god,
it's full of certs...

Certificate Transparency (CT)

- Laurie, Langley, Käsper 2013
    - IETF experimental draft

- Enter every issued cert in a global log
- CT log is weakly trusted
    - Publicly verifiable
    - Append-only
- Relied on for availability, fork consistency
- Certs include "Signed certificate timestamp"
    - This is all clients check!
- Mis-issued certs detectable by scans

Certificate Transparency logging



pfj.org

I'd like

CN: pfj.org
Issuer: Verisign
SPKI: **K**
SCT

CA

I'm ready to issue the following certificate:

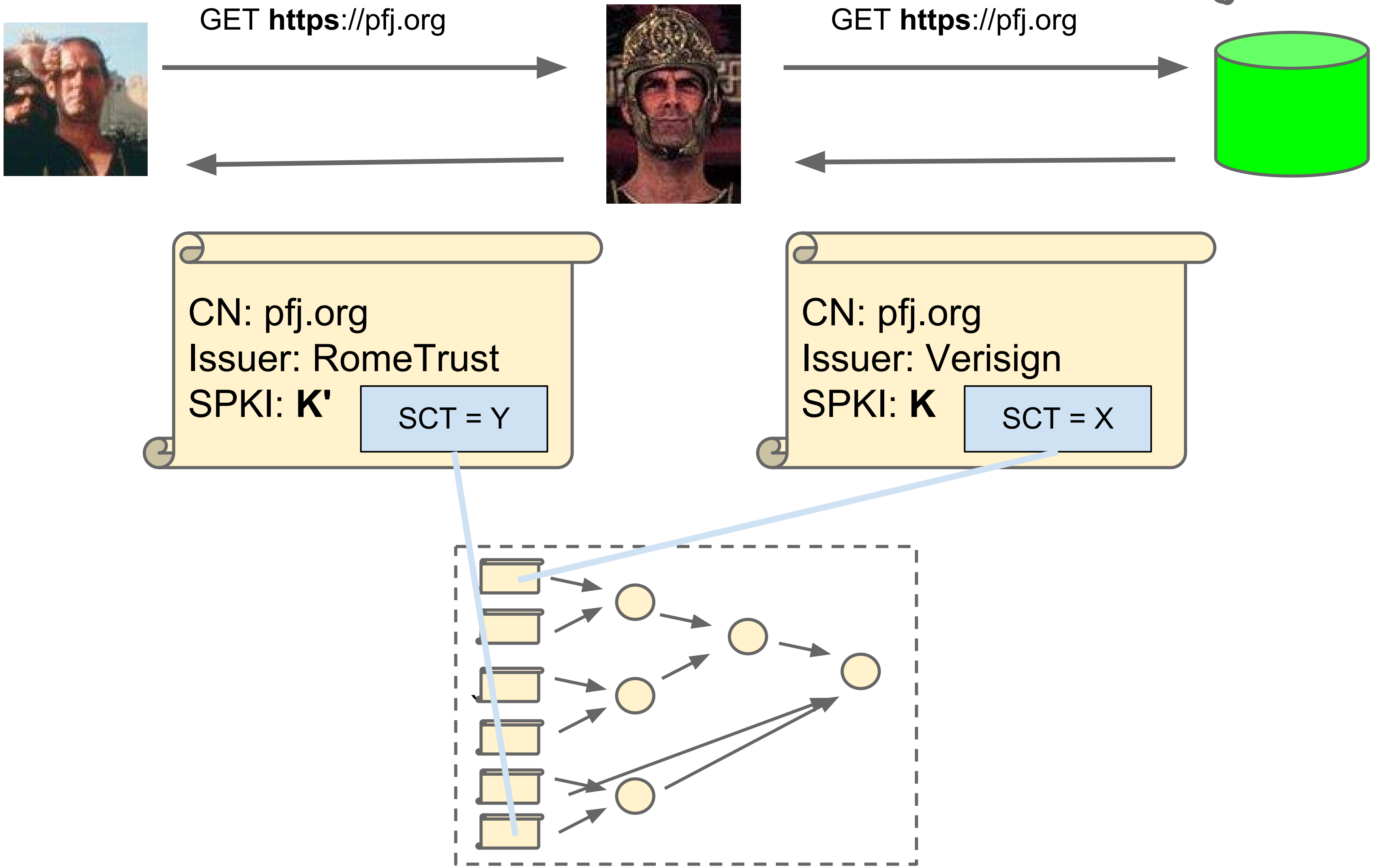
CN: pfj.org
Issuer: Verisign
SPKI
SCT

Got it, here's my commitment to log it soon

CT log

MITM attacks under CT

Something's not right..



GET **https**://pfj.org

GET **https**://pfj.org

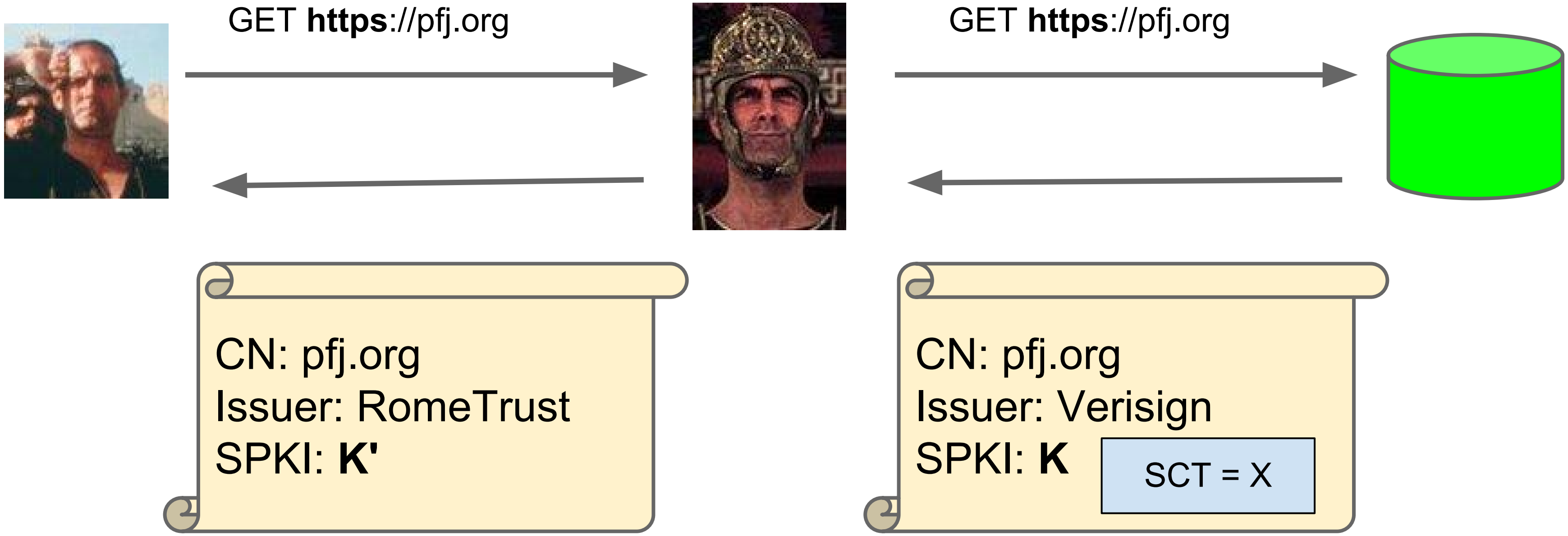
CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

SCT = Y

CN: pfj.org
Issuer: Verisign
SPKI: **K**

SCT = X

CT downgrade attacks



GET **https**://pfj.org

GET **https**://pfj.org

CN: pfj.org
Issuer: RomeTrust
SPKI: **K'**

CN: pfj.org
Issuer: Verisign
SPKI: **K**

SCT = X

Enhanced Certificate Transparency

- Ryan 2014

- Idea: log maintains a second tree
  - Certs in lexicographic order by domain
  - Order by insertion date
- Can query for most recent cert
- Revocation highly efficient

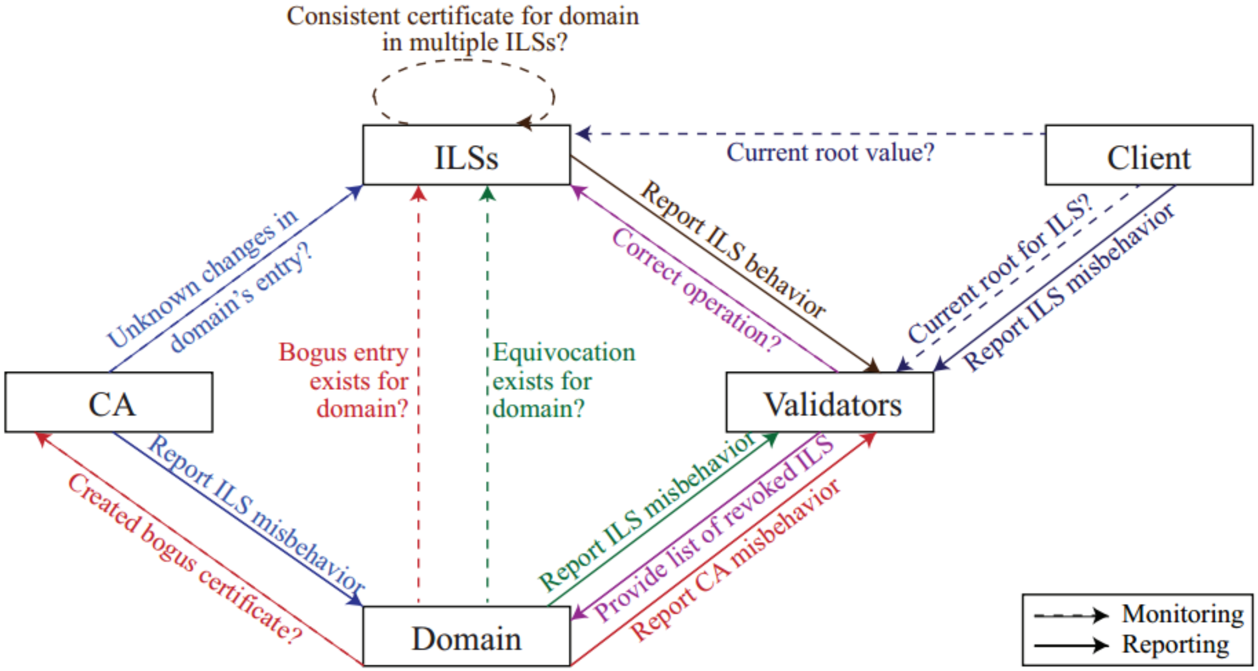
Sovereign Keys

- Eckersley 2011
- Elements of:
  - Certificate Transparency
  - TACK
  - Tor hidden services

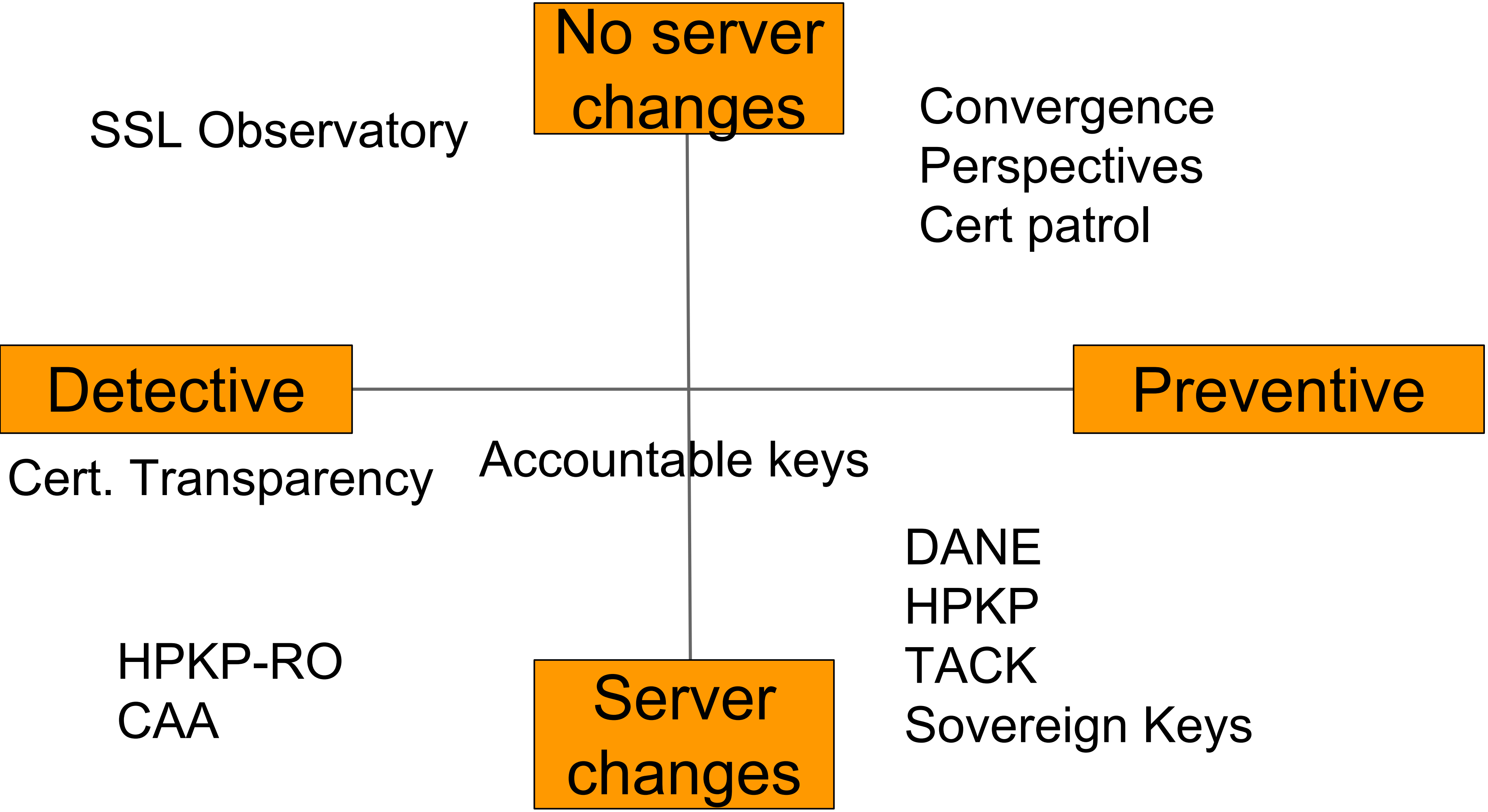
ELECTRONIC FRONTIER FOUNDATION
DEFENDING YOUR RIGHTS IN THE DIGITAL WORLD

Accountable Key Infrastructure

- Kim, Huang, Perrig, Jackson, Gligor, 2011
- Transparency plus a whole lot more



Proposals to deal with rogue certs



No server changes

SSL Observatory

Convergence
Perspectives
Cert patrol

Detective

Preventive

Cert. Transparency

Accountable keys

DANE
HPKP
TACK
Sovereign Keys

HPKP-RO
CAA

Server changes

5 predictions for the next 5 years

- CAs will not go away
- Multiple security protocols deployed
  - At least HPKP & CT
- Preload/link/continuity paradigm will solidify
  - Policy specifications may merge
- Web hubs will develop into security notaries
- Perfect Forward Secrecy hits mainstream

Big-picture questions

- Whom do we have to trust?

- Can we change who we have to trust?
  - **Trust agility**

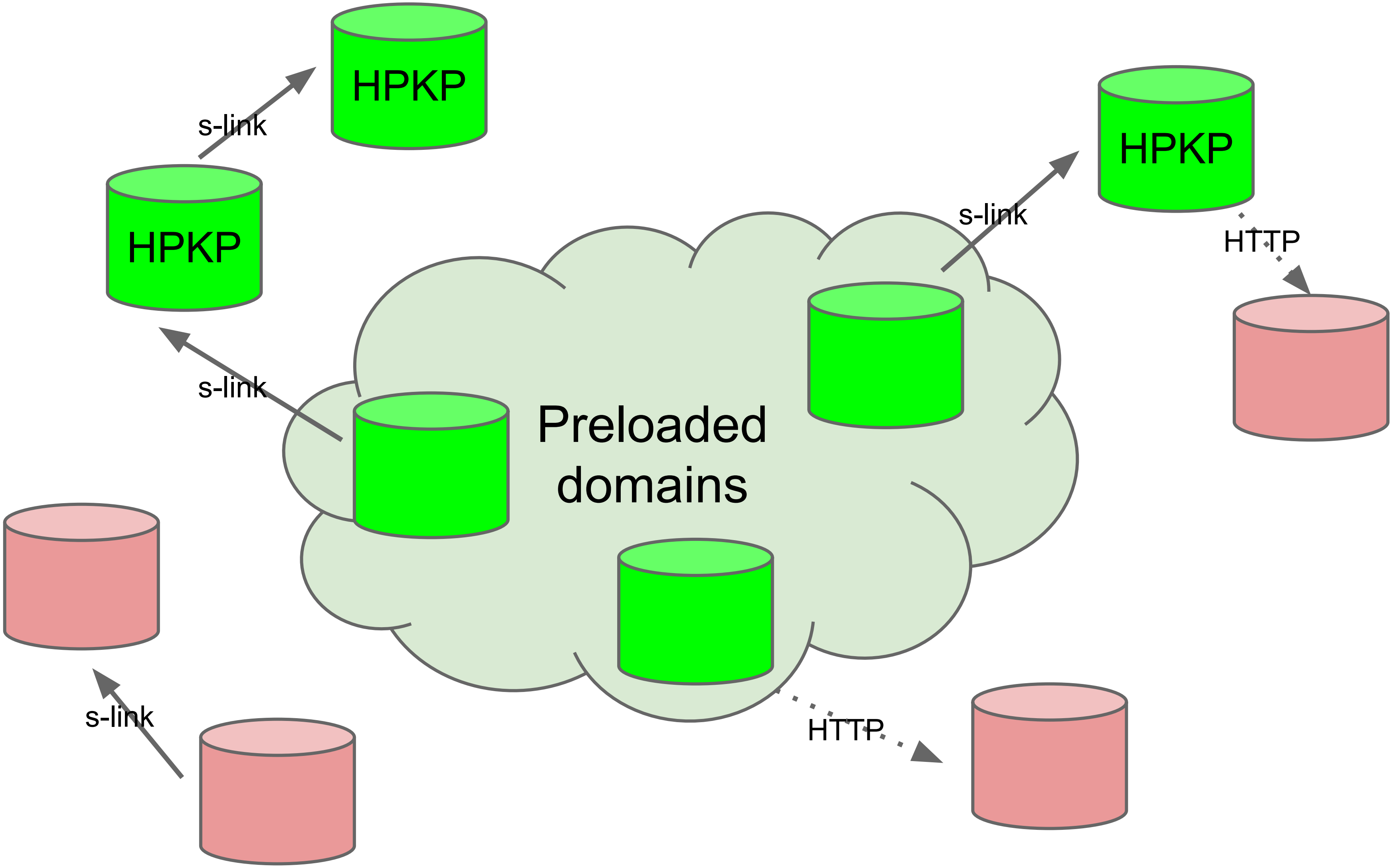
- Can users tell whom they're trusting?
  - **Trust affordance**

Certificate Transparency questions

- How many logs will be run?
  - Can we kill logs?

- Security with <100% CA adoption?

The end-to-end picture


HPKP
s-link
HPKP
HPKP
s-link
HTTP
s-link
Preloaded domains
s-link
HTTP