CMSC414 Computer and Network Security

How Crypto Fails in Practice

Yizheng Chen | University of Maryland
surrealyz.github.io

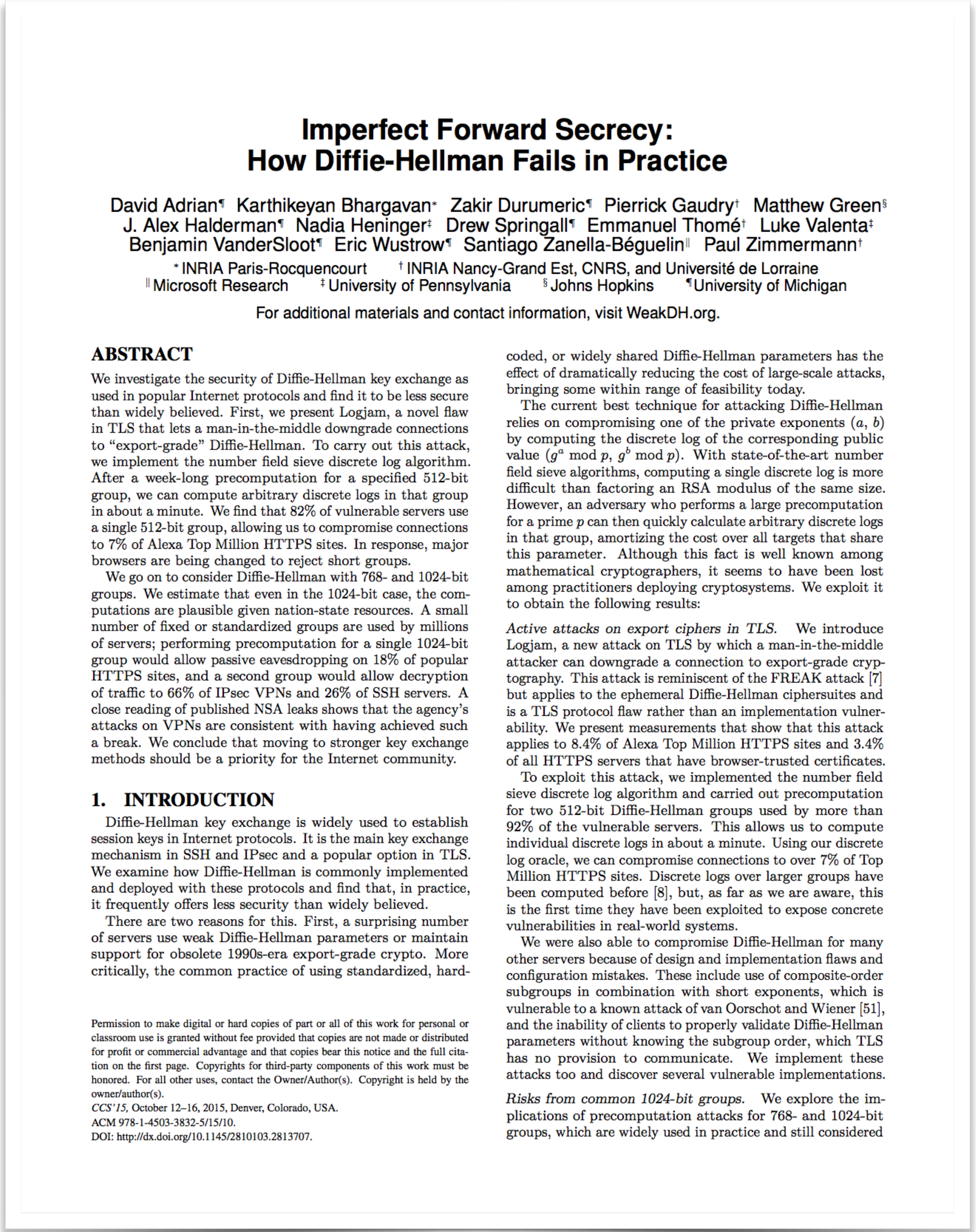
Apr 9, 2024

Credits: original slides from Dave Levin

Announcement

- Project 3 Deadline extended to Thursday, April 11
- Submission: both source code and executable (See Piazza and ELMS announcement for details)
- Needs to work inside the VM
- All tasks will be graded
- **Transaction types to stdout; other content in *.out**

RELATED PAPERS



Imperfect Forward Secrecy:
How Diffie-Hellman Fails in Practice

David Adrian[¶] Karthikeyan Bhargavan[*] Zakir Durumeric[¶] Pierrick Gaudry[†] Matthew Green[§]
J. Alex Halderman[¶] Nadia Heninger[‡] Drew Springall[¶] Emmanuel Thomé[†] Luke Valenta[‡]
Benjamin VanderSloot[¶] Eric Wustrow[¶] Santiago Zanella-Béguelin[‖] Paul Zimmermann[†]

[*] INRIA Paris-Rocquencourt [†] INRIA Nancy-Grand Est, CNRS, and Université de Lorraine
[‖] Microsoft Research [‡] University of Pennsylvania [§] Johns Hopkins [¶] University of Michigan

For additional materials and contact information, visit WeakDH.org.

ABSTRACT

We investigate the security of Diffie-Hellman key exchange as used in popular Internet protocols and find it to be less secure than widely believed. First, we present Logjam, a novel flaw in TLS that lets a man-in-the-middle downgrade connections to "export-grade" Diffie-Hellman. To carry out this attack, we implement the number field sieve discrete log algorithm. After a week-long precomputation for a specified 512-bit group, we can compute arbitrary discrete logs in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, allowing us to compromise connections to 7% of Alexa Top Million HTTPS sites. In response, major browsers are being changed to reject short groups.

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. We estimate that even in the 1024-bit case, the computations are plausible given nation-state resources. A small number of fixed or standardized groups are used by millions of servers; performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers. A close reading of published NSA leaks shows that the agency's attacks on VPNs are consistent with having achieved such a break. We conclude that moving to stronger key exchange methods should be a priority for the Internet community.

1. INTRODUCTION

Diffie-Hellman key exchange is widely used to establish session keys in Internet protocols. It is the main key exchange mechanism in SSH and IPsec and a popular option in TLS. We examine how Diffie-Hellman is commonly implemented and deployed with these protocols and find that, in practice, it frequently offers less security than widely believed.

There are two reasons for this. First, a surprising number of servers use weak Diffie-Hellman parameters or maintain support for obsolete 1990s-era export-grade crypto. More critically, the common practice of using standardized, hard-coded, or widely shared Diffie-Hellman parameters has the effect of dramatically reducing the cost of large-scale attacks, bringing some within range of feasibility today.

The current best technique for attacking Diffie-Hellman relies on compromising one of the private exponents (a, b) by computing the discrete log of the corresponding public value ($g^a \bmod p$, $g^b \bmod p$). With state-of-the-art number field sieve algorithms, computing a single discrete log is more difficult than factoring an RSA modulus of the same size. However, an adversary who performs a large precomputation for a prime p can then quickly calculate arbitrary discrete logs in that group, amortizing the cost over all targets that share this parameter. Although this fact is well known among mathematical cryptographers, it seems to have been lost among practitioners deploying cryptosystems. We exploit it to obtain the following results:

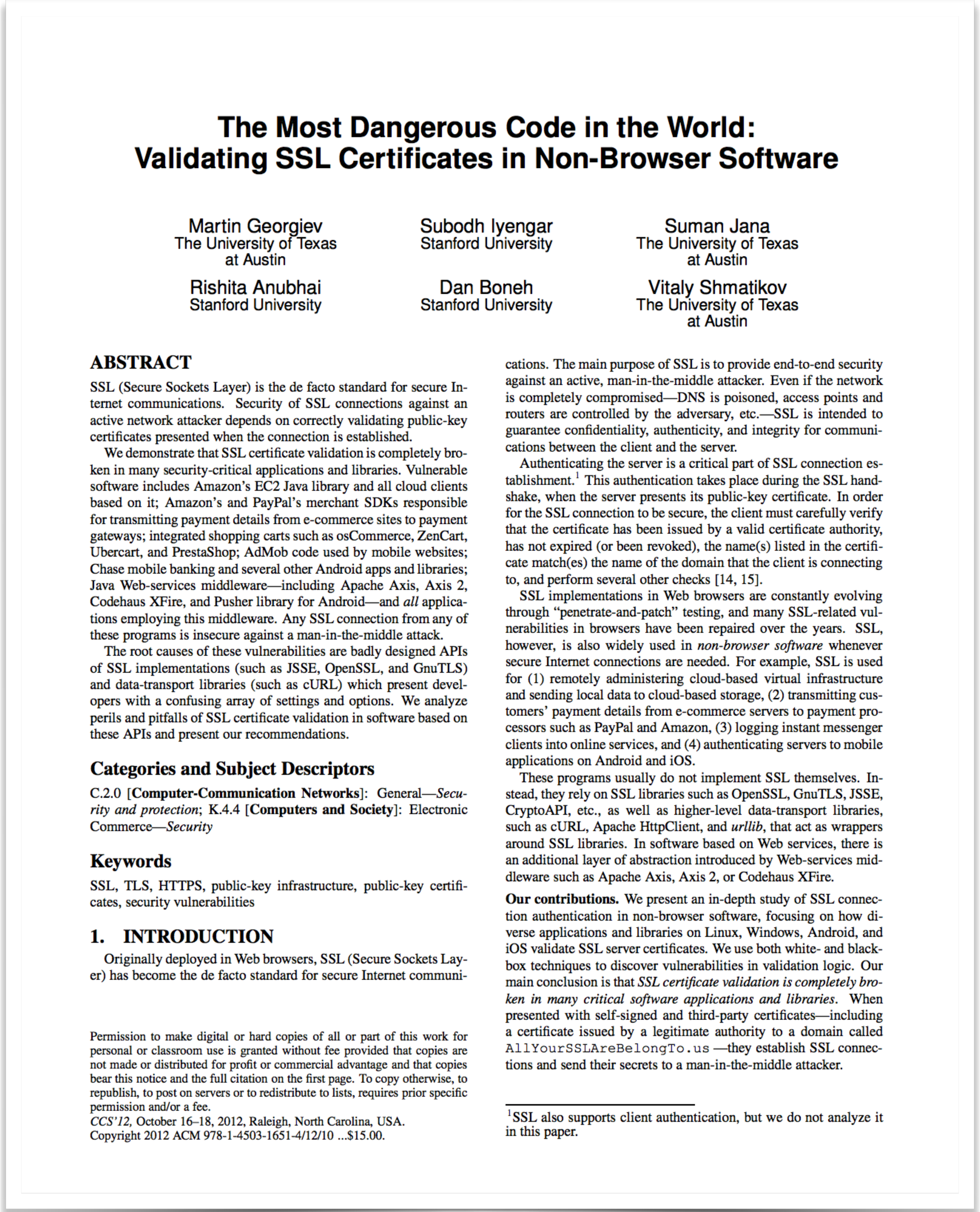
Active attacks on export ciphers in TLS. We introduce Logjam, a new attack on TLS by which a man-in-the-middle attacker can downgrade a connection to export-grade cryptography. This attack is reminiscent of the FREAK attack [7] but applies to the ephemeral Diffie-Hellman ciphersuites and is a TLS protocol flaw rather than an implementation vulnerability. We present measurements that show that this attack applies to 8.4% of Alexa Top Million HTTPS sites and 3.4% of all HTTPS servers that have browser-trusted certificates.

To exploit this attack, we implemented the number field sieve discrete log algorithm and carried out precomputation for two 512-bit Diffie-Hellman groups used by more than 92% of the vulnerable servers. This allows us to compute individual discrete logs in about a minute. Using our discrete log oracle, we can compromise connections to over 7% of Top Million HTTPS sites. Discrete logs over larger groups have been computed before [8], but, as far as we are aware, this is the first time they have been exploited to expose concrete vulnerabilities in real-world systems.

We were also able to compromise Diffie-Hellman for many other servers because of design and implementation flaws and configuration mistakes. These include use of composite-order subgroups in combination with short exponents, which is vulnerable to a known attack of van Oorschot and Wiener [51], and the inability of clients to properly validate Diffie-Hellman parameters without knowing the subgroup order, which TLS has no provision to communicate. We implement these attacks too and discover several vulnerable implementations.

Risks from common 1024-bit groups. We explore the implications of precomputation attacks for 768- and 1024-bit groups, which are widely used in practice and still considered

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
ACM 978-1-4503-3832-5/15/10.
DOI: http://dx.doi.org/10.1145/2810103.2813707.



The Most Dangerous Code in the World:
Validating SSL Certificates in Non-Browser Software

Martin Georgiev
The University of Texas
at Austin

Subodh Iyengar
Stanford University

Suman Jana
The University of Texas
at Austin

Rishita Anubhai
Stanford University

Dan Boneh
Stanford University

Vitaly Shmatikov
The University of Texas
at Austin

ABSTRACT

SSL (Secure Sockets Layer) is the de facto standard for secure Internet communications. Security of SSL connections against an active network attacker depends on correctly validating public-key certificates presented when the connection is established.

We demonstrate that SSL certificate validation is completely broken in many security-critical applications and libraries. Vulnerable software includes Amazon's EC2 Java library and all cloud clients based on it; Amazon's and PayPal's merchant SDKs responsible for transmitting payment details from e-commerce sites to payment gateways; integrated shopping carts such as osCommerce, ZenCart, Ubercart, and PrestaShop; AdMob code used by mobile websites; Chase mobile banking and several other Android apps and libraries; Java Web-services middleware—including Apache Axis, Axis 2, Codehaus XFire, and Pusher library for Android—and *all* applications employing this middleware. Any SSL connection from any of these programs is insecure against a man-in-the-middle attack.

The root causes of these vulnerabilities are badly designed APIs of SSL implementations (such as JSSE, OpenSSL, and GnuTLS) and data-transport libraries (such as cURL) which present developers with a confusing array of settings and options. We analyze perils and pitfalls of SSL certificate validation in software based on these APIs and present our recommendations.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*

Keywords

SSL, TLS, HTTPS, public-key infrastructure, public-key certificates, security vulnerabilities

1. INTRODUCTION

Originally deployed in Web browsers, SSL (Secure Sockets Layer) has become the de facto standard for secure Internet communications. The main purpose of SSL is to provide end-to-end security against an active, man-in-the-middle attacker. Even if the network is completely compromised—DNS is poisoned, access points and routers are controlled by the adversary, etc.—SSL is intended to guarantee confidentiality, authenticity, and integrity for communications between the client and the server.

Authenticating the server is a critical part of SSL connection establishment.[1] This authentication takes place during the SSL handshake, when the server presents its public-key certificate. In order for the SSL connection to be secure, the client must carefully verify that the certificate has been issued by a valid certificate authority, has not expired (or been revoked), the name(s) listed in the certificate match(es) the name of the domain that the client is connecting to, and perform several other checks [14, 15].

SSL implementations in Web browsers are constantly evolving through "penetrate-and-patch" testing, and many SSL-related vulnerabilities in browsers have been repaired over the years. SSL, however, is also widely used in *non-browser software* whenever secure Internet connections are needed. For example, SSL is used for (1) remotely administering cloud-based virtual infrastructure and sending local data to cloud-based storage, (2) transmitting customers' payment details from e-commerce servers to payment processors such as PayPal and Amazon, (3) logging instant messenger clients into online services, and (4) authenticating servers to mobile applications on Android and iOS.

These programs usually do not implement SSL themselves. Instead, they rely on SSL libraries such as OpenSSL, GnuTLS, JSSE, CryptoAPI, etc., as well as higher-level data-transport libraries, such as cURL, Apache HttpClient, and *urllib*, that act as wrappers around SSL libraries. In software based on Web services, there is an additional layer of abstraction introduced by Web-services middleware such as Apache Axis, Axis 2, or Codehaus XFire.

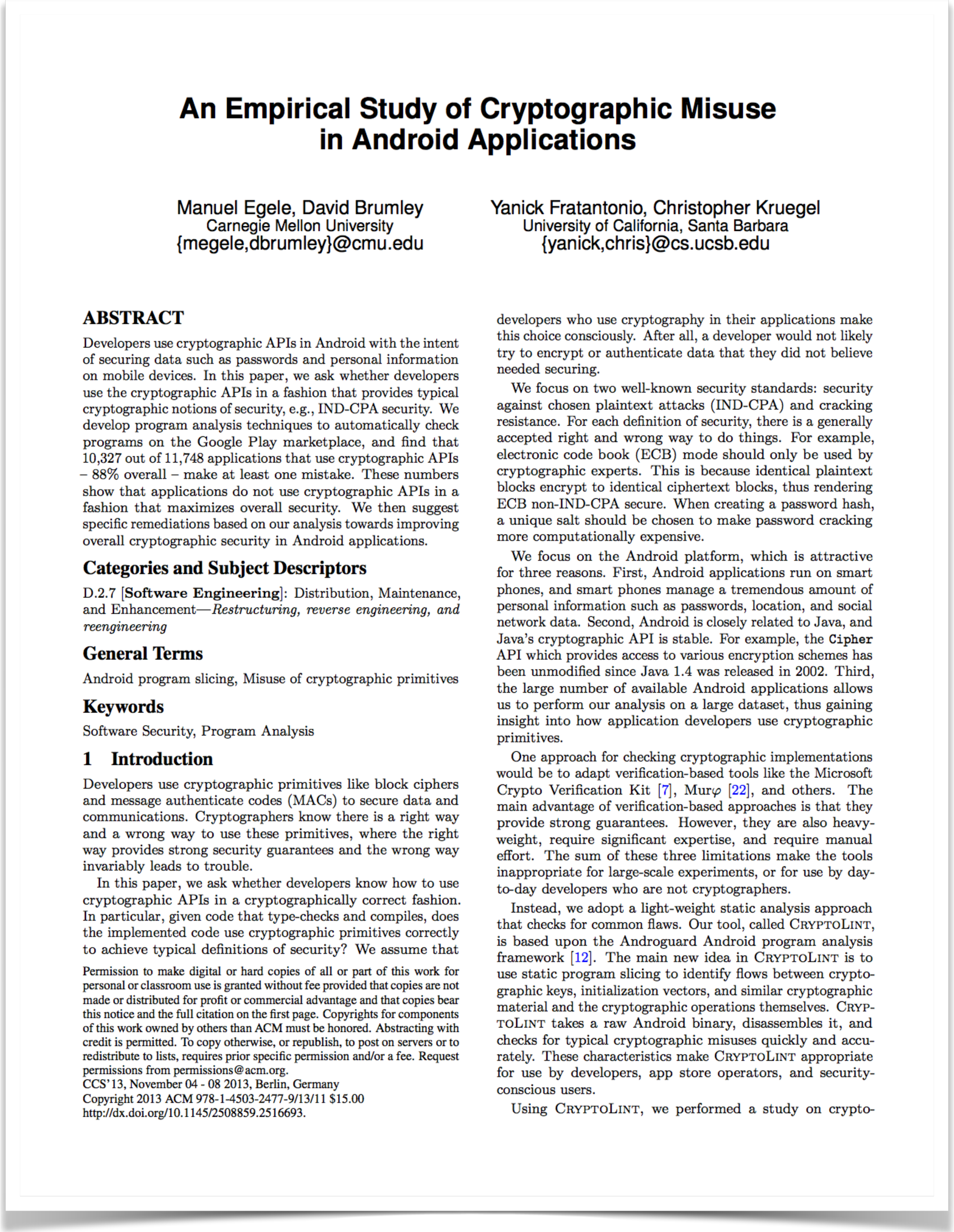
Our contributions. We present an in-depth study of SSL connection authentication in non-browser software, focusing on how diverse applications and libraries on Linux, Windows, Android, and iOS validate SSL server certificates. We use both white- and black-box techniques to discover vulnerabilities in validation logic. Our main conclusion is that *SSL certificate validation is completely broken in many critical software applications and libraries.* When presented with self-signed and third-party certificates—including a certificate issued by a legitimate authority to a domain called `AllYourSSLAreBelongTo.us` —they establish SSL connections and send their secrets to a man-in-the-middle attacker.

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.

---

[1] SSL also supports client authentication, but we do not analyze it in this paper.

POOR PROGRAMING



An Empirical Study of Cryptographic Misuse in Android Applications

Manuel Egele, David Brumley
Carnegie Mellon University
{megele,dbrumley}@cmu.edu

Yanick Fratantonio, Christopher Kruegel
University of California, Santa Barbara
{yanick,chris}@cs.ucsb.edu

ABSTRACT

Developers use cryptographic APIs in Android with the intent of securing data such as passwords and personal information on mobile devices. In this paper, we ask whether developers use the cryptographic APIs in a fashion that provides typical cryptographic notions of security, e.g., IND-CPA security. We develop program analysis techniques to automatically check programs on the Google Play marketplace, and find that 10,327 out of 11,748 applications that use cryptographic APIs – 88% overall – make at least one mistake. These numbers show that applications do not use cryptographic APIs in a fashion that maximizes overall security. We then suggest specific remediations based on our analysis towards improving overall cryptographic security in Android applications.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms

Android program slicing, Misuse of cryptographic primitives

Keywords

Software Security, Program Analysis

1 Introduction

Developers use cryptographic primitives like block ciphers and message authenticate codes (MACs) to secure data and communications. Cryptographers know there is a right way and a wrong way to use these primitives, where the right way provides strong security guarantees and the wrong way invariably leads to trouble.

In this paper, we ask whether developers know how to use cryptographic APIs in a cryptographically correct fashion. In particular, given code that type-checks and compiles, does the implemented code use cryptographic primitives correctly to achieve typical definitions of security? We assume that developers who use cryptography in their applications make this choice consciously. After all, a developer would not likely try to encrypt or authenticate data that they did not believe needed securing.

We focus on two well-known security standards: security against chosen plaintext attacks (IND-CPA) and cracking resistance. For each definition of security, there is a generally accepted right and wrong way to do things. For example, electronic code book (ECB) mode should only be used by cryptographic experts. This is because identical plaintext blocks encrypt to identical ciphertext blocks, thus rendering ECB non-IND-CPA secure. When creating a password hash, a unique salt should be chosen to make password cracking more computationally expensive.

We focus on the Android platform, which is attractive for three reasons. First, Android applications run on smart phones, and smart phones manage a tremendous amount of personal information such as passwords, location, and social network data. Second, Android is closely related to Java, and Java's cryptographic API is stable. For example, the `Cipher` API which provides access to various encryption schemes has been unmodified since Java 1.4 was released in 2002. Third, the large number of available Android applications allows us to perform our analysis on a large dataset, thus gaining insight into how application developers use cryptographic primitives.

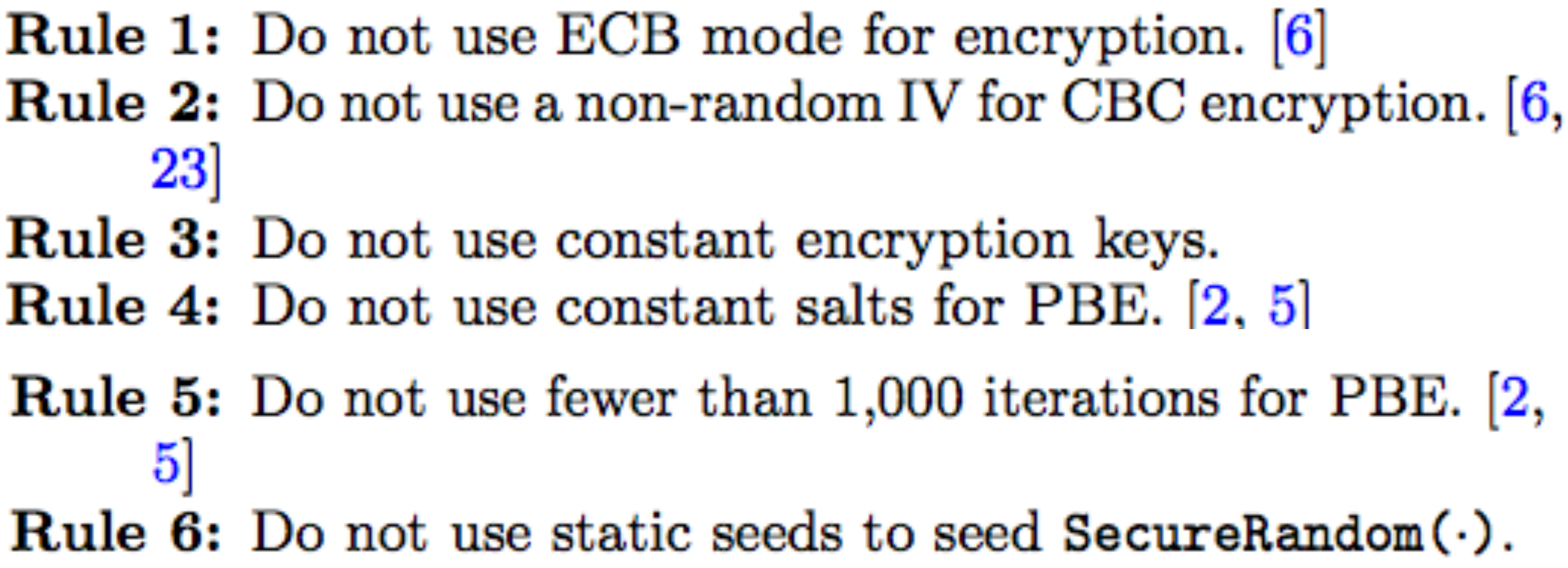
One approach for checking cryptographic implementations would be to adapt verification-based tools like the Microsoft Crypto Verification Kit [7], Murφ [22], and others. The main advantage of verification-based approaches is that they provide strong guarantees. However, they are also heavy-weight, require significant expertise, and require manual effort. The sum of these three limitations make the tools inappropriate for large-scale experiments, or for use by day-to-day developers who are not cryptographers.

Instead, we adopt a light-weight static analysis approach that checks for common flaws. Our tool, called CRYPTOLINT, is based upon the Androguard Android program analysis framework [12]. The main new idea in CRYPTOLINT is to use static program slicing to identify flows between cryptographic keys, initialization vectors, and similar cryptographic material and the cryptographic operations themselves. CRYPTOLINT takes a raw Android binary, disassembles it, and checks for typical cryptographic misuses quickly and accurately. These characteristics make CRYPTOLINT appropriate for use by developers, app store operators, and security-conscious users.

Using CRYPTOLINT, we performed a study on crypto-

CCS'13, November 04 - 08 2013, Berlin, Germany

http://dx.doi.org/10.1145/2508859.2516693.

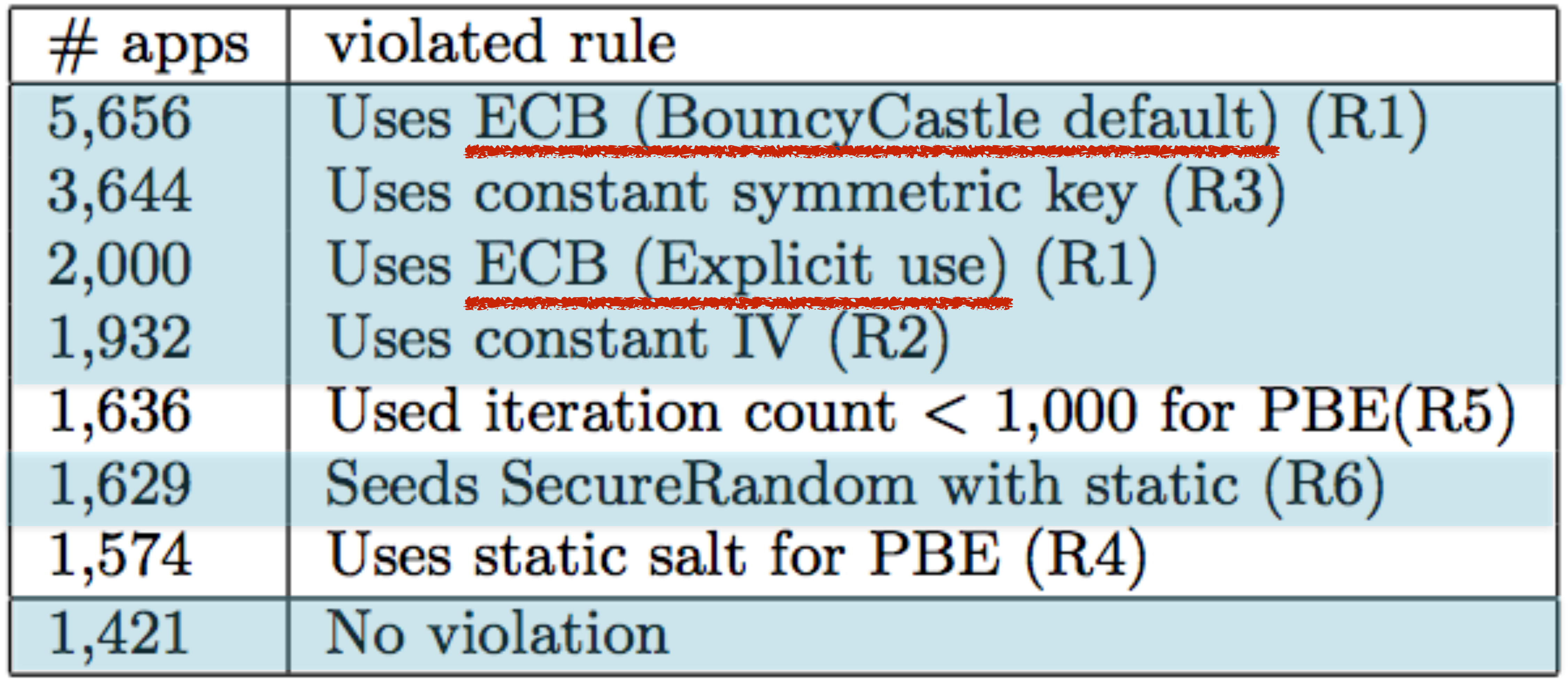


Rule 1: Do not use ECB mode for encryption. [6]
Rule 2: Do not use a non-random IV for CBC encryption. [6, 23]
Rule 3: Do not use constant encryption keys.
Rule 4: Do not use constant salts for PBE. [2, 5]
Rule 5: Do not use fewer than 1,000 iterations for PBE. [2, 5]
Rule 6: Do not use static seeds to seed `SecureRandom(·)`.

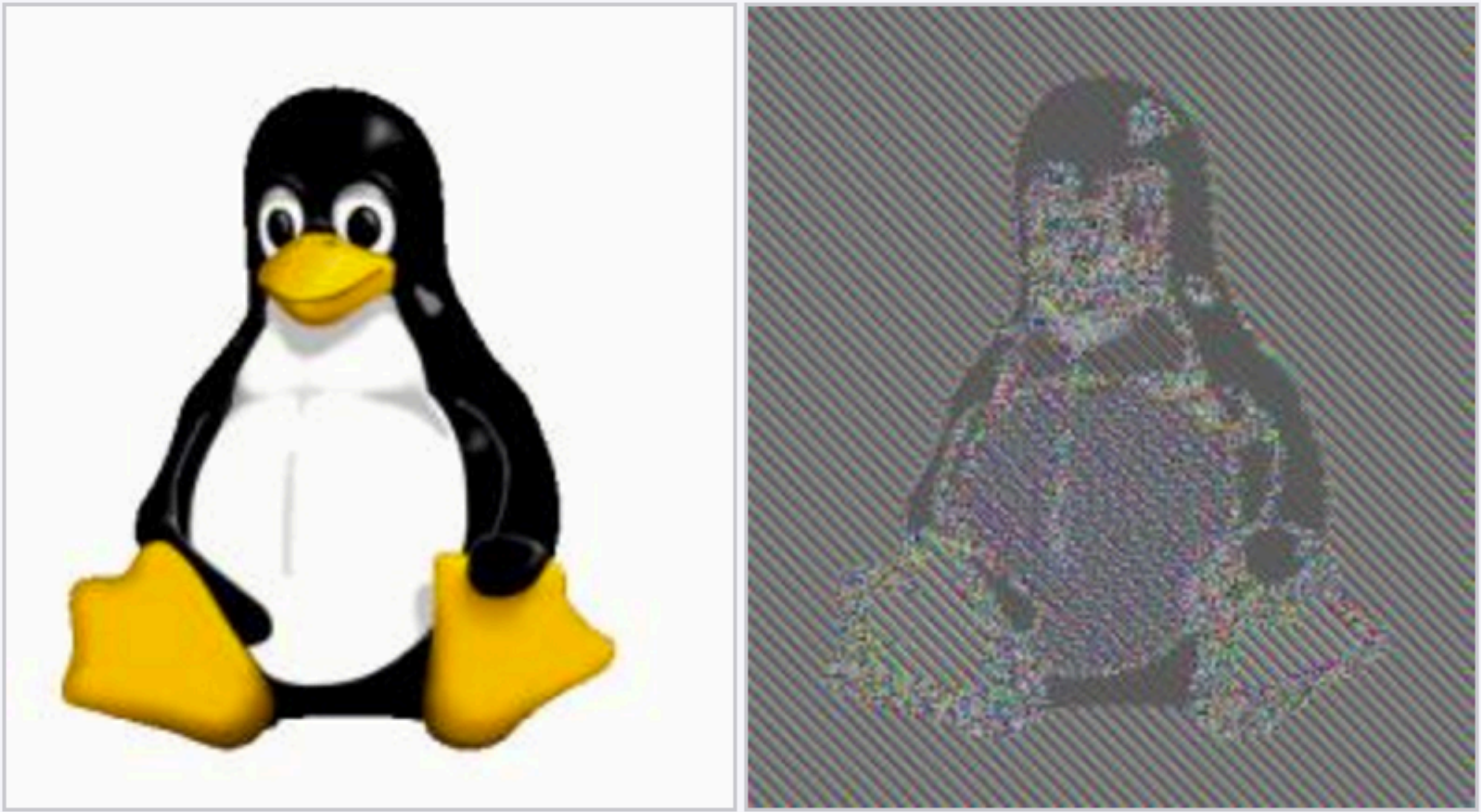
CryptoLint tool to perform static analysis on Android apps to detect how they are using crypto libraries

CRYPTO MISUSE IN ANDROID APPS

15,134 apps from Google play used crypto;
Analyzed **11,748** of them



	# apps	violated rule
48%	5,656	Uses ECB (BouncyCastle default) (R1)
31%	3,644	Uses constant symmetric key (R3)
17%	2,000	Uses ECB (Explicit use) (R1)
16%	1,932	Uses constant IV (R2)
	1,636	Used iteration count < 1,000 for PBE(R5)
14%	1,629	Seeds SecureRandom with static (R6)
	1,574	Uses static salt for PBE (R4)
12%	1,421	No violation



Original image

Encrypted using ECB mode

NEVER use ECB
(but over 50% of Android apps do)

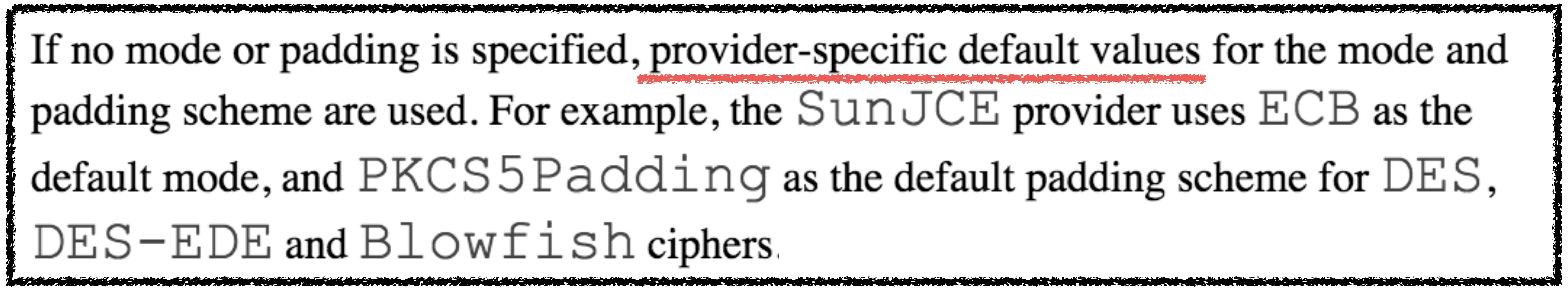
BOUNCYCASTLE DEFAULTS

- BouncyCastle is a library that conforms to Java's `Cipher` interface:

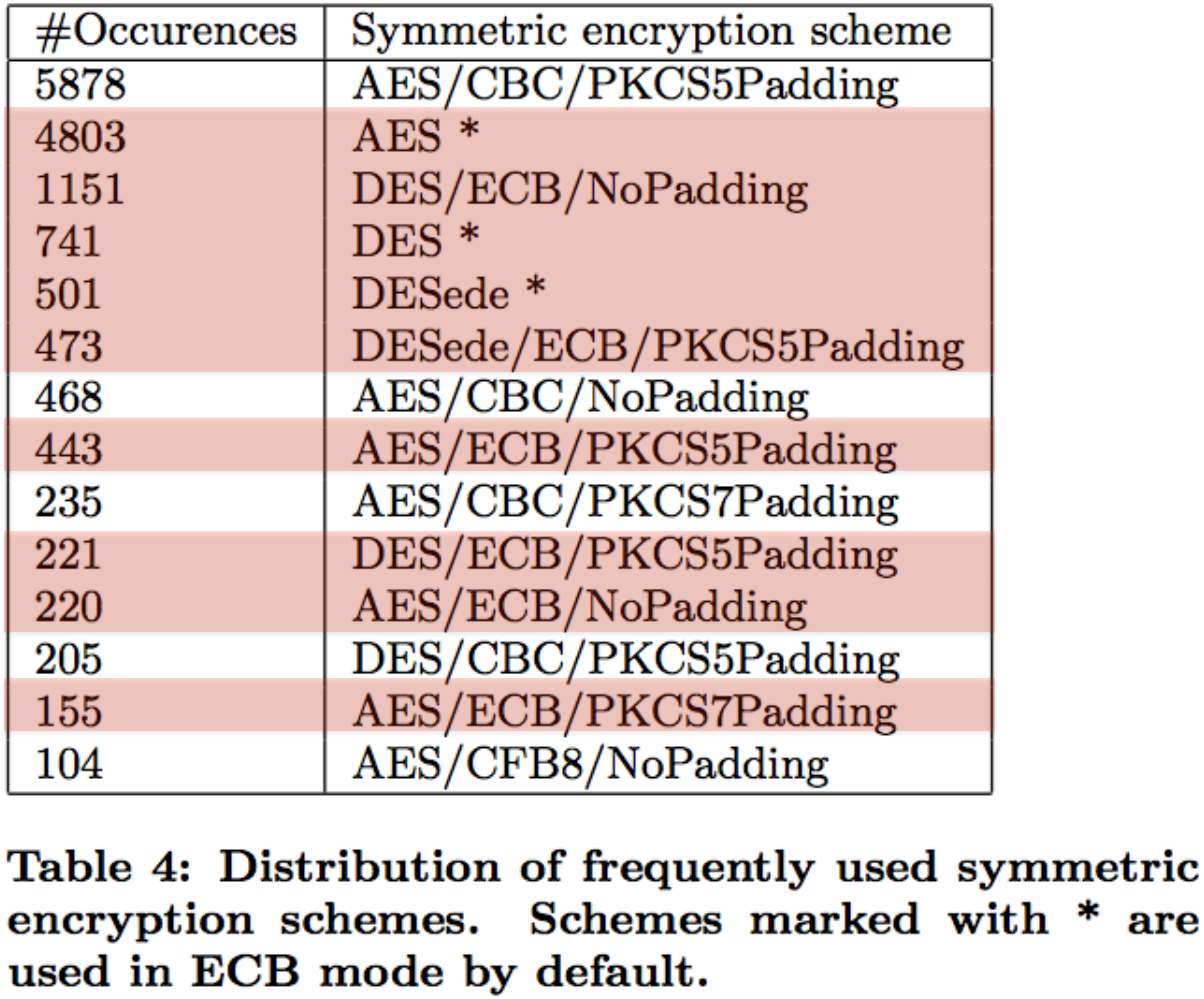
```
Cipher c =
   Cipher.getInstance("AES/CBC/PKCS5Padding");

// Ultimately end up wrapping a ByteArrayOutputStream
// in a CipherOutputStream
```

- Java documentation specifies:



If no mode or padding is specified, provider-specific default values for the mode and padding scheme are used. For example, the `SunJCE` provider uses `ECB` as the default mode, and `PKCS5Padding` as the default padding scheme for `DES`, `DES-EDE` and `Blowfish` ciphers.

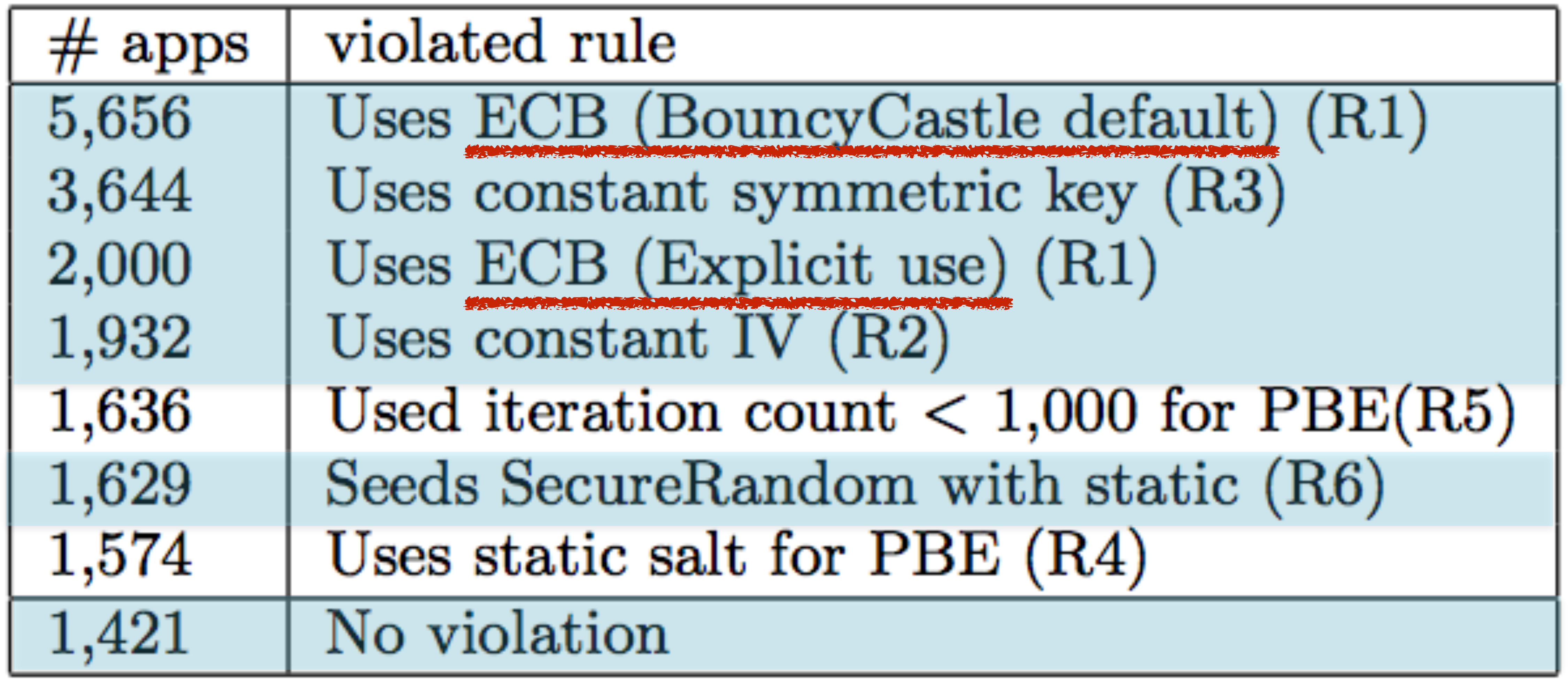


#Occurences	Symmetric encryption scheme
5878	AES/CBC/PKCS5Padding
4803	AES *
1151	DES/ECB/NoPadding
741	DES *
501	DESede *
473	DESede/ECB/PKCS5Padding
468	AES/CBC/NoPadding
443	AES/ECB/PKCS5Padding
235	AES/CBC/PKCS7Padding
221	DES/ECB/PKCS5Padding
220	AES/ECB/NoPadding
205	DES/CBC/PKCS5Padding
155	AES/ECB/PKCS7Padding
104	AES/CFB8/NoPadding

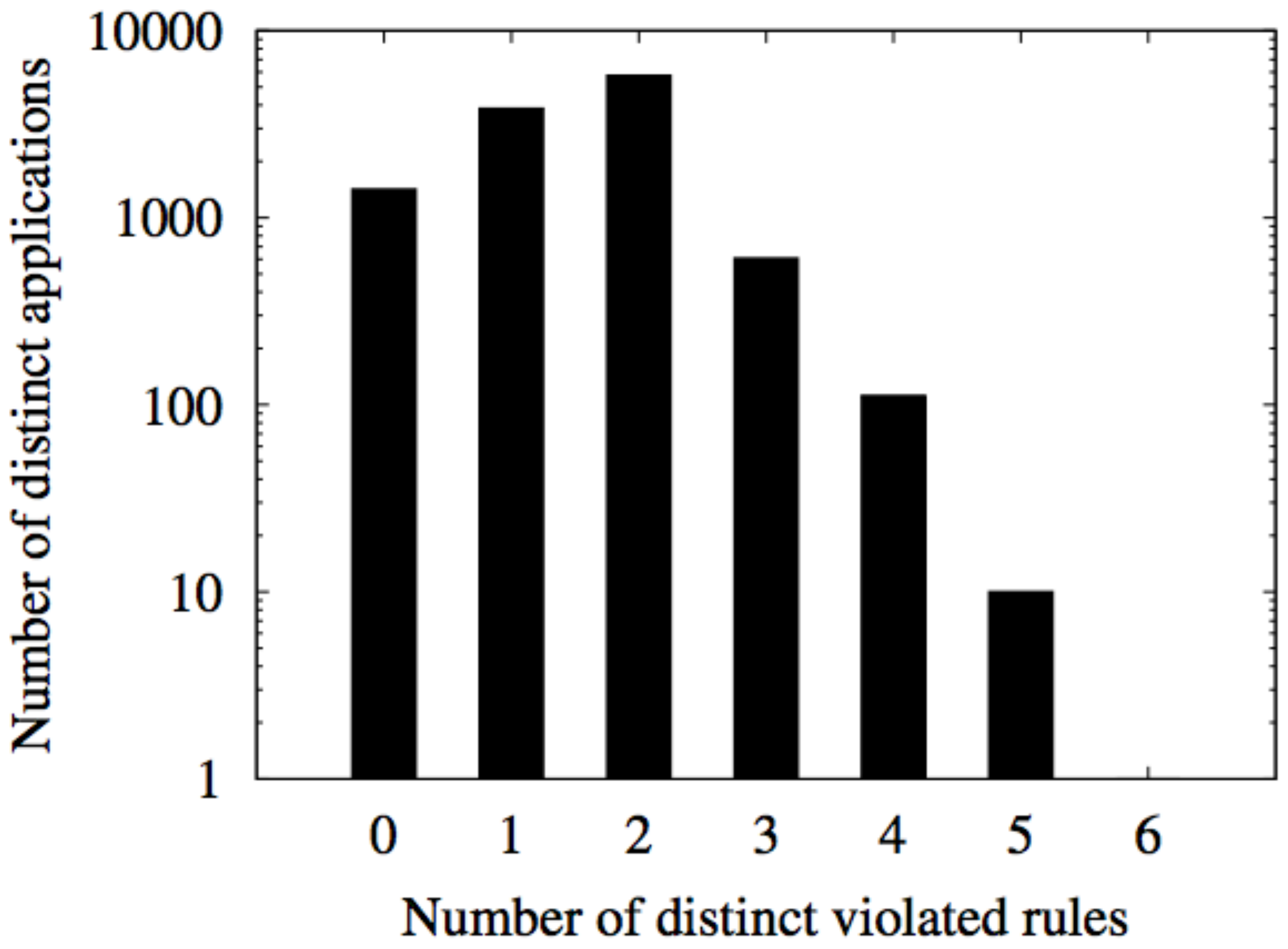
Table 4: Distribution of frequently used symmetric encryption schemes. Schemes marked with * are used in ECB mode by default.

CRYPTO MISUSE IN ANDROID APPS

15,134 apps from Google play used crypto;
Analyzed **11,748** of them



	# apps	violated rule
48%	5,656	Uses ECB (BouncyCastle default) (R1)
31%	3,644	Uses constant symmetric key (R3)
17%	2,000	Uses ECB (Explicit use) (R1)
16%	1,932	Uses constant IV (R2)
	1,636	Used iteration count < 1,000 for PBE(R5)
14%	1,629	Seeds SecureRandom with static (R6)
	1,574	Uses static salt for PBE (R4)
12%	1,421	No violation



A failure of the programmers to **know the tools** they use

A failure of library writers to **provide safe defaults**

MISUSING CRYPTO

Avoid shooting yourself in the foot:

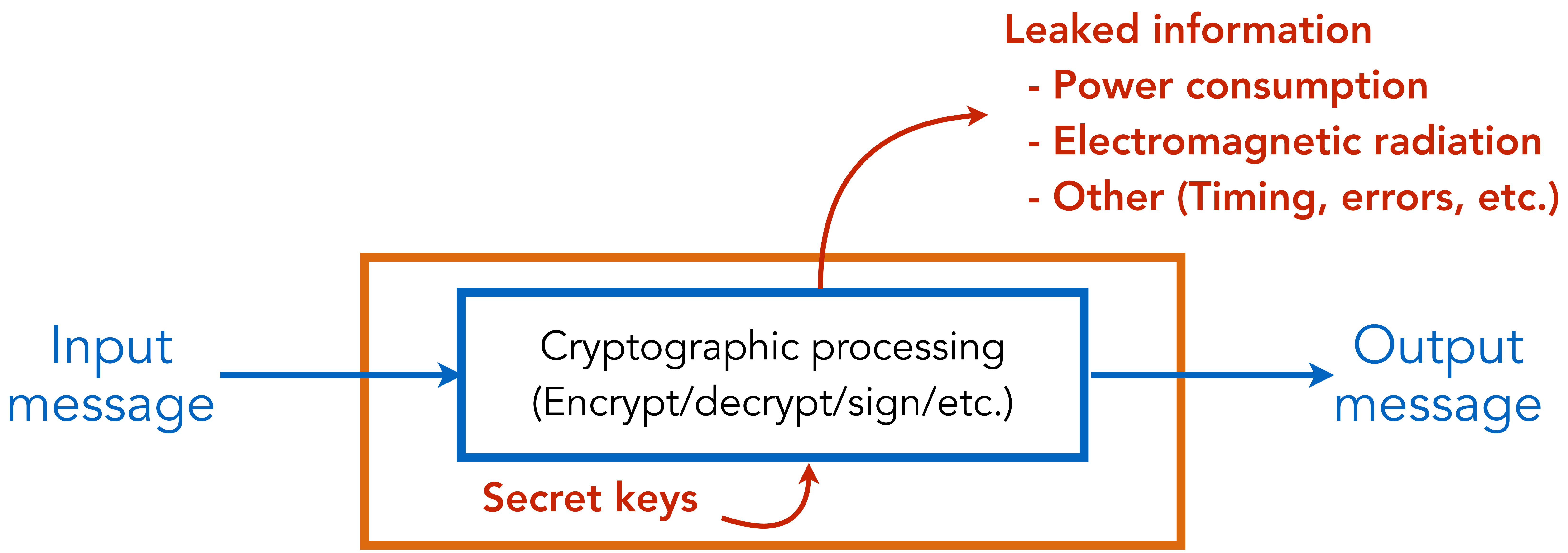
- Do not **roll your own** cryptographic mechanisms
  - Takes peer review
  - Apply Kerkhoff's principle
- Do not ***misuse*** existing crypto
- Do not even ***implement*** the underlying crypto

WHY NOT IMPLEMENT AES/RSA YOURSELF?

- Not talking about creating a brand new crypto scheme, just implementing one that's already widely accepted and used.
- Kerkhoff's principle: these are all open standards; should be implementable.
- Potentially buggy/incorrect code, but so might be others' implementations (viz. OpenSSL bugs, poor defaults in Bouncy castles, etc.)
- So why not implement it yourself?

SIDE-CHANNEL ATTACKS

- Cryptography concerns the *theoretical* difficulty in breaking a cipher



Leaked information
- **Power consumption**
- **Electromagnetic radiation**
- **Other (Timing, errors, etc.)**

Input message

Cryptographic processing
(Encrypt/decrypt/sign/etc.)

Output message

Secret keys

- But what about the information that a particular *implementation* could leak?
  - Attacks based on these are "**side-channel attacks**"

SIMPLE POWER ANALYSIS (SPA)

- Interpret *power traces* taken during a cryptographic operation
- Simple power analysis can reveal the sequence of instructions executed

SPA ON DES

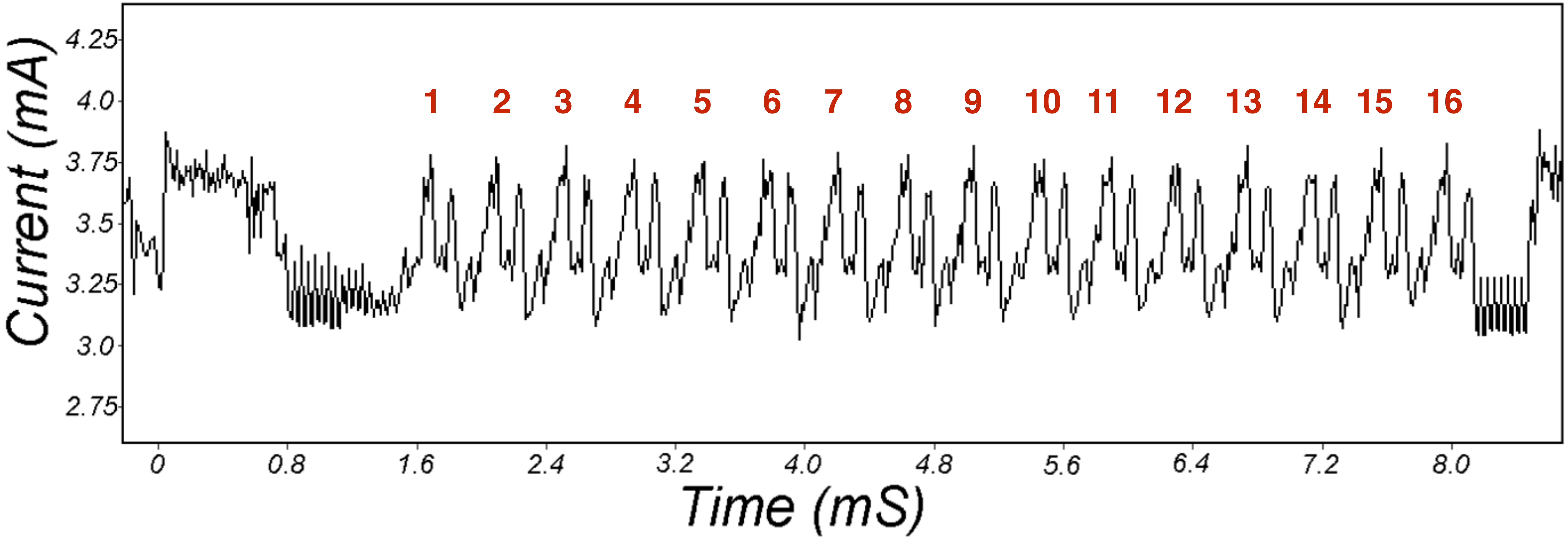


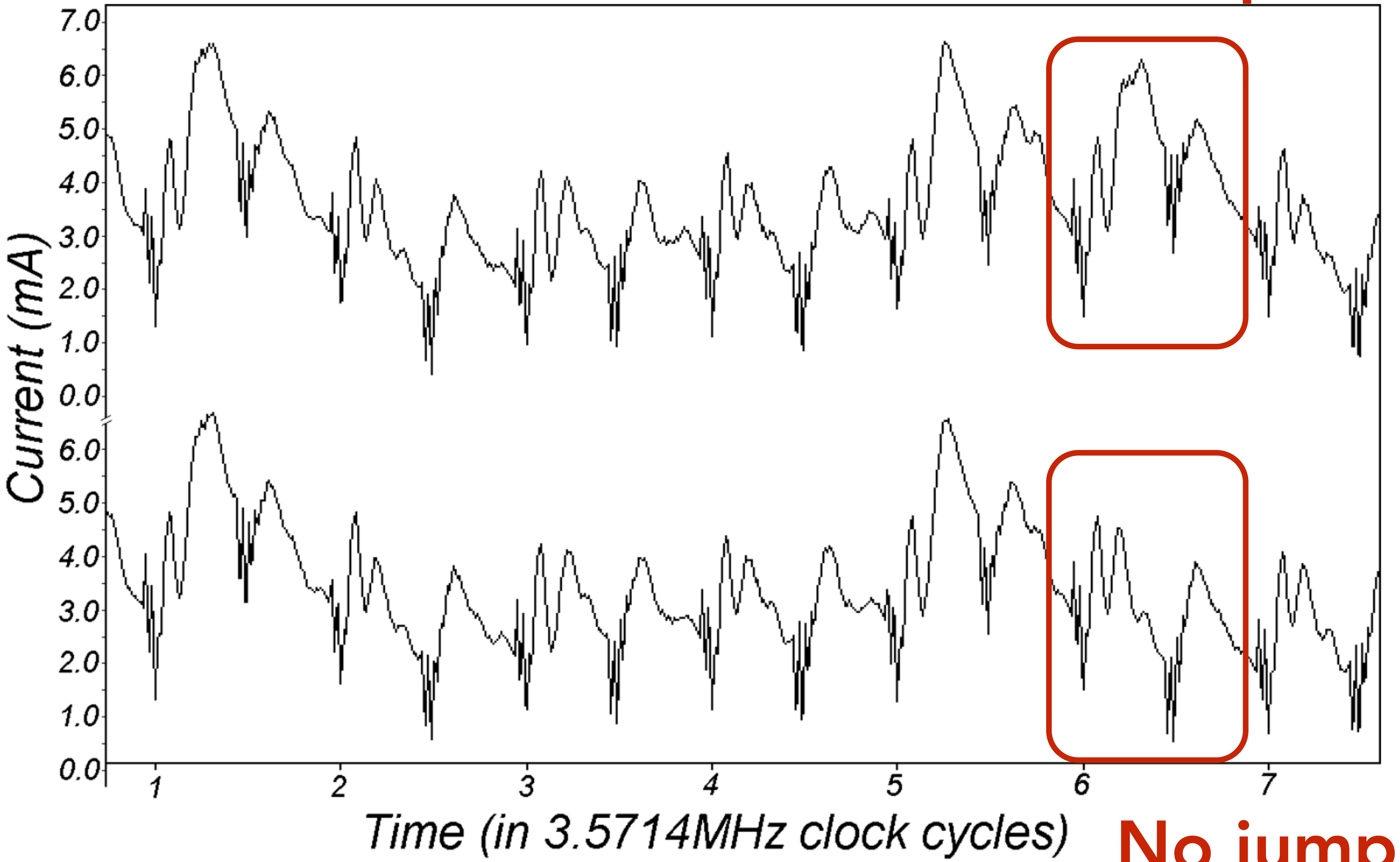
Figure 1: SPA trace showing an entire DES operation.

Overall operation clearly visible:

Can identify the **16 rounds of DES**

SPA ON DES

Jump taken



No jump taken

Figure 3: SPA trace showing individual clock cycles.

Specific **instructions** are also discernible

HIGH-LEVEL IDEA

```
HypotheticalEncrypt(msg, key) {
   for(int i=0; i < key.len(); i++) {
      if(key[i] == 0)
         // branch 0
      else
         // branch 1
    }
}
```

What if branch 0 had, e.g., a `jmp` that brand 1 didn’t?

What if branch 0
- took longer? (timing attacks)
- gave off more heat?
- made more noise?
- …

Implementation issue: If the execution path depends on the inputs (key/data), then *SPA can reveal keys*

DIFFERENTIAL POWER ANALYSIS (DPA)

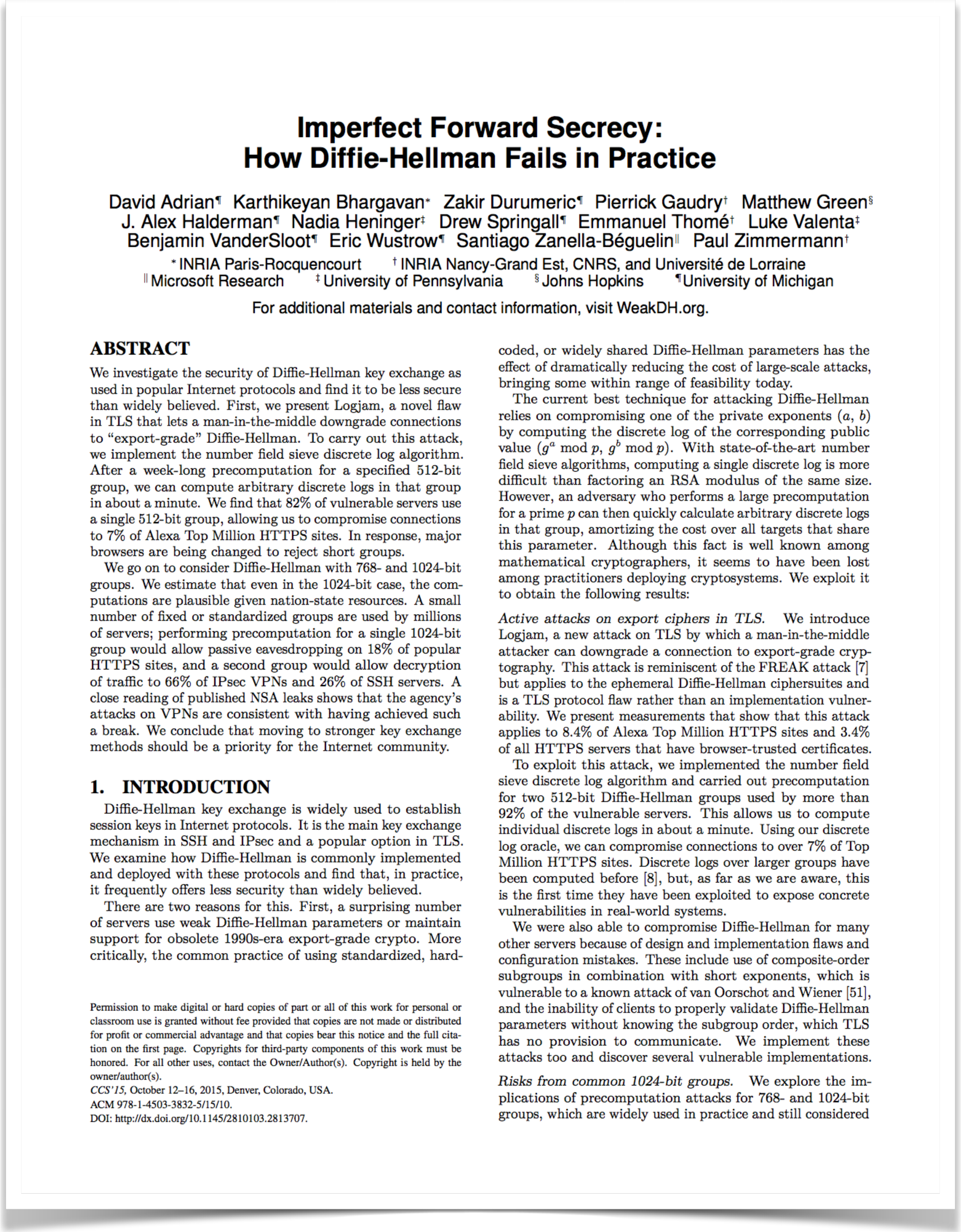
- SPA just visually inspects a single run
- DPA runs iteratively and reactively
  - Get multiple samples
  - Based on these, construct new plaintext messages as inputs, and repeat

MITIGATING SUCH ATTACKS

- Hide information by making the execution paths depend on the inputs as little as possible
  - Have to *give up some optimizations* that depend on particular bit values in keys
    - Some Chinese Remainder Theorem (CRT) optimizations permitted remote timing attacks on SSL servers

- The crypto community should seek to design cryptosystems under the assumption that some information is going to leak

POOR POLICIES



Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

David Adrian[¶] Karthikeyan Bhargavan[*] Zakir Durumeric[¶] Pierrick Gaudry[†] Matthew Green[§]
J. Alex Halderman[¶] Nadia Heninger[‡] Drew Springall[¶] Emmanuel Thomé[†] Luke Valenta[‡]
Benjamin VanderSloot[¶] Eric Wustrow[¶] Santiago Zanella-Béguelin[‖] Paul Zimmermann[†]
[*] INRIA Paris-Rocquencourt [†] INRIA Nancy-Grand Est, CNRS, and Université de Lorraine
[‖] Microsoft Research [‡] University of Pennsylvania [§] Johns Hopkins [¶] University of Michigan

For additional materials and contact information, visit WeakDH.org.

ABSTRACT

We investigate the security of Diffie-Hellman key exchange as used in popular Internet protocols and find it to be less secure than widely believed. First, we present Logjam, a novel flaw in TLS that lets a man-in-the-middle downgrade connections to "export-grade" Diffie-Hellman. To carry out this attack, we implement the number field sieve discrete log algorithm. After a week-long precomputation for a specified 512-bit group, we can compute arbitrary discrete logs in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, allowing us to compromise connections to 7% of Alexa Top Million HTTPS sites. In response, major browsers are being changed to reject short groups.

We go on to consider Diffie-Hellman with 768- and 1024-bit groups. We estimate that even in the 1024-bit case, the computations are plausible given nation-state resources. A small number of fixed or standardized groups are used by millions of servers; performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers. A close reading of published NSA leaks shows that the agency's attacks on VPNs are consistent with having achieved such a break. We conclude that moving to stronger key exchange methods should be a priority for the Internet community.

1. INTRODUCTION

Diffie-Hellman key exchange is widely used to establish session keys in Internet protocols. It is the main key exchange mechanism in SSH and IPsec and a popular option in TLS. We examine how Diffie-Hellman is commonly implemented and deployed with these protocols and find that, in practice, it frequently offers less security than widely believed.

There are two reasons for this. First, a surprising number of servers use weak Diffie-Hellman parameters or maintain support for obsolete 1990s-era export-grade crypto. More critically, the common practice of using standardized, hard-coded, or widely shared Diffie-Hellman parameters has the effect of dramatically reducing the cost of large-scale attacks, bringing some within range of feasibility today.

The current best technique for attacking Diffie-Hellman relies on compromising one of the private exponents (a, b) by computing the discrete log of the corresponding public value ($g^a \bmod p$, $g^b \bmod p$). With state-of-the-art number field sieve algorithms, computing a single discrete log is more difficult than factoring an RSA modulus of the same size. However, an adversary who performs a large precomputation for a prime p can then quickly calculate arbitrary discrete logs in that group, amortizing the cost over all targets that share this parameter. Although this fact is well known among mathematical cryptographers, it seems to have been lost among practitioners deploying cryptosystems. We exploit it to obtain the following results:

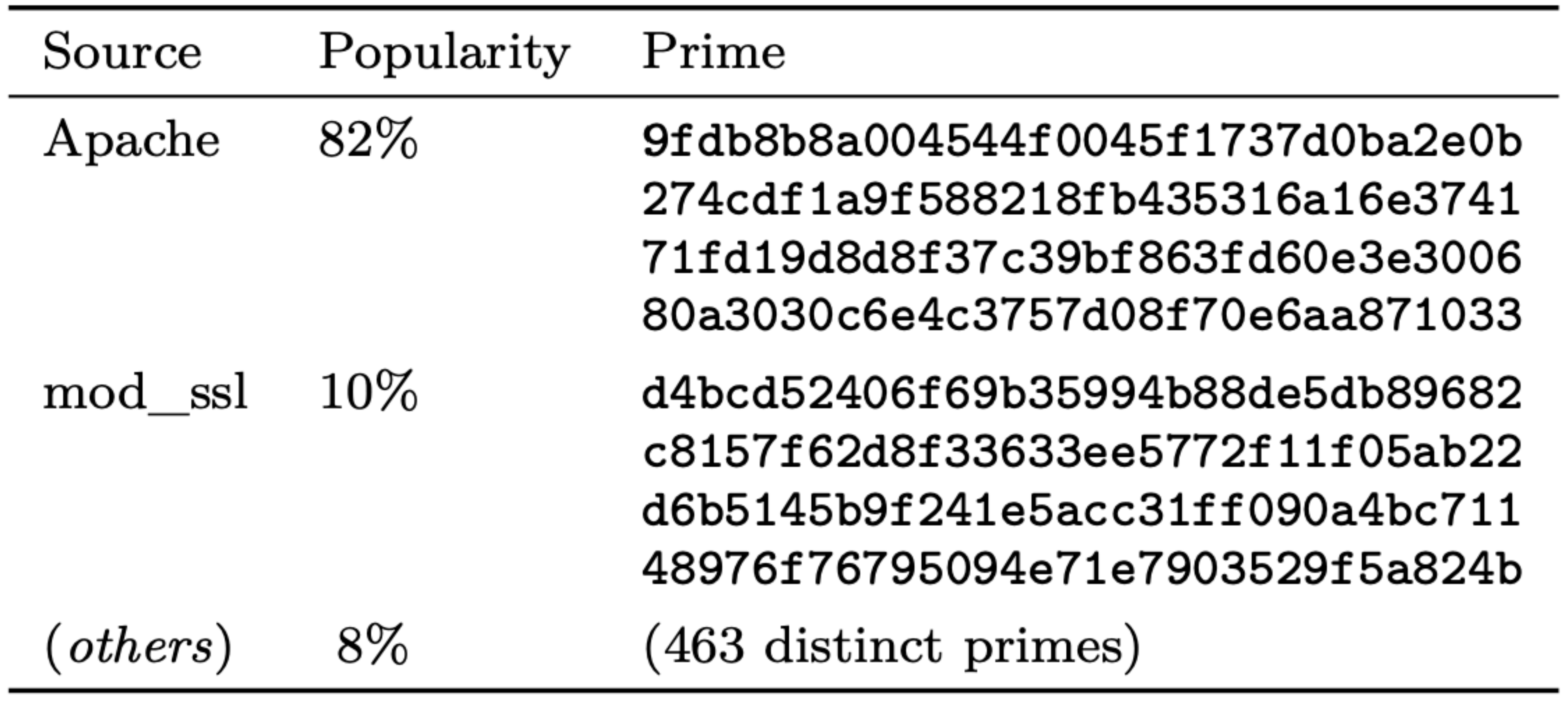
Active attacks on export ciphers in TLS. We introduce Logjam, a new attack on TLS by which a man-in-the-middle attacker can downgrade a connection to export-grade cryptography. This attack is reminiscent of the FREAK attack [7] but applies to the ephemeral Diffie-Hellman ciphersuites and is a TLS protocol flaw rather than an implementation vulnerability. We present measurements that show that this attack applies to 8.4% of Alexa Top Million HTTPS sites and 3.4% of all HTTPS servers that have browser-trusted certificates.

To exploit this attack, we implemented the number field sieve discrete log algorithm and carried out precomputation for two 512-bit Diffie-Hellman groups used by more than 92% of the vulnerable servers. This allows us to compute individual discrete logs in about a minute. Using our discrete log oracle, we can compromise connections to over 7% of Top Million HTTPS sites. Discrete logs over larger groups have been computed before [8], but, as far as we are aware, this is the first time they have been exploited to expose concrete vulnerabilities in real-world systems.

We were also able to compromise Diffie-Hellman for many other servers because of design and implementation flaws and configuration mistakes. These include use of composite-order subgroups in combination with short exponents, which is vulnerable to a known attack of van Oorschot and Wiener [51], and the inability of clients to properly validate Diffie-Hellman parameters without knowing the subgroup order, which TLS has no provision to communicate. We implement these attacks too and discover several vulnerable implementations.

Risks from common 1024-bit groups. We explore the implications of precomputation attacks for 768- and 1024-bit groups, which are widely used in practice and still considered

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
ACM 978-1-4503-3832-5/15/10.
DOI: http://dx.doi.org/10.1145/2810103.2813707.



Source	Popularity	Prime

Apache	82%	`9fdb8b8a004544f0045f1737d0ba2e0b`
		`274cdf1a9f588218fb435316a16e3741`
		`71fd19d8d8f37c39bf863fd60e3e3006`
		`80a3030c6e4c3757d08f70e6aa871033`

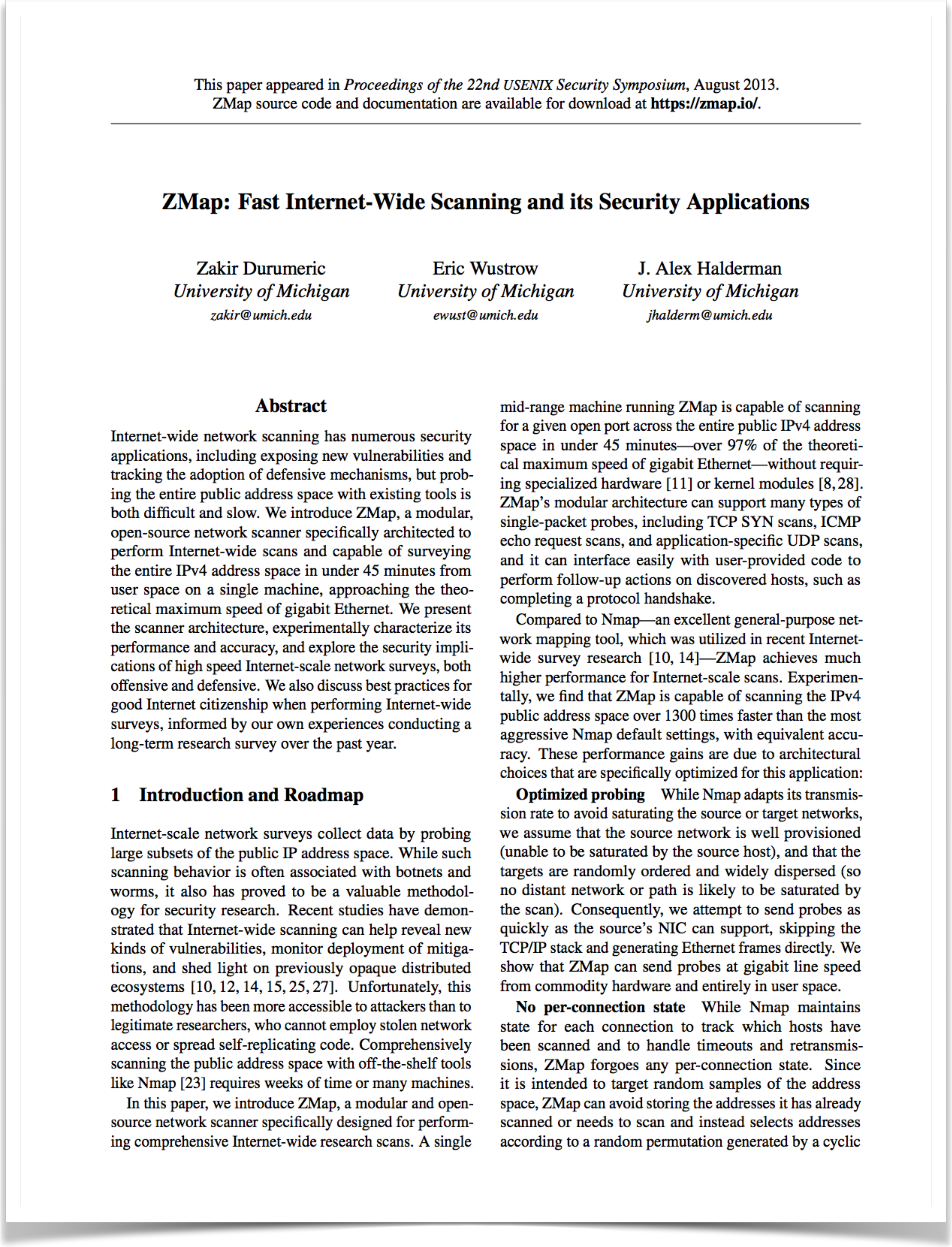
mod_ssl	10%	`d4bcd52406f69b35994b88de5db89682`
		`c8157f62d8f33633ee5772f11f05ab22`
		`d6b5145b9f241e5acc31ff090a4bc711`
		`48976f76795094e71e7903529f5a824b`

(*others*)	8%	(463 distinct primes)

Table 1: **Top 512-bit DH primes for TLS.** 8.4% of Alexa Top 1M HTTPS domains allow DHE_EXPORT, of which 92.3% use one of the two most popular primes, shown here.

*"After a week-long **precomputation** for a specified 512-bit group, we can compute arbitrary discrete logs in that group in about a minute. We find that 82% of vulnerable servers use a single 512-bit group, allowing us to compromise connections to 7% of Alexa Top Million HTTPS sites."*

USEFUL TOOL: ZMAP



This paper appeared in *Proceedings of the 22nd USENIX Security Symposium*, August 2013.
ZMap source code and documentation are available for download at **https://zmap.io/**.

ZMap: Fast Internet-Wide Scanning and its Security Applications

Zakir Durumeric
University of Michigan
zakir@umich.edu

Eric Wustrow
University of Michigan
ewust@umich.edu

J. Alex Halderman
University of Michigan
jhalderm@umich.edu

Abstract

Internet-wide network scanning has numerous security applications, including exposing new vulnerabilities and tracking the adoption of defensive mechanisms, but probing the entire public address space with existing tools is both difficult and slow. We introduce ZMap, a modular, open-source network scanner specifically architected to perform Internet-wide scans and capable of surveying the entire IPv4 address space in under 45 minutes from user space on a single machine, approaching the theoretical maximum speed of gigabit Ethernet. We present the scanner architecture, experimentally characterize its performance and accuracy, and explore the security implications of high speed Internet-scale network surveys, both offensive and defensive. We also discuss best practices for good Internet citizenship when performing Internet-wide surveys, informed by our own experiences conducting a long-term research survey over the past year.

1 Introduction and Roadmap

Internet-scale network surveys collect data by probing large subsets of the public IP address space. While such scanning behavior is often associated with botnets and worms, it also has proved to be a valuable methodology for security research. Recent studies have demonstrated that Internet-wide scanning can help reveal new kinds of vulnerabilities, monitor deployment of mitigations, and shed light on previously opaque distributed ecosystems [10, 12, 14, 15, 25, 27]. Unfortunately, this methodology has been more accessible to attackers than to legitimate researchers, who cannot employ stolen network access or spread self-replicating code. Comprehensively scanning the public address space with off-the-shelf tools like Nmap [23] requires weeks of time or many machines.

In this paper, we introduce ZMap, a modular and open-source network scanner specifically designed for performing comprehensive Internet-wide research scans. A single mid-range machine running ZMap is capable of scanning for a given open port across the entire public IPv4 address space in under 45 minutes—over 97% of the theoretical maximum speed of gigabit Ethernet—without requiring specialized hardware [11] or kernel modules [8, 28]. ZMap's modular architecture can support many types of single-packet probes, including TCP SYN scans, ICMP echo request scans, and application-specific UDP scans, and it can interface easily with user-provided code to perform follow-up actions on discovered hosts, such as completing a protocol handshake.

Compared to Nmap—an excellent general-purpose network mapping tool, which was utilized in recent Internet-wide survey research [10, 14]—ZMap achieves much higher performance for Internet-scale scans. Experimentally, we find that ZMap is capable of scanning the IPv4 public address space over 1300 times faster than the most aggressive Nmap default settings, with equivalent accuracy. These performance gains are due to architectural choices that are specifically optimized for this application:

Optimized probing While Nmap adapts its transmission rate to avoid saturating the source or target networks, we assume that the source network is well provisioned (unable to be saturated by the source host), and that the targets are randomly ordered and widely dispersed (so no distant network or path is likely to be saturated by the scan). Consequently, we attempt to send probes as quickly as the source's NIC can support, skipping the TCP/IP stack and generating Ethernet frames directly. We show that ZMap can send probes at gigabit line speed from commodity hardware and entirely in user space.

No per-connection state While Nmap maintains state for each connection to track which hosts have been scanned and to handle timeouts and retransmissions, ZMap forgoes any per-connection state. Since it is intended to target random samples of the address space, ZMap can avoid storing the addresses it has already scanned or needs to scan and instead selects addresses according to a random permutation generated by a cyclic

***Goal**: port-scan the entire Internet in less than an hour*

Approaches:

Non-blocking, stateless
⟹ Highly parallelizable

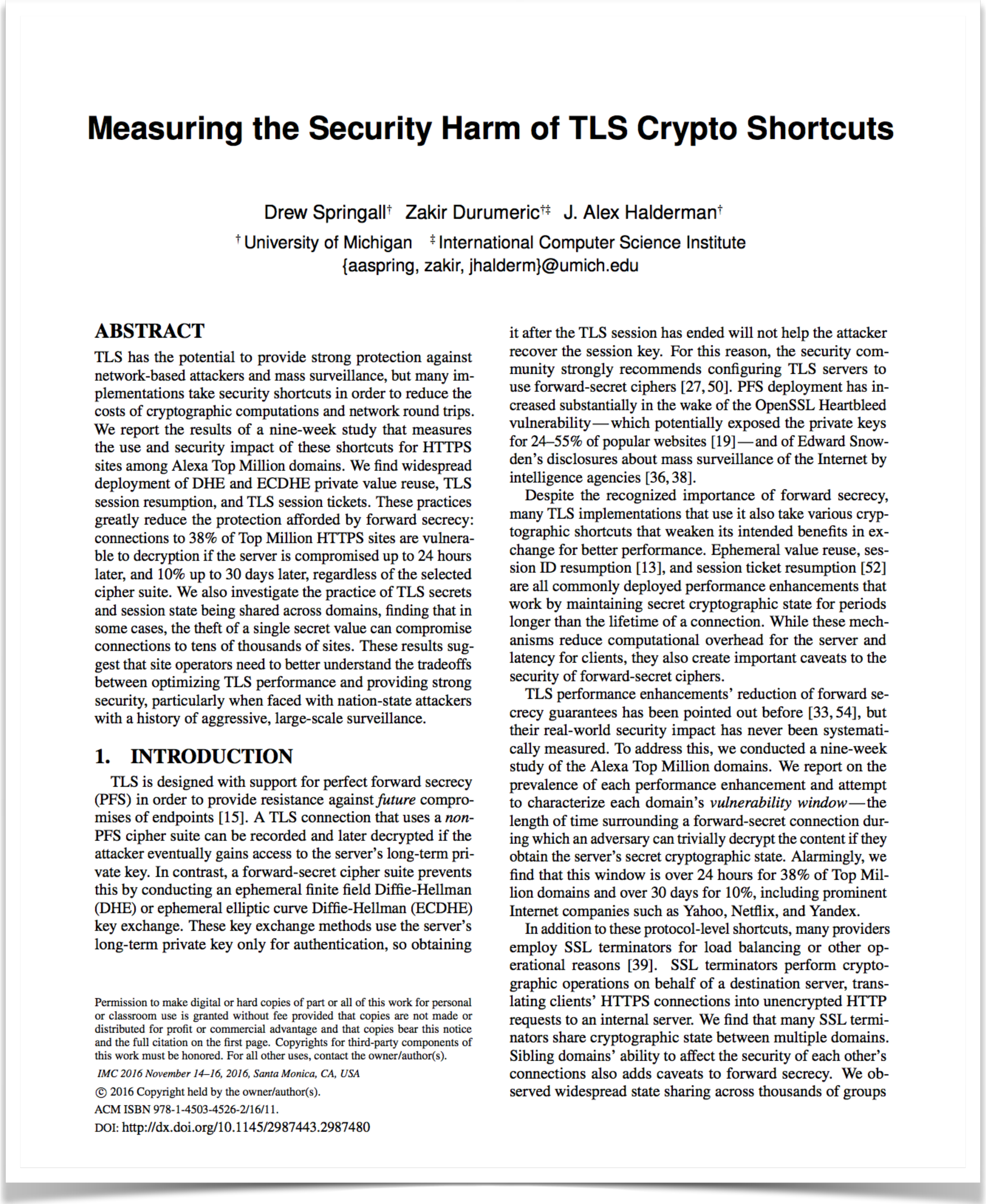
Randomize addresses
⟹ Avoid takedown notices

***Datasets**: Rapid7, censys.io*

FORWARD SECRECY

- Compromising a long-term key should not compromise past session keys

UNSAFE OPTIMIZATIONS



Measuring the Security Harm of TLS Crypto Shortcuts

Drew Springall[†] Zakir Durumeric[†‡] J. Alex Halderman[†]
[†]University of Michigan [‡]International Computer Science Institute
{aaspring, zakir, jhalderm}@umich.edu

ABSTRACT

TLS has the potential to provide strong protection against network-based attackers and mass surveillance, but many implementations take security shortcuts in order to reduce the costs of cryptographic computations and network round trips. We report the results of a nine-week study that measures the use and security impact of these shortcuts for HTTPS sites among Alexa Top Million domains. We find widespread deployment of DHE and ECDHE private value reuse, TLS session resumption, and TLS session tickets. These practices greatly reduce the protection afforded by forward secrecy: connections to 38% of Top Million HTTPS sites are vulnerable to decryption if the server is compromised up to 24 hours later, and 10% up to 30 days later, regardless of the selected cipher suite. We also investigate the practice of TLS secrets and session state being shared across domains, finding that in some cases, the theft of a single secret value can compromise connections to tens of thousands of sites. These results suggest that site operators need to better understand the tradeoffs between optimizing TLS performance and providing strong security, particularly when faced with nation-state attackers with a history of aggressive, large-scale surveillance.

1. INTRODUCTION

TLS is designed with support for perfect forward secrecy (PFS) in order to provide resistance against *future* compromises of endpoints [15]. A TLS connection that uses a *non*-PFS cipher suite can be recorded and later decrypted if the attacker eventually gains access to the server's long-term private key. In contrast, a forward-secret cipher suite prevents this by conducting an ephemeral finite field Diffie-Hellman (DHE) or ephemeral elliptic curve Diffie-Hellman (ECDHE) key exchange. These key exchange methods use the server's long-term private key only for authentication, so obtaining it after the TLS session has ended will not help the attacker recover the session key. For this reason, the security community strongly recommends configuring TLS servers to use forward-secret ciphers [27, 50]. PFS deployment has increased substantially in the wake of the OpenSSL Heartbleed vulnerability—which potentially exposed the private keys for 24–55% of popular websites [19]—and of Edward Snowden's disclosures about mass surveillance of the Internet by intelligence agencies [36, 38].

Despite the recognized importance of forward secrecy, many TLS implementations that use it also take various cryptographic shortcuts that weaken its intended benefits in exchange for better performance. Ephemeral value reuse, session ID resumption [13], and session ticket resumption [52] are all commonly deployed performance enhancements that work by maintaining secret cryptographic state for periods longer than the lifetime of a connection. While these mechanisms reduce computational overhead for the server and latency for clients, they also create important caveats to the security of forward-secret ciphers.

TLS performance enhancements' reduction of forward secrecy guarantees has been pointed out before [33, 54], but their real-world security impact has never been systematically measured. To address this, we conducted a nine-week study of the Alexa Top Million domains. We report on the prevalence of each performance enhancement and attempt to characterize each domain's *vulnerability window*—the length of time surrounding a forward-secret connection during which an adversary can trivially decrypt the content if they obtain the server's secret cryptographic state. Alarmingly, we find that this window is over 24 hours for 38% of Top Million domains and over 30 days for 10%, including prominent Internet companies such as Yahoo, Netflix, and Yandex.

In addition to these protocol-level shortcuts, many providers employ SSL terminators for load balancing or other operational reasons [39]. SSL terminators perform cryptographic operations on behalf of a destination server, translating clients' HTTPS connections into unencrypted HTTP requests to an internal server. We find that many SSL terminators share cryptographic state between multiple domains. Sibling domains' ability to affect the security of each other's connections also adds caveats to forward secrecy. We observed widespread state sharing across thousands of groups

IMC 2016 November 14–16, 2016, Santa Monica, CA, USA

ACM ISBN 978-1-4503-4526-2/16/11.
DOI: http://dx.doi.org/10.1145/2987443.2987480

TLS session ticket resumption

Session ticket: session keys and other data to resume the session

Server sends an "opaque" ticket (encrypted with the Session Ticket Encryption Key, STEK)

Client sends the encrypted session ticket during handshake; server uses the STEK to recover it and pick up in one round-trip of communication

UNSAFE OPTIMIZATIONS

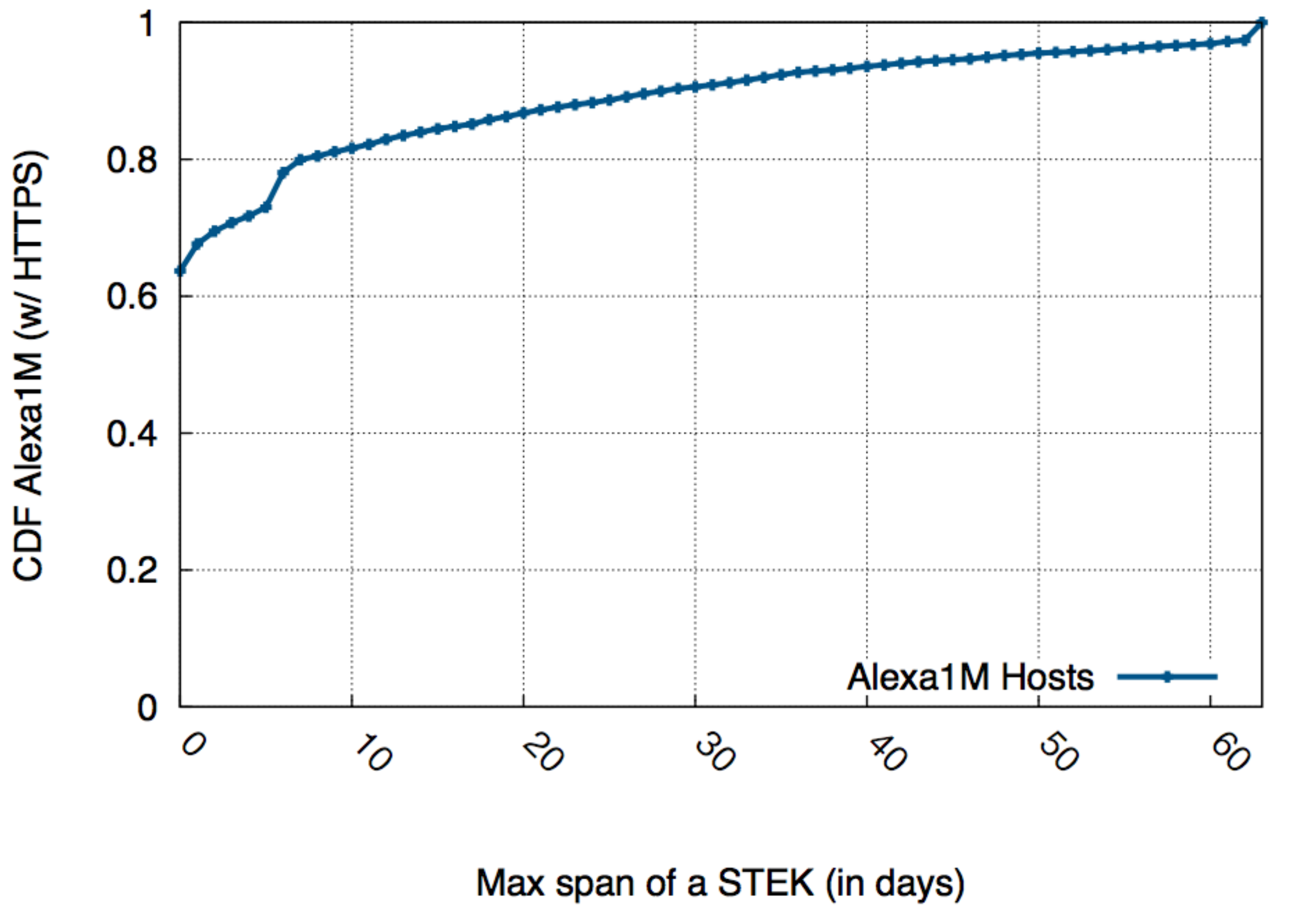


Figure 3: **STEK Lifetime**—TLS connections cannot achieve forward secrecy until the STEK (the key used by the server to encrypt the session ticket) is discarded.

Incentive to hold onto STEKs (lower RTTs)

But they're holding onto them long enough for nation-states to recover them

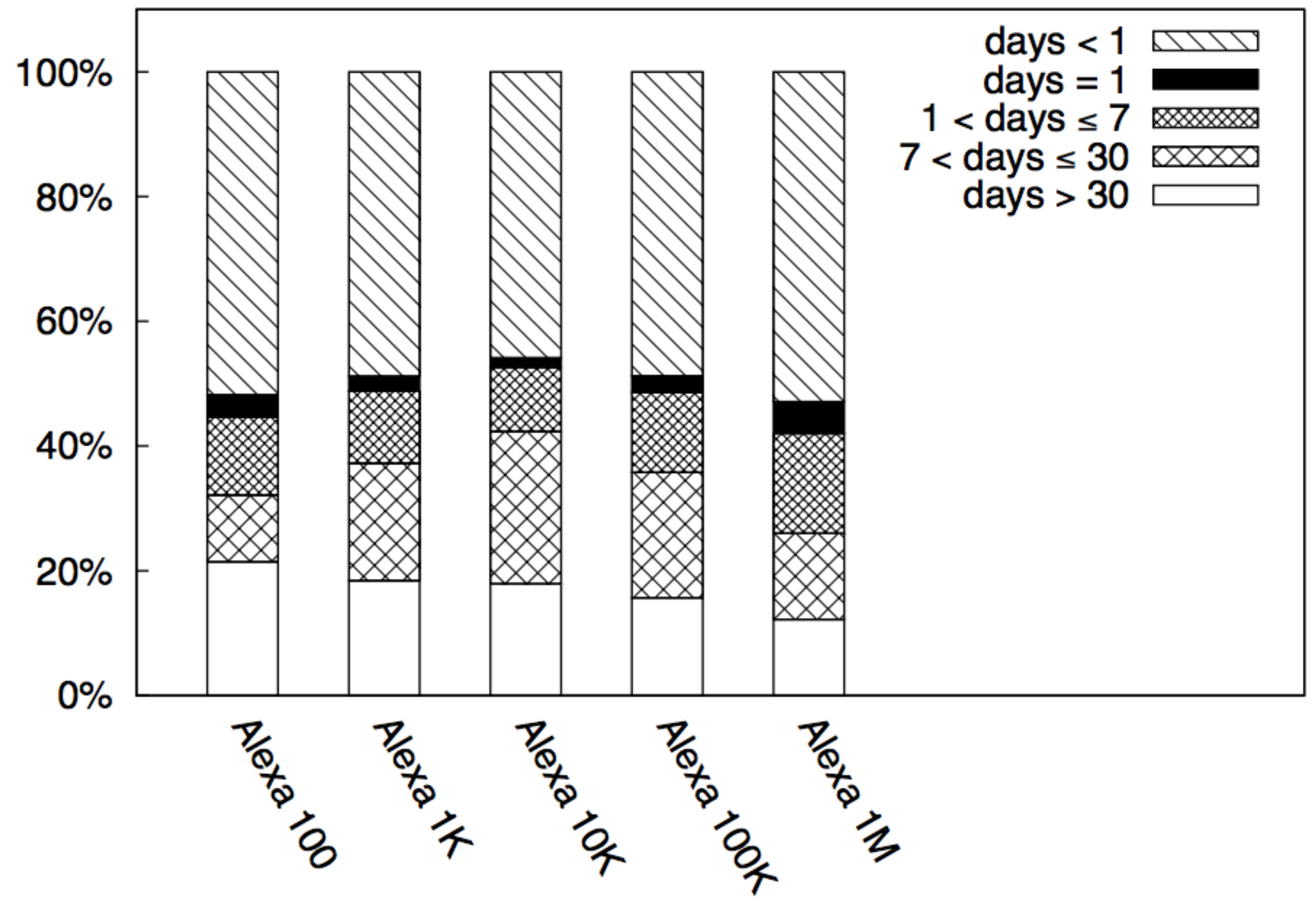


Figure 4: **STEK Lifetime by Alexa Rank**—We found 12 Alexa Top 100 sites that persisted STEKs for at least 30 days.

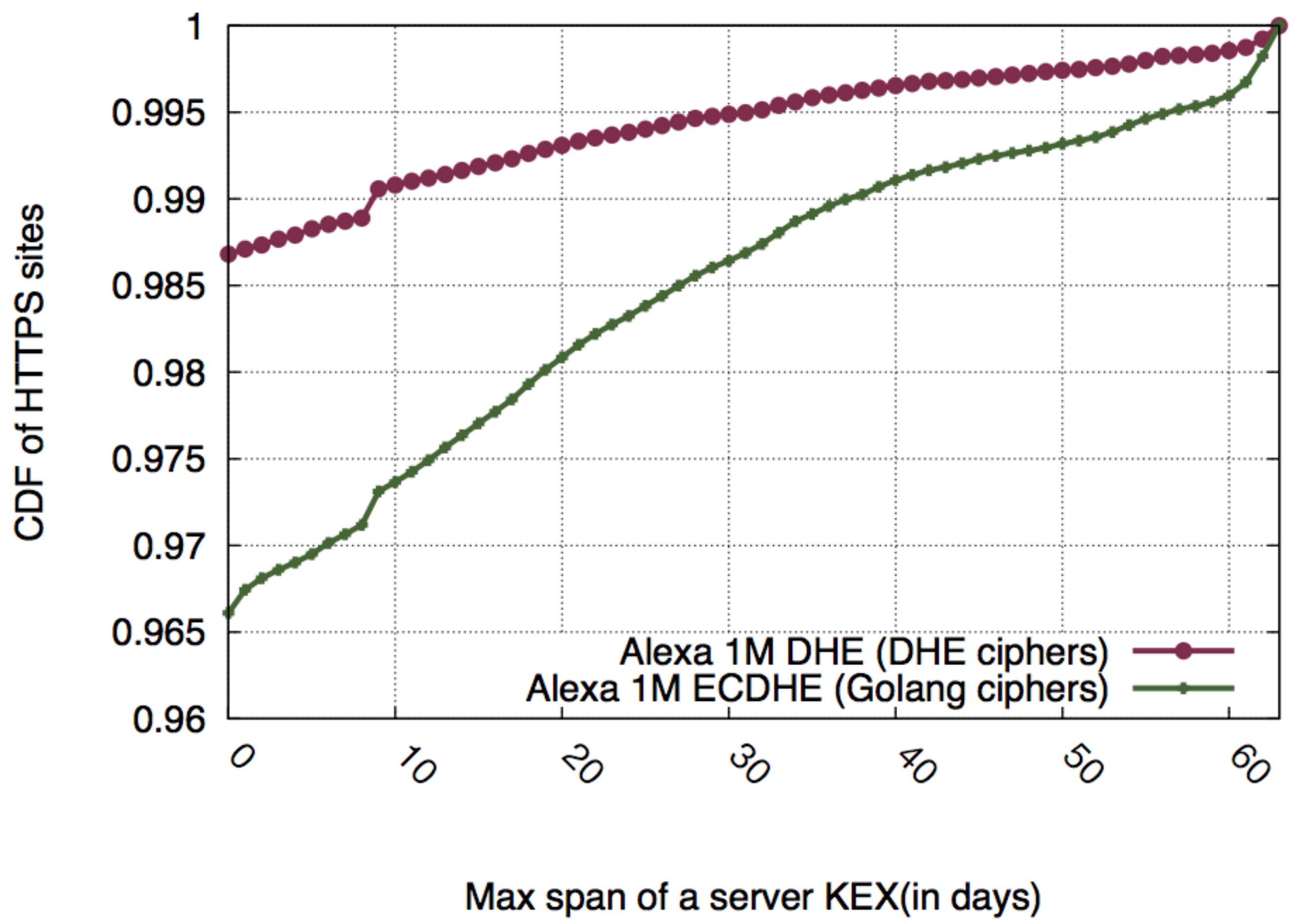
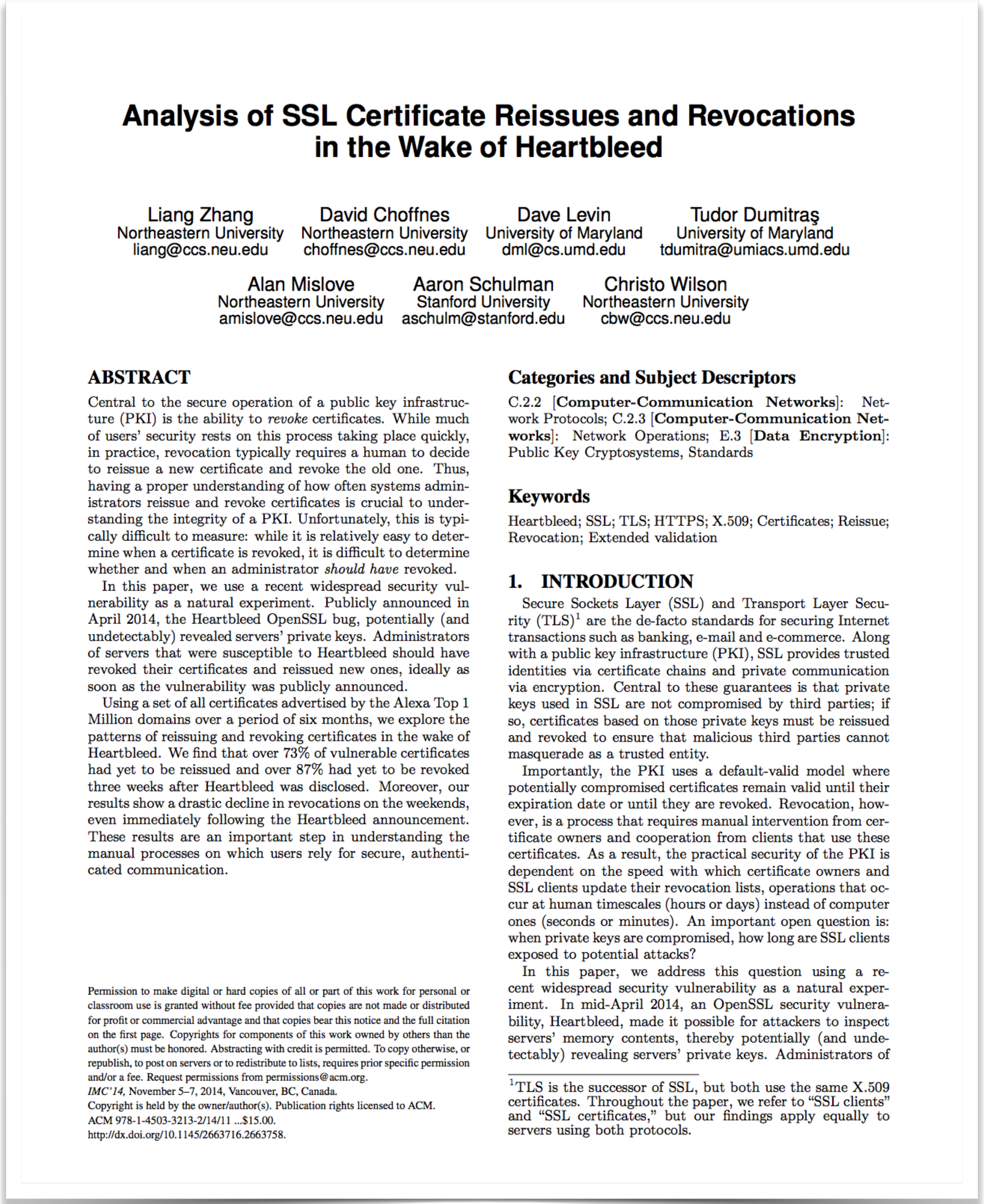


Figure 5: **Ephemeral Exchange Value Reuse**—We measured how long Alexa Top Million websites served identical DHE and ECDHE values (note vertical scale is cropped).

POOR CERTIFICATE MANAGEMENT



Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed

Liang Zhang
Northeastern University
liang@ccs.neu.edu

David Choffnes
Northeastern University
choffnes@ccs.neu.edu

Dave Levin
University of Maryland
dml@cs.umd.edu

Tudor Dumitraş
University of Maryland
tdumitra@umiacs.umd.edu

Alan Mislove
Northeastern University
amislove@ccs.neu.edu

Aaron Schulman
Stanford University
aschulm@stanford.edu

Christo Wilson
Northeastern University
cbw@ccs.neu.edu

ABSTRACT

Central to the secure operation of a public key infrastructure (PKI) is the ability to *revoke* certificates. While much of users' security rests on this process taking place quickly, in practice, revocation typically requires a human to decide to reissue a new certificate and revoke the old one. Thus, having a proper understanding of how often systems administrators reissue and revoke certificates is crucial to understanding the integrity of a PKI. Unfortunately, this is typically difficult to measure: while it is relatively easy to determine when a certificate is revoked, it is difficult to determine whether and when an administrator *should have* revoked.

In this paper, we use a recent widespread security vulnerability as a natural experiment. Publicly announced in April 2014, the Heartbleed OpenSSL bug, potentially (and undetectably) revealed servers' private keys. Administrators of servers that were susceptible to Heartbleed should have revoked their certificates and reissued new ones, ideally as soon as the vulnerability was publicly announced.

Using a set of all certificates advertised by the Alexa Top 1 Million domains over a period of six months, we explore the patterns of reissuing and revoking certificates in the wake of Heartbleed. We find that over 73% of vulnerable certificates had yet to be reissued and over 87% had yet to be revoked three weeks after Heartbleed was disclosed. Moreover, our results show a drastic decline in revocations on the weekends, even immediately following the Heartbleed announcement. These results are an important step in understanding the manual processes on which users rely for secure, authenticated communication.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.3 [**Computer-Communication Networks**]: Network Operations; E.3 [**Data Encryption**]: Public Key Cryptosystems, Standards

Keywords

Heartbleed; SSL; TLS; HTTPS; X.509; Certificates; Reissue; Revocation; Extended validation

1. INTRODUCTION

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)[1] are the de-facto standards for securing Internet transactions such as banking, e-mail and e-commerce. Along with a public key infrastructure (PKI), SSL provides trusted identities via certificate chains and private communication via encryption. Central to these guarantees is that private keys used in SSL are not compromised by third parties; if so, certificates based on those private keys must be reissued and revoked to ensure that malicious third parties cannot masquerade as a trusted entity.

Importantly, the PKI uses a default-valid model where potentially compromised certificates remain valid until their expiration date or until they are revoked. Revocation, however, is a process that requires manual intervention from certificate owners and cooperation from clients that use these certificates. As a result, the practical security of the PKI is dependent on the speed with which certificate owners and SSL clients update their revocation lists, operations that occur at human timescales (hours or days) instead of computer ones (seconds or minutes). An important open question is: when private keys are compromised, how long are SSL clients exposed to potential attacks?

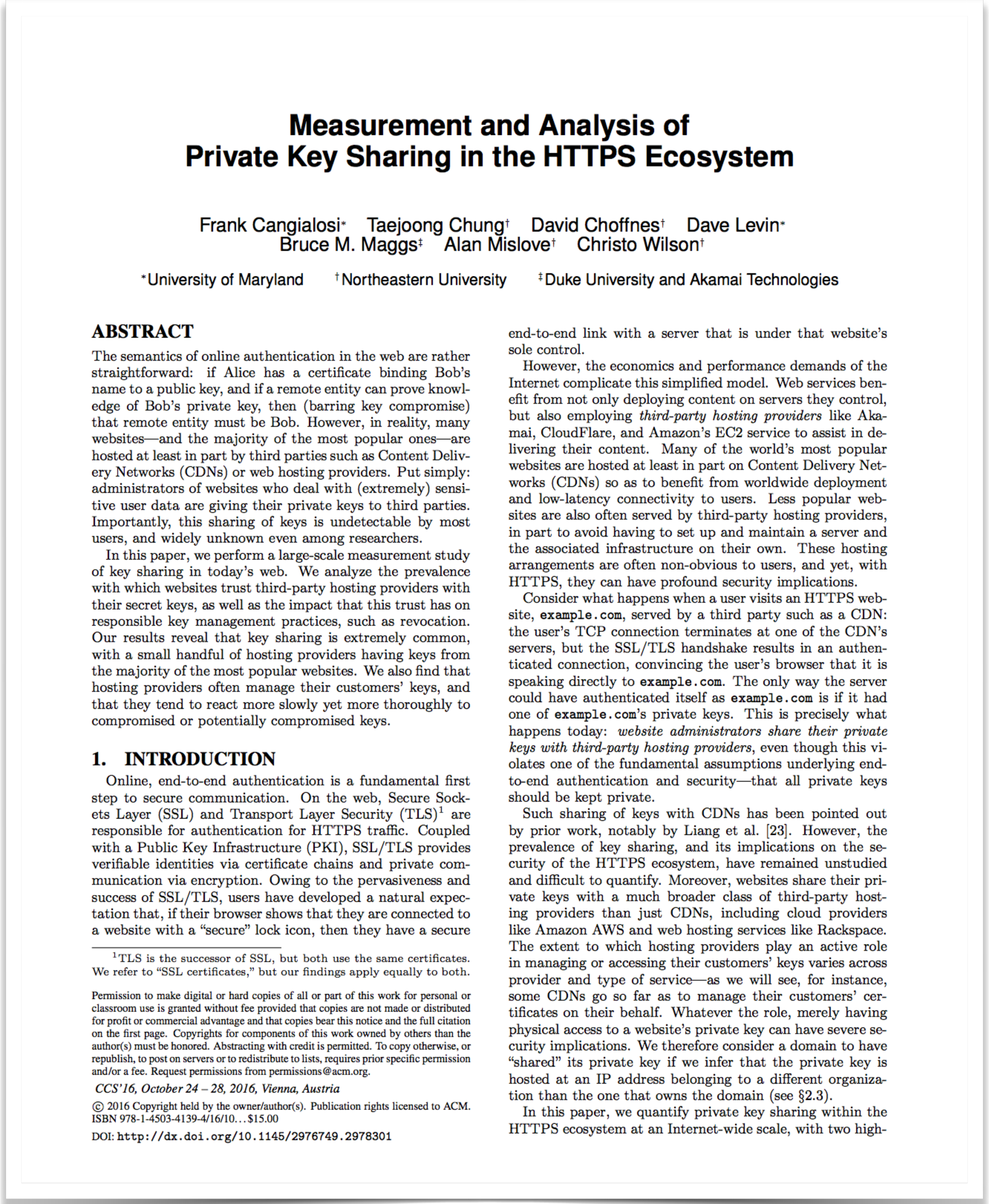
In this paper, we address this question using a recent widespread security vulnerability as a natural experiment. In mid-April 2014, an OpenSSL security vulnerability, Heartbleed, made it possible for attackers to inspect servers' memory contents, thereby potentially (and undetectably) revealing servers' private keys. Administrators of

IMC'14, November 5–7, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-3213-2/14/11 ...$15.00.
http://dx.doi.org/10.1145/2663716.2663758.

---

[1] TLS is the successor of SSL, but both use the same X.509 certificates. Throughout the paper, we refer to "SSL clients" and "SSL certificates," but our findings apply equally to servers using both protocols.



Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

Frank Cangialosi[*] Taejoong Chung[†] David Choffnes[†] Dave Levin[*]
Bruce M. Maggs[‡] Alan Mislove[†] Christo Wilson[†]

[*]University of Maryland [†]Northeastern University [‡]Duke University and Akamai Technologies

ABSTRACT

The semantics of online authentication in the web are rather straightforward: if Alice has a certificate binding Bob's name to a public key, and if a remote entity can prove knowledge of Bob's private key, then (barring key compromise) that remote entity must be Bob. However, in reality, many websites—and the majority of the most popular ones—are hosted at least in part by third parties such as Content Delivery Networks (CDNs) or web hosting providers. Put simply: administrators of websites who deal with (extremely) sensitive user data are giving their private keys to third parties. Importantly, this sharing of keys is undetectable by most users, and widely unknown even among researchers.

In this paper, we perform a large-scale measurement study of key sharing in today's web. We analyze the prevalence with which websites trust third-party hosting providers with their secret keys, as well as the impact that this trust has on responsible key management practices, such as revocation. Our results reveal that key sharing is extremely common, with a small handful of hosting providers having keys from the majority of the most popular websites. We also find that hosting providers often manage their customers' keys, and that they tend to react more slowly yet more thoroughly to compromised or potentially compromised keys.

1. INTRODUCTION

Online, end-to-end authentication is a fundamental first step to secure communication. On the web, Secure Sockets Layer (SSL) and Transport Layer Security (TLS)[1] are responsible for authentication for HTTPS traffic. Coupled with a Public Key Infrastructure (PKI), SSL/TLS provides verifiable identities via certificate chains and private communication via encryption. Owing to the pervasiveness and success of SSL/TLS, users have developed a natural expectation that, if their browser shows that they are connected to a website with a "secure" lock icon, then they have a secure end-to-end link with a server that is under that website's sole control.

However, the economics and performance demands of the Internet complicate this simplified model. Web services benefit from not only deploying content on servers they control, but also employing *third-party hosting providers* like Akamai, CloudFlare, and Amazon's EC2 service to assist in delivering their content. Many of the world's most popular websites are hosted at least in part on Content Delivery Networks (CDNs) so as to benefit from worldwide deployment and low-latency connectivity to users. Less popular websites are also often served by third-party hosting providers, in part to avoid having to set up and maintain a server and the associated infrastructure on their own. These hosting arrangements are often non-obvious to users, and yet, with HTTPS, they can have profound security implications.

Consider what happens when a user visits an HTTPS website, `example.com`, served by a third party such as a CDN: the user's TCP connection terminates at one of the CDN's servers, but the SSL/TLS handshake results in an authenticated connection, convincing the user's browser that it is speaking directly to `example.com`. The only way the server could have authenticated itself as `example.com` is if it had one of `example.com`'s private keys. This is precisely what happens today: *website administrators share their private keys with third-party hosting providers*, even though this violates one of the fundamental assumptions underlying end-to-end authentication and security—that all private keys should be kept private.

Such sharing of keys with CDNs has been pointed out by prior work, notably by Liang et al. [23]. However, the prevalence of key sharing, and its implications on the security of the HTTPS ecosystem, have remained unstudied and difficult to quantify. Moreover, websites share their private keys with a much broader class of third-party hosting providers than just CDNs, including cloud providers like Amazon AWS and web hosting services like Rackspace. The extent to which hosting providers play an active role in managing or accessing their customers' keys varies across provider and type of service—as we will see, for instance, some CDNs go so far as to manage their customers' certificates on their behalf. Whatever the role, merely having physical access to a website's private key can have severe security implications. We therefore consider a domain to have "shared" its private key if we infer that the private key is hosted at an IP address belonging to a different organization than the one that owns the domain (see §2.3).

In this paper, we quantify private key sharing within the HTTPS ecosystem at an Internet-wide scale, with two high-

CCS'16, October 24 – 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10…$15.00
DOI: `http://dx.doi.org/10.1145/2976749.2978301`

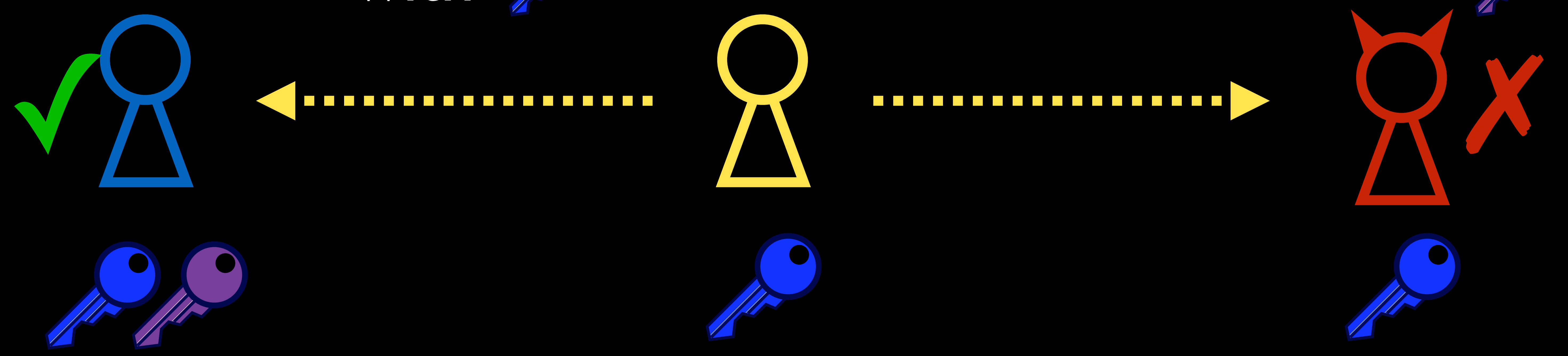
---

[1] TLS is the successor of SSL, but both use the same certificates. We refer to "SSL certificates," but our findings apply equally to both.

Public Key Cryptography

public private

Encrypts Decrypts



Messages encrypted with

Cannot decrypt without

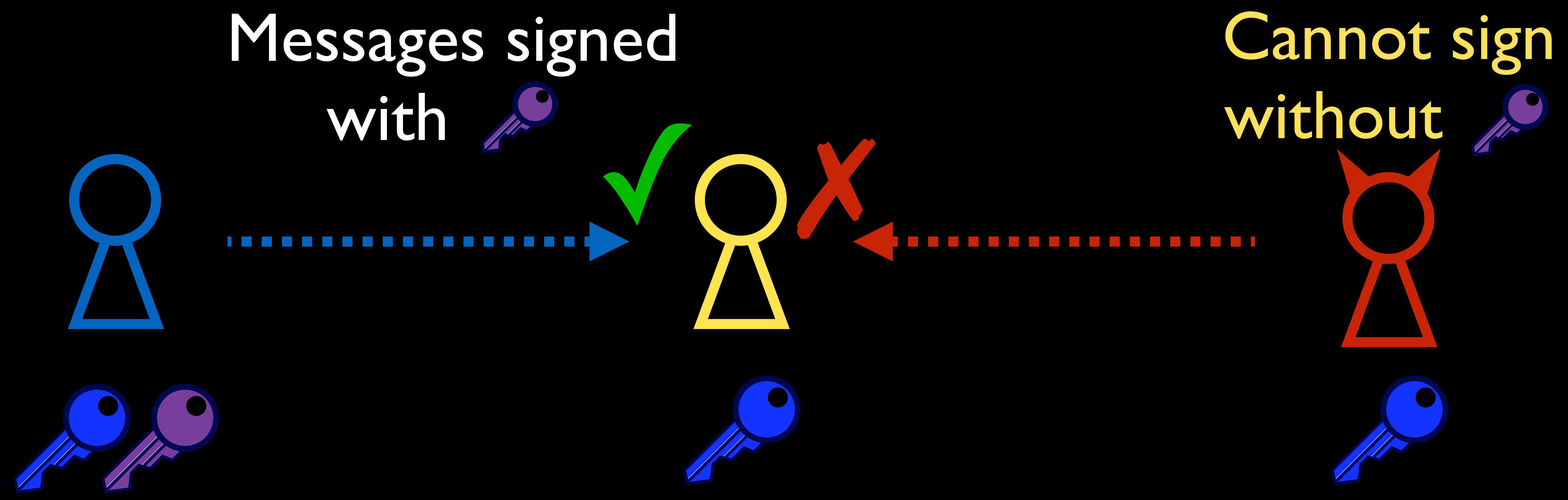
Confidentiality

Encrypt the data in a way that only the owner of a given public key can decrypt

Public Key Cryptography

public 🔑 🔑 private

Verifies Signs



Messages signed with 🔑

✓ ✗

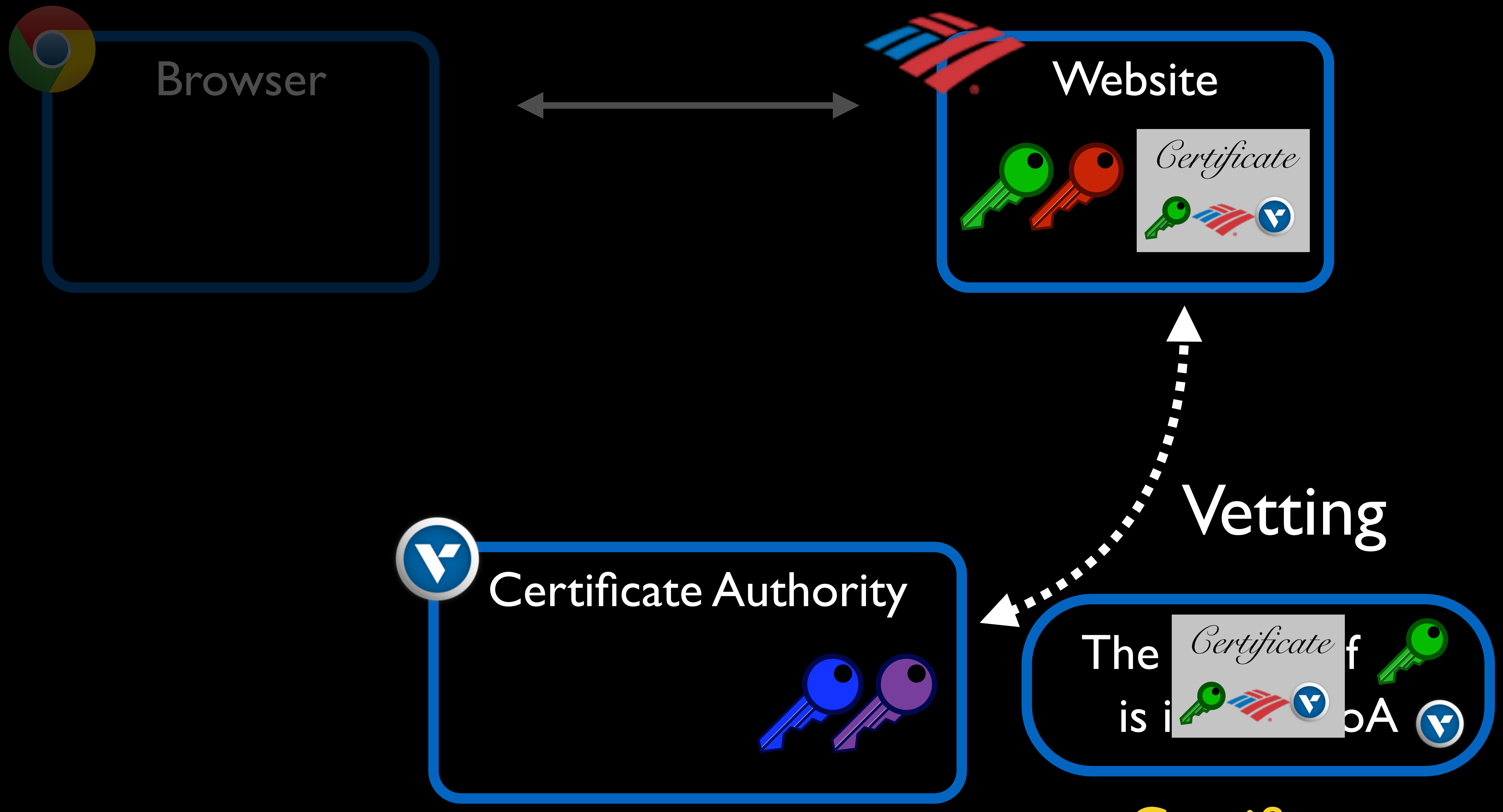
Cannot sign without 🔑

Authentication

Sign the data in a way that only the owner of a given public key can

Public Key Infrastructures (PKIs)

How can users truly know with whom they are communicating?



Browser

Website

Certificate

Vetting

Certificate Authority

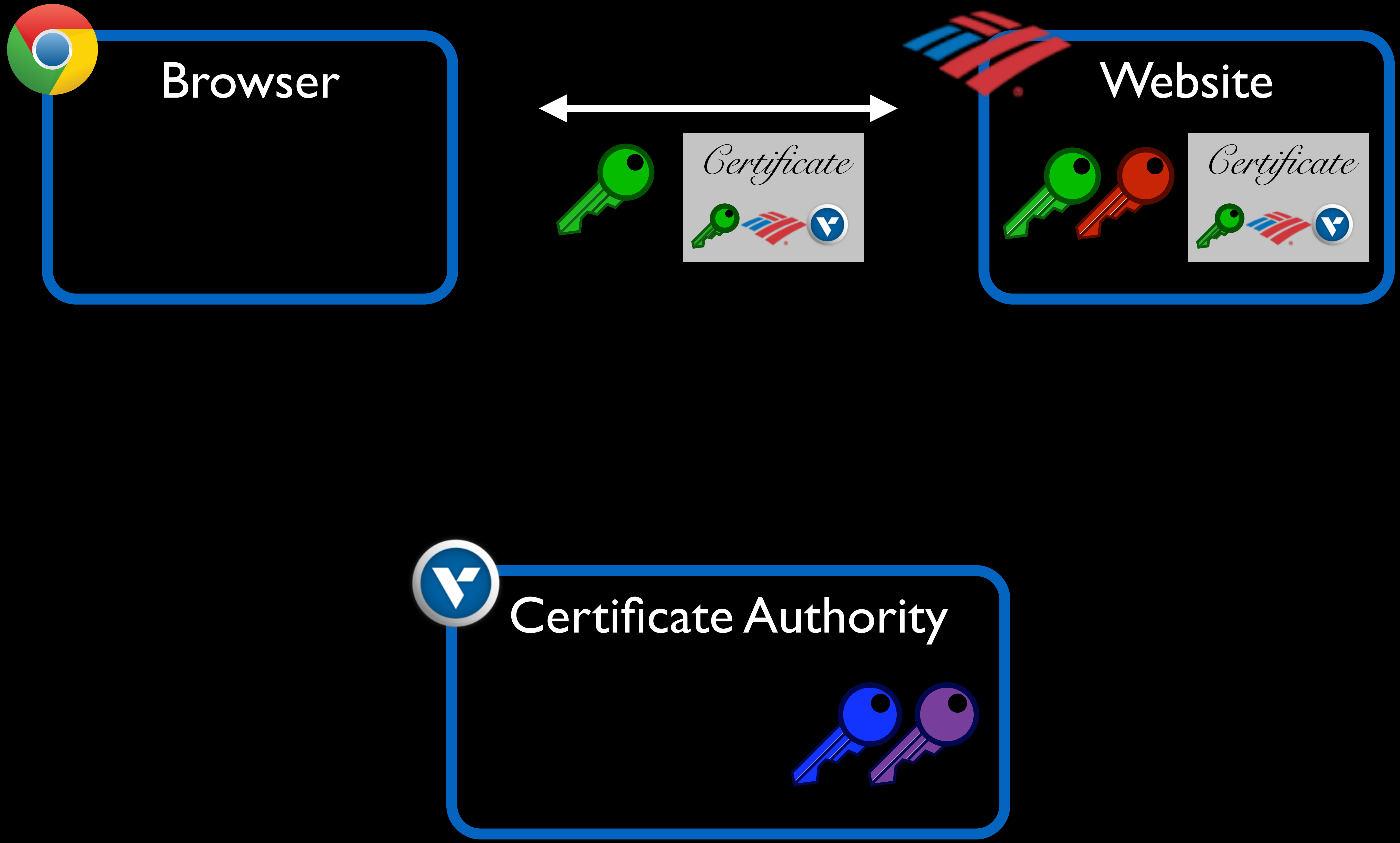
The Certificate f

is i oA

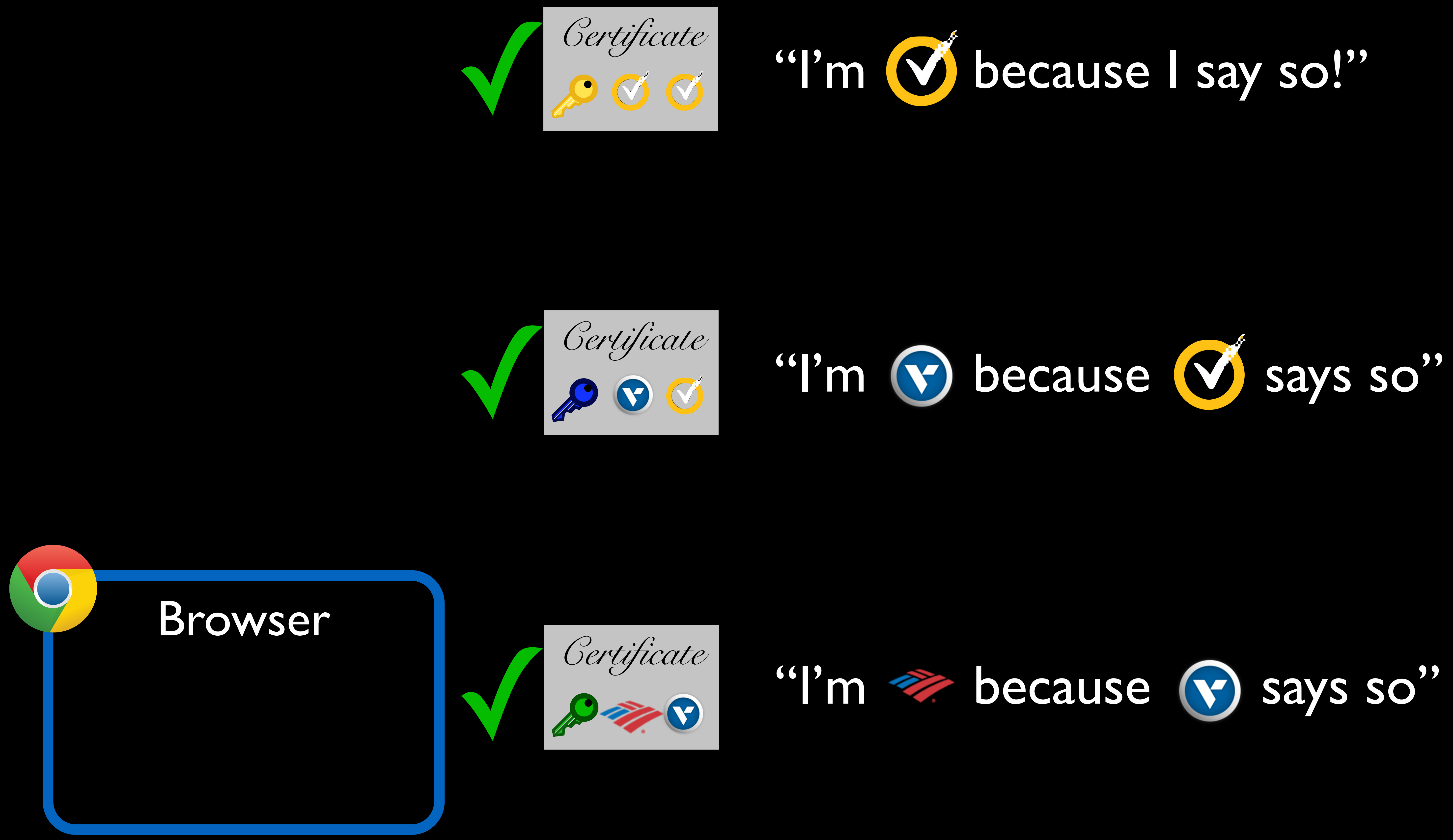
Certificate

Public Key Infrastructures (PKIs)

How can users truly know with whom they are communicating?



Verifying certificates

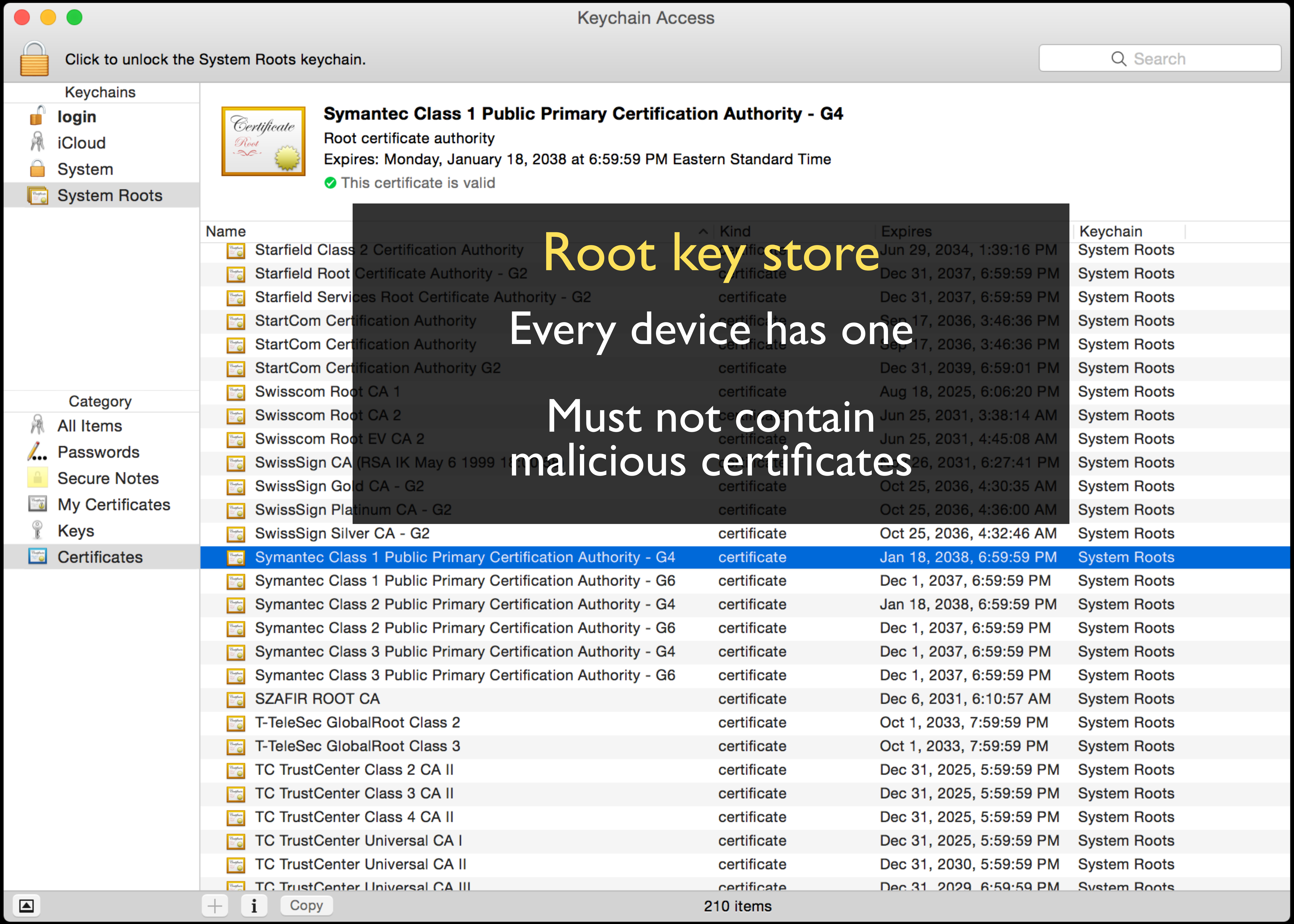


“I’m ✓ because I say so!”

“I’m V because ✓ says so”

“I’m because V says so”

Verifying certificates



Click to unlock the System Roots keychain.

Symantec Class 1 Public Primary Certification Authority - G4
Root certificate authority
Expires: Monday, January 18, 2038 at 6:59:59 PM Eastern Standard Time
✓ This certificate is valid

Name	Kind	Expires	Keychain
Starfield Class 2 Certification Authority	certificate	Jun 29, 2034, 1:39:16 PM	System Roots
Starfield Root Certificate Authority - G2	certificate	Dec 31, 2037, 6:59:59 PM	System Roots
Starfield Services Root Certificate Authority - G2	certificate	Dec 31, 2037, 6:59:59 PM	System Roots
StartCom Certification Authority	certificate	Sep 17, 2036, 3:46:36 PM	System Roots
StartCom Certification Authority	certificate	Sep 17, 2036, 3:46:36 PM	System Roots
StartCom Certification Authority G2	certificate	Dec 31, 2039, 6:59:01 PM	System Roots
Swisscom Root CA 1	certificate	Aug 18, 2025, 6:06:20 PM	System Roots
Swisscom Root CA 2	certificate	Jun 25, 2031, 3:38:14 AM	System Roots
Swisscom Root EV CA 2	certificate	Jun 25, 2031, 4:45:08 AM	System Roots
SwissSign CA (RSA IK May 6 1999 1	certificate	26, 2031, 6:27:41 PM	System Roots
SwissSign Gold CA - G2	certificate	Oct 25, 2036, 4:30:35 AM	System Roots
SwissSign Platinum CA - G2	certificate	Oct 25, 2036, 4:36:00 AM	System Roots
SwissSign Silver CA - G2	certificate	Oct 25, 2036, 4:32:46 AM	System Roots
Symantec Class 1 Public Primary Certification Authority - G4	certificate	Jan 18, 2038, 6:59:59 PM	System Roots
Symantec Class 1 Public Primary Certification Authority - G6	certificate	Dec 1, 2037, 6:59:59 PM	System Roots
Symantec Class 2 Public Primary Certification Authority - G4	certificate	Jan 18, 2038, 6:59:59 PM	System Roots
Symantec Class 2 Public Primary Certification Authority - G6	certificate	Dec 1, 2037, 6:59:59 PM	System Roots
Symantec Class 3 Public Primary Certification Authority - G4	certificate	Dec 1, 2037, 6:59:59 PM	System Roots
Symantec Class 3 Public Primary Certification Authority - G6	certificate	Dec 1, 2037, 6:59:59 PM	System Roots
SZAFIR ROOT CA	certificate	Dec 6, 2031, 6:10:57 AM	System Roots
T-TeleSec GlobalRoot Class 2	certificate	Oct 1, 2033, 7:59:59 PM	System Roots
T-TeleSec GlobalRoot Class 3	certificate	Oct 1, 2033, 7:59:59 PM	System Roots
TC TrustCenter Class 2 CA II	certificate	Dec 31, 2025, 5:59:59 PM	System Roots
TC TrustCenter Class 3 CA II	certificate	Dec 31, 2025, 5:59:59 PM	System Roots
TC TrustCenter Class 4 CA II	certificate	Dec 31, 2025, 5:59:59 PM	System Roots
TC TrustCenter Universal CA I	certificate	Dec 31, 2025, 5:59:59 PM	System Roots
TC TrustCenter Universal CA II	certificate	Dec 31, 2030, 5:59:59 PM	System Roots
TC TrustCenter Universal CA III	certificate	Dec 31, 2029, 6:59:59 PM	System Roots

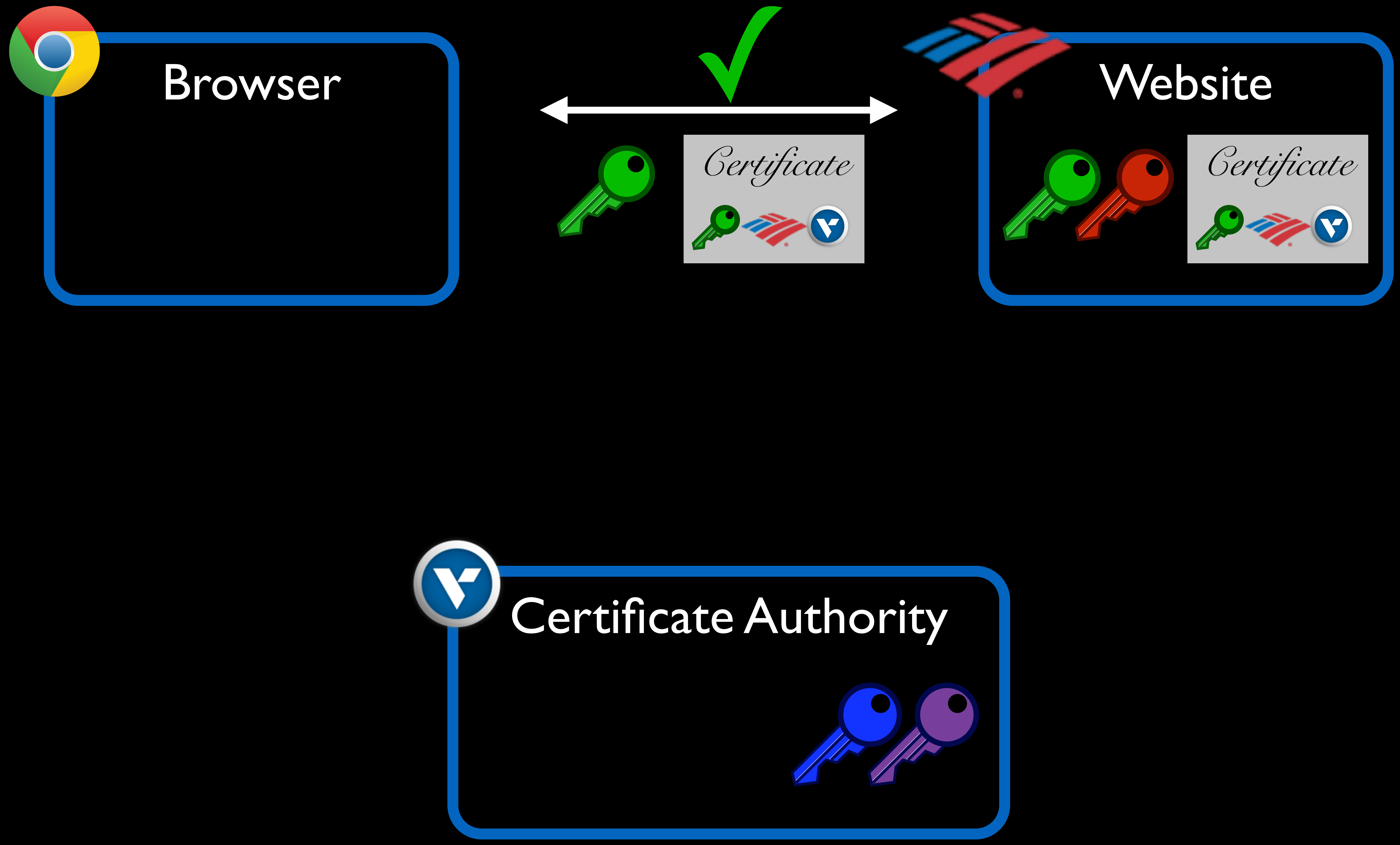
Root key store

Every device has one

Must not contain malicious certificates

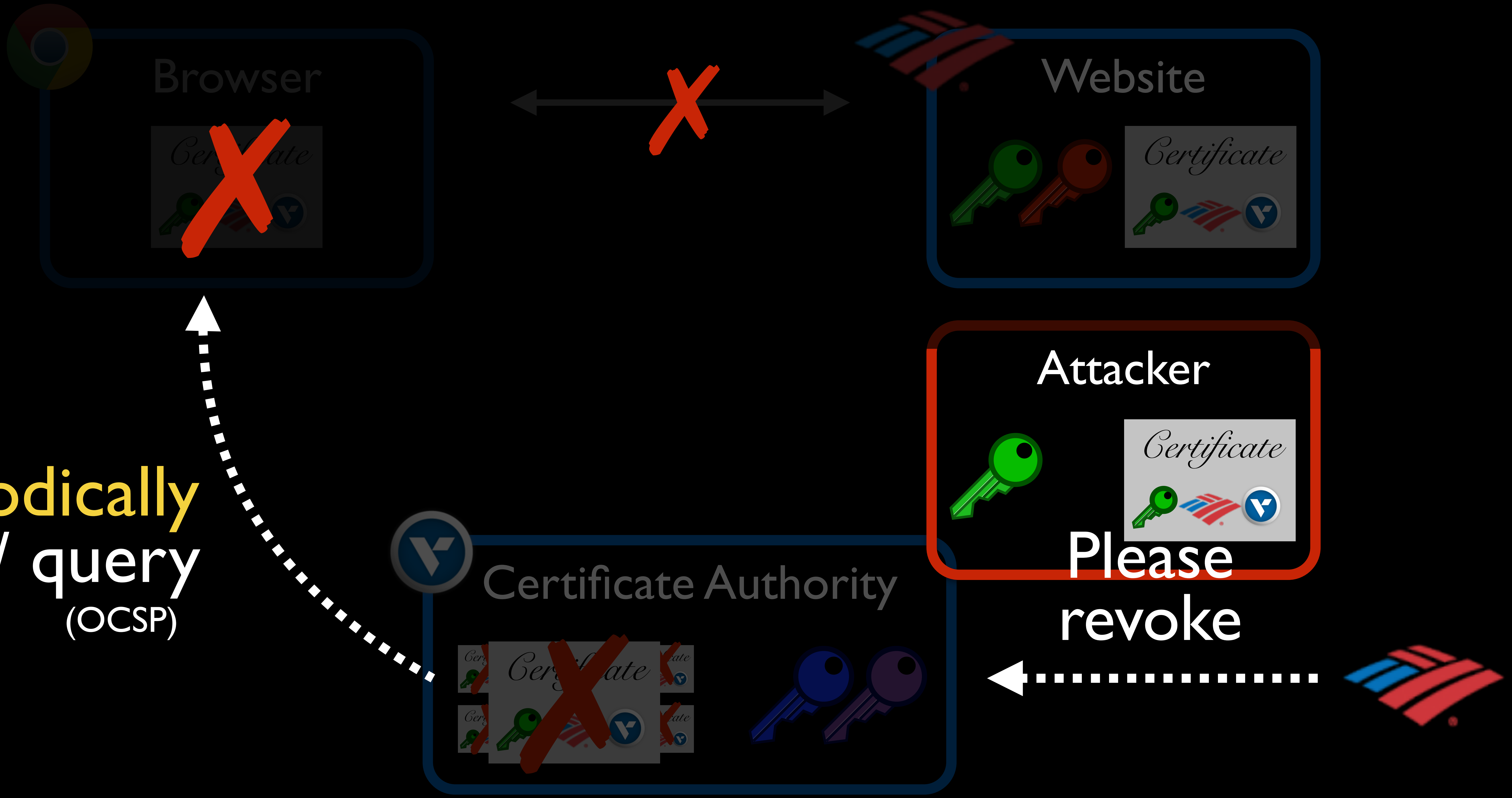
Public Key Infrastructures (PKIs)

How can users truly know with whom they are communicating?



Certificate revocation

What happens when a certificate is no longer valid?



Periodically
pull / query
(CRL) (OCSP)

Please
revoke

Certificate
Revocation

Certificate revocation

CRL

- Certificate Revocation List
  - Pull

OCSP

- Online Certificate Status Protocol
  - Query

Certificate revocation

is a critical part of any PKI

Administrators must revoke and reissue as quickly as possible

Browsers/OSes should obtain revocations as quickly as possible

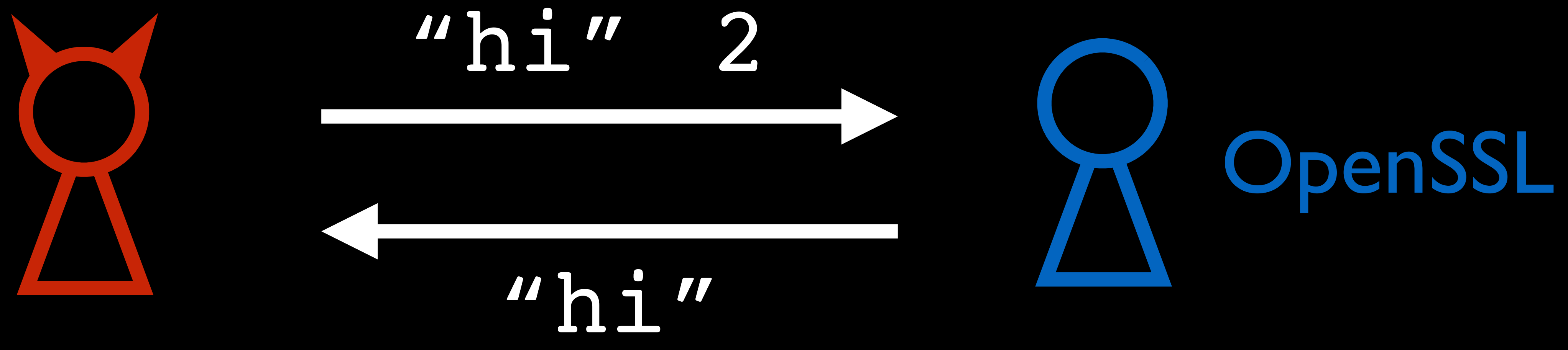
Administrators must revoke and reissue
as quickly as possible

Analysis of SSL certificate reissues and revocations in the wake of Heartbleed

Liang Zhang, David Choffnes, Tudor Dumitras, Dave Levin,
Alan Mislove, Aaron Schulman, Christo Wilson

ACM IMC 2014

Heartbleed

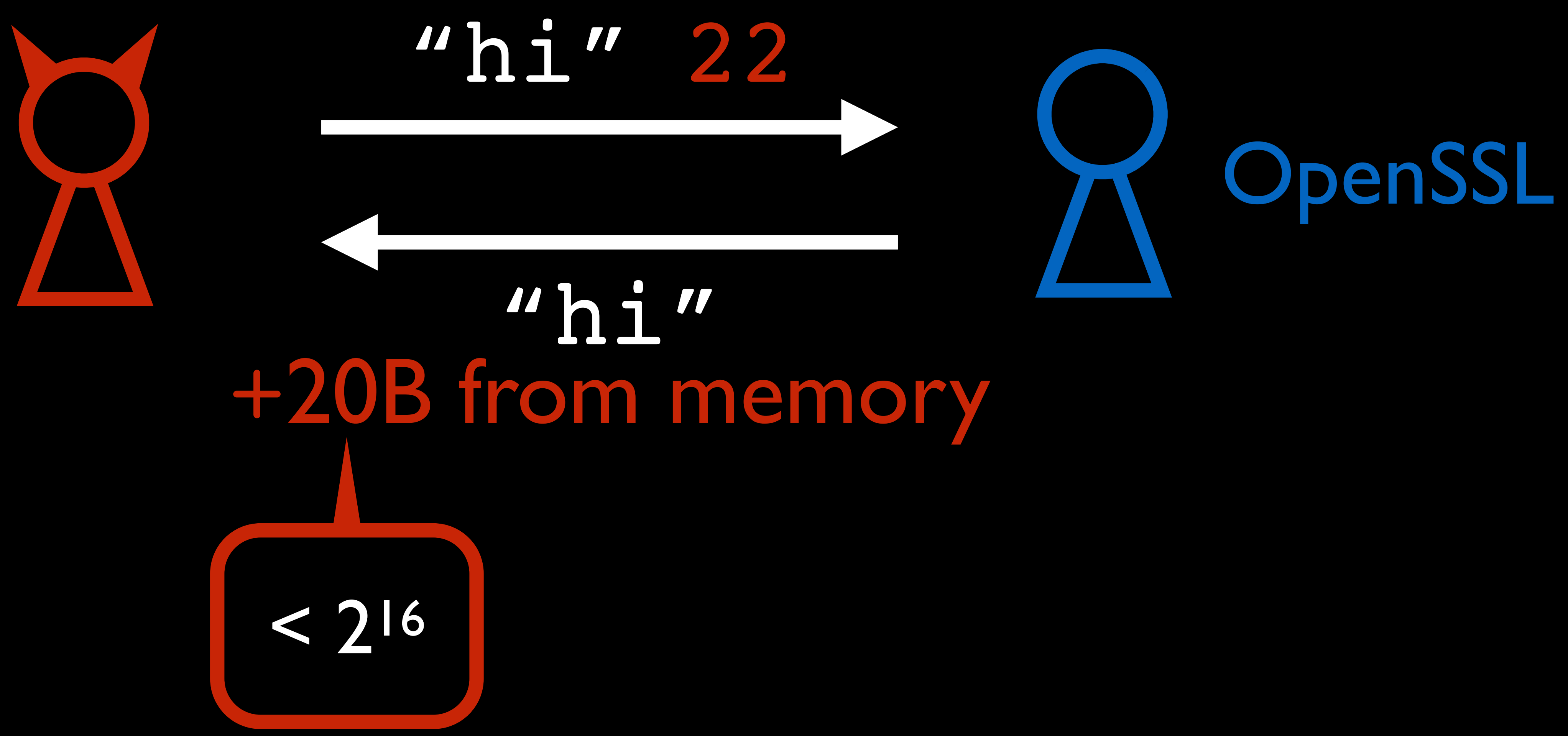


“hi” 2

“hi”

OpenSSL

Heartbleed



“hi” 22

OpenSSL

“hi”

+20B from memory

< 2^{16}

Potentially reveals user data and private keys

Heartbleed exploits were undetectable

Why study Heartbleed?

Discovered
03/21

Akamai patched
04/02

Publicly announced
04/07

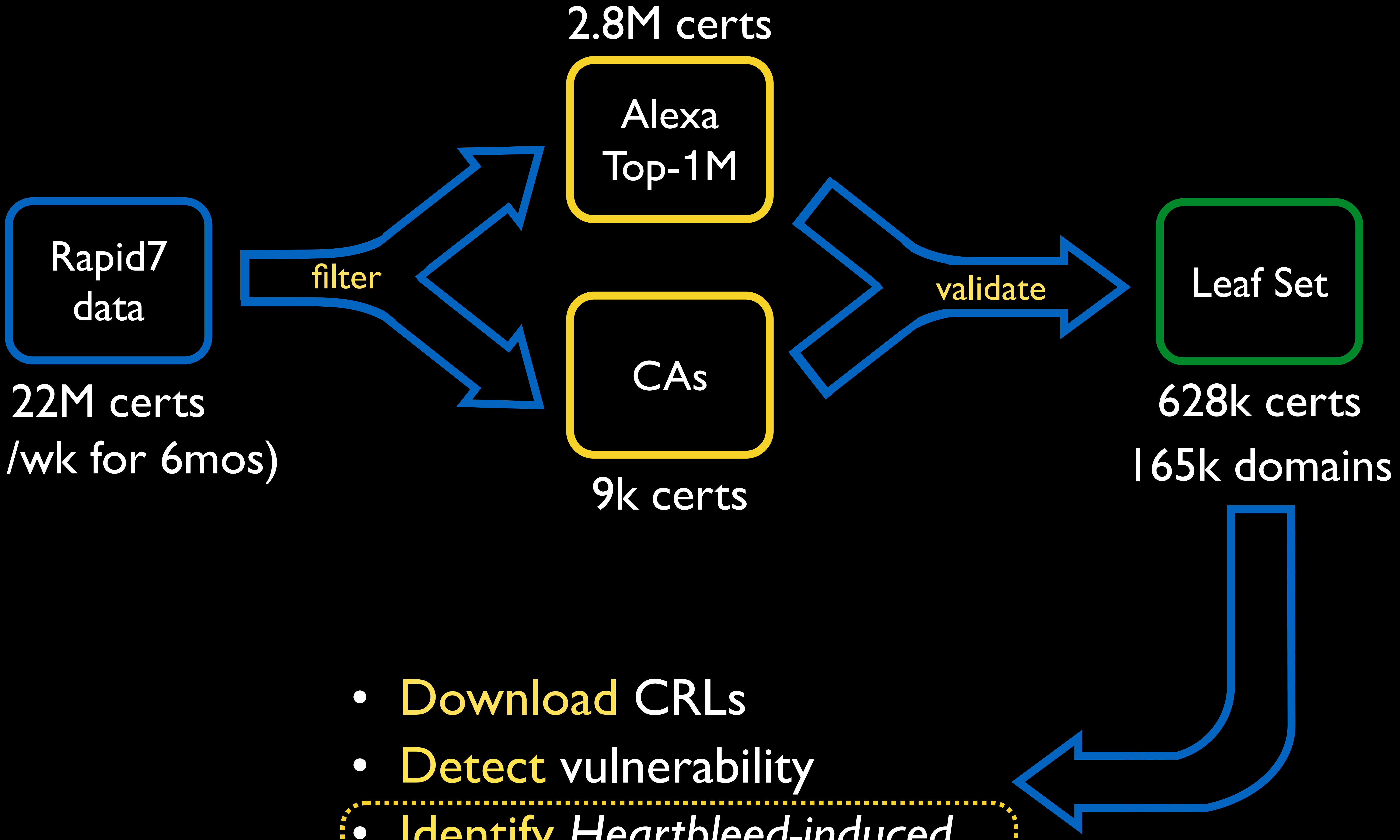
Every vulnerable website should have:

① Patched ② Revoked ③ Reissued

Heartbleed is a natural experiment:

How quickly and thoroughly do administrators act?

Dataset



2.8M certs

Alexa Top-1M

Rapid7 data

filter

22M certs
(~1/wk for 6mos)

CAs

9k certs

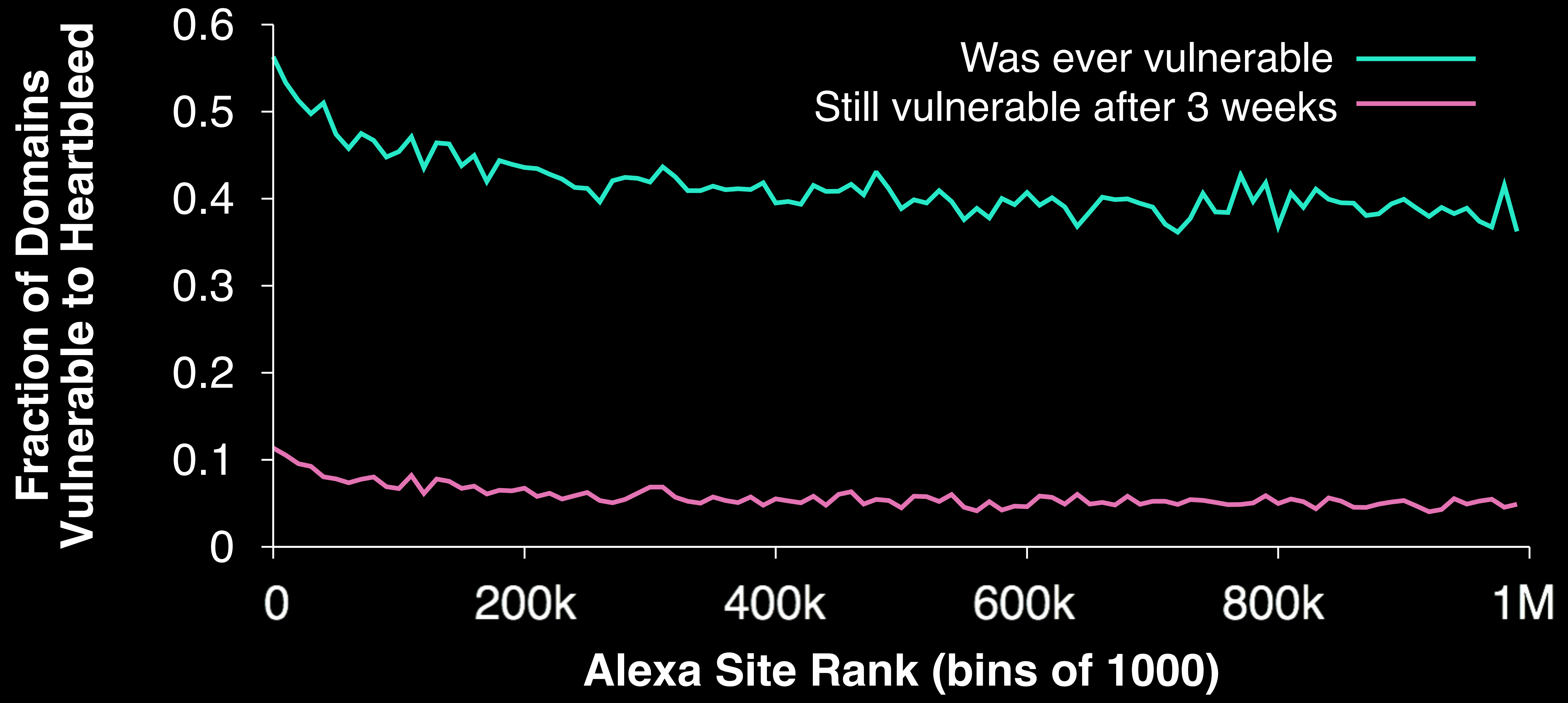
validate

Leaf Set

628k certs
165k domains

- Download CRLs
- Detect vulnerability
- Identify *Heartbleed-induced* reissues & revocations

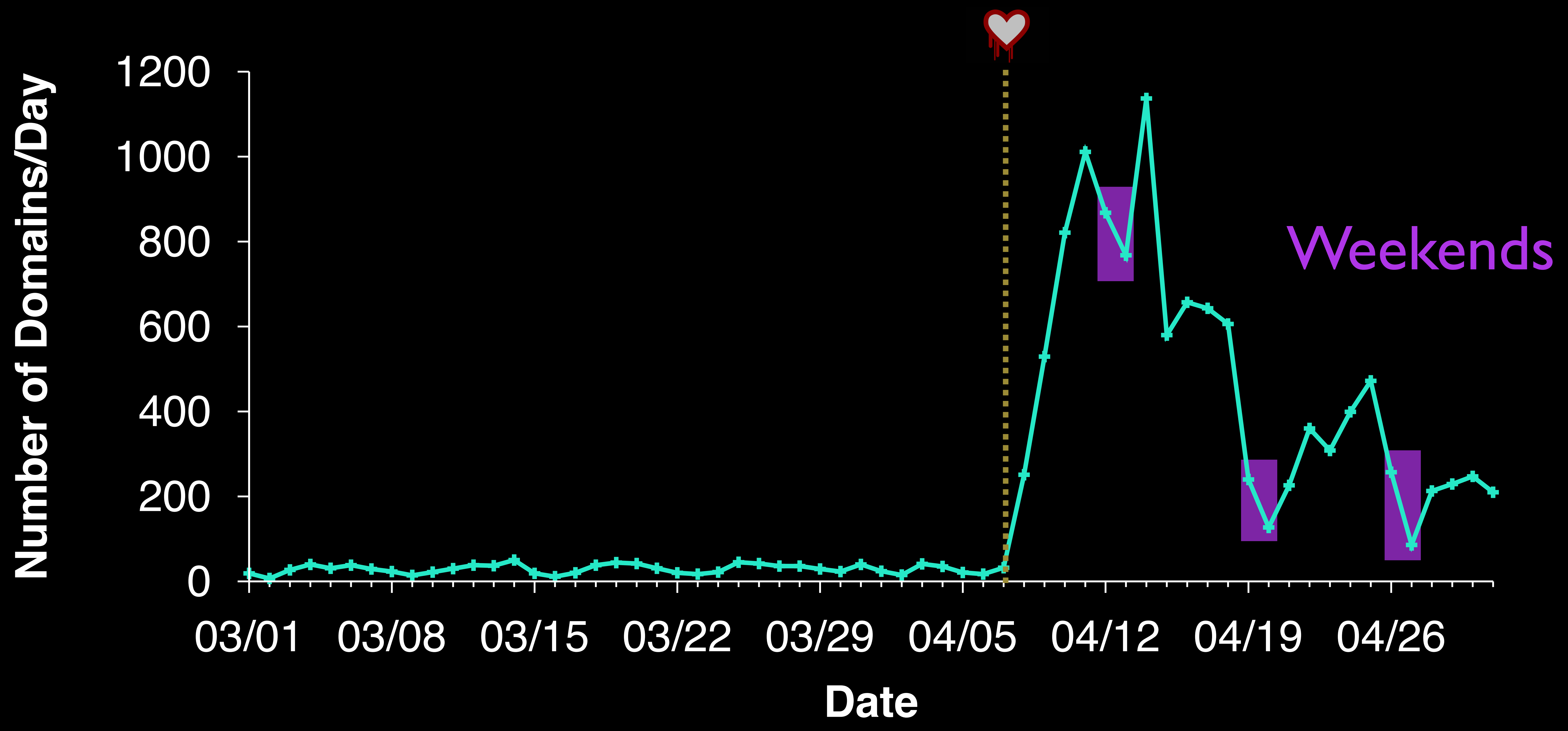
Prevalence and patch rates



Patching rates are mostly positive

Only ~7% had not patched within 3 weeks

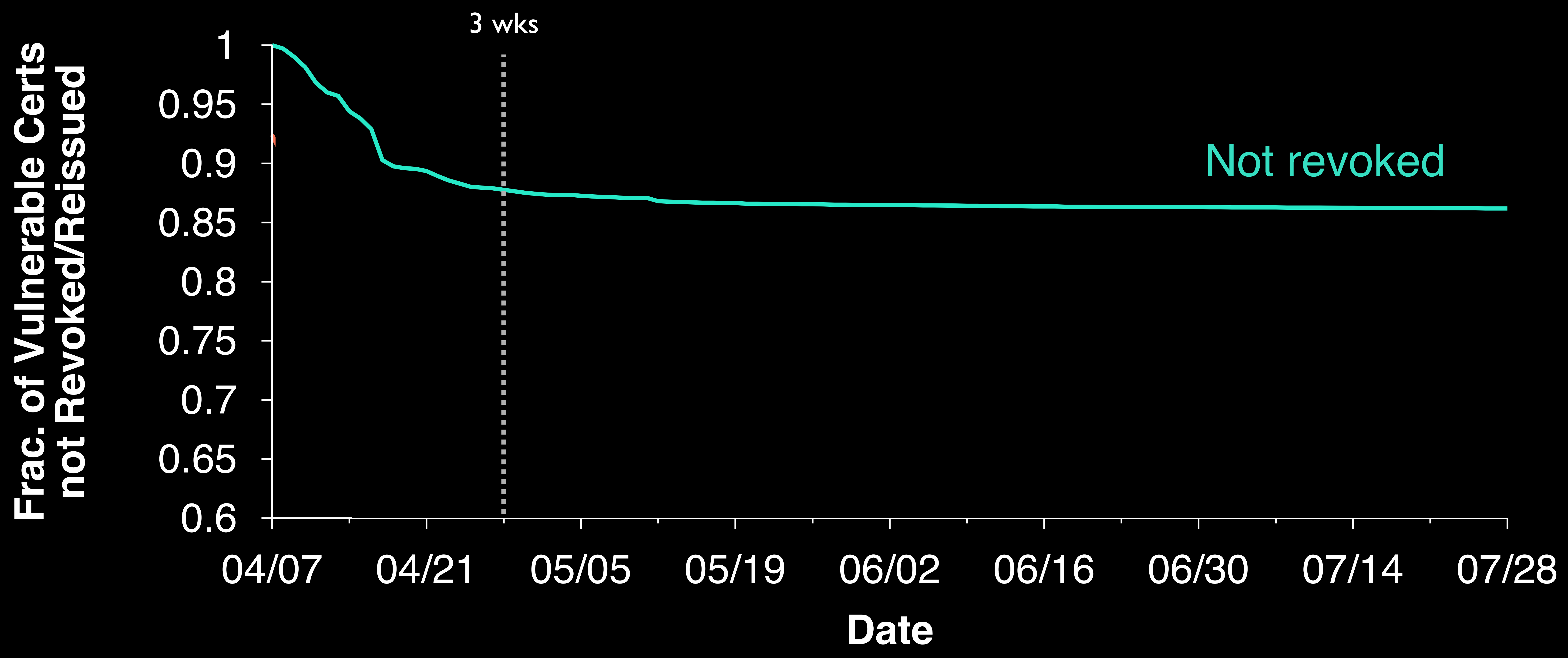
How quickly were certs revoked?



Reaction ramps up quickly

Security takes the weekends off

Certificate update rates

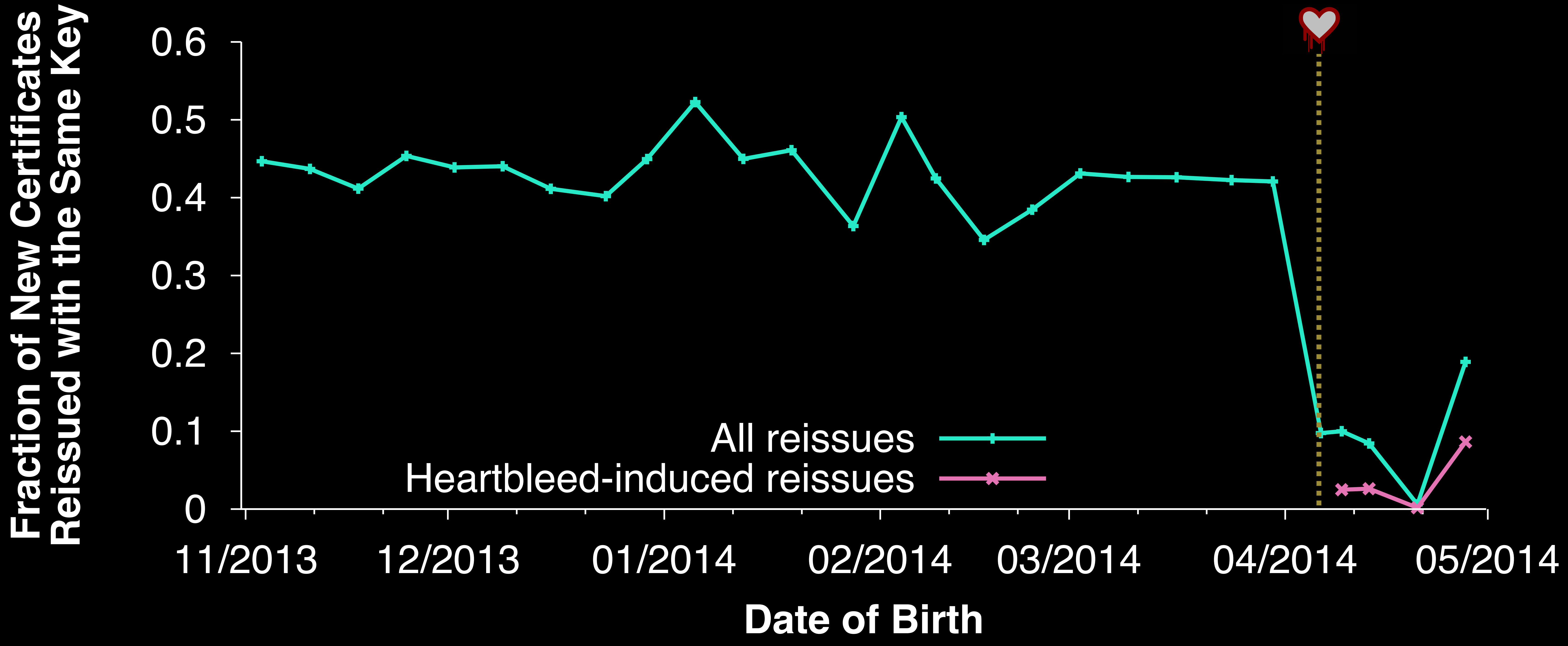


Similar pattern to patches:

Exponential drop-off, then levels out

After 3 weeks: 13% Revoked 27% Reissued

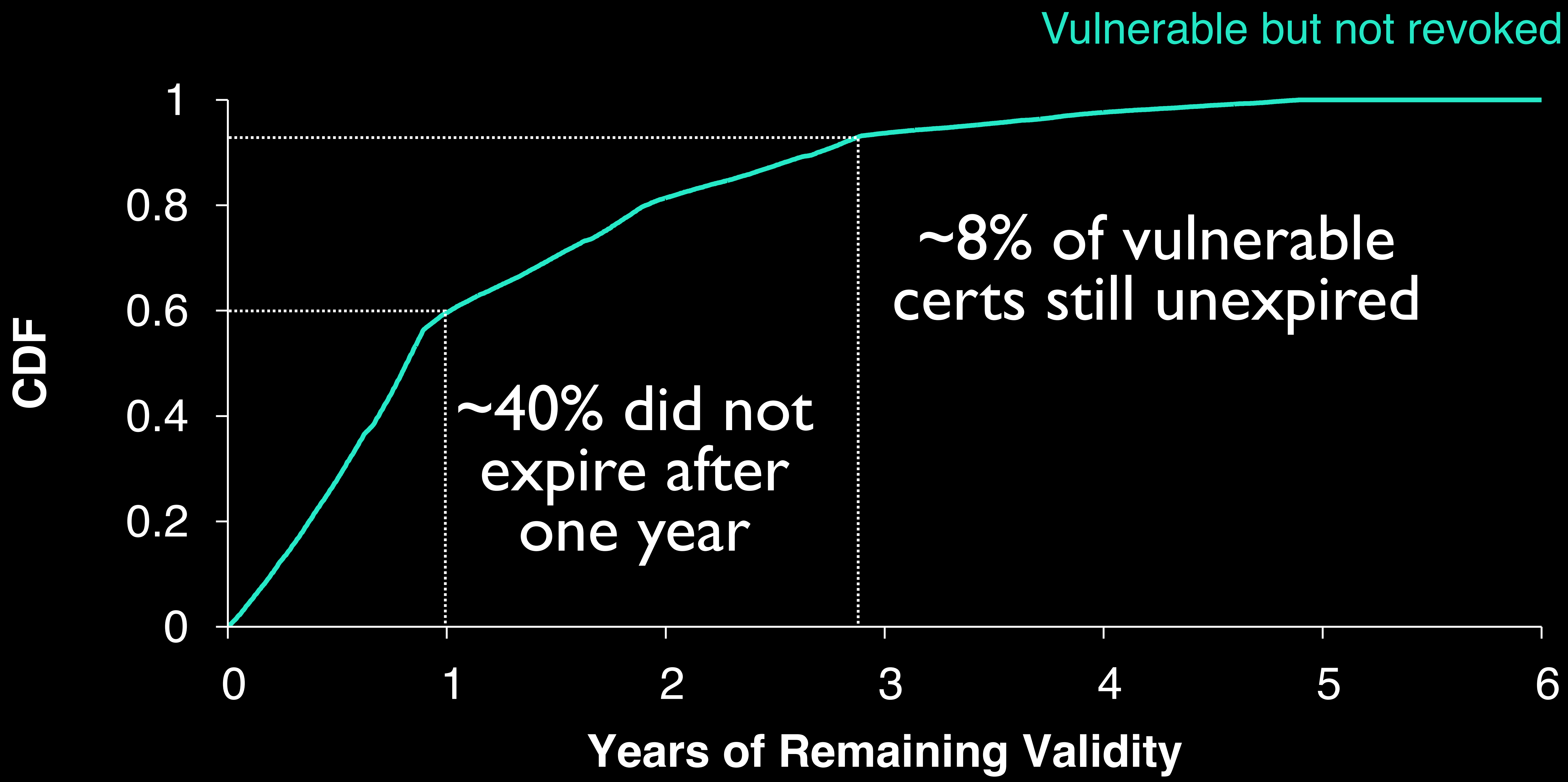
Reissue ⇒ New key?



Reissuing the same key is common practice

4.1% Heartbleed-induced

Can we wait for expiration?



~40% did not expire after one year

~8% of vulnerable certs still unexpired

We may be dealing with Heartbleed for *years*

Certificate revocation

is a critical part of any PKI

Administrators must revoke and reissue as quickly as possible

Browsers/OSes should obtain revocations as quickly as possible

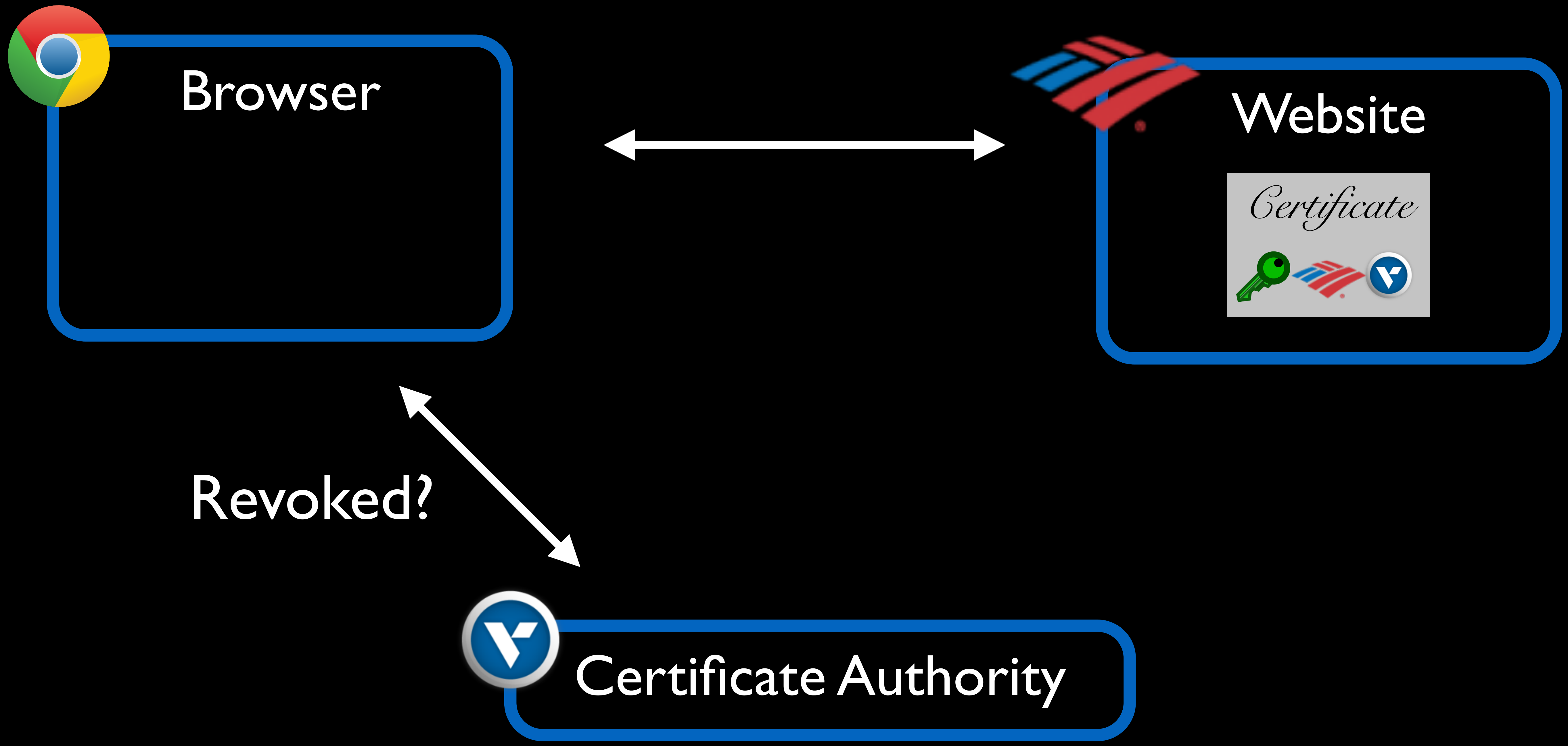
An End-to-End Measurement of Certificate Revocation in the Web's PKI

Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs,
Alan Mislove, Aaron Schulman, Christo Wilson

Browsers/OSes should obtain revocations as quickly as possible

ACM IMC 2015

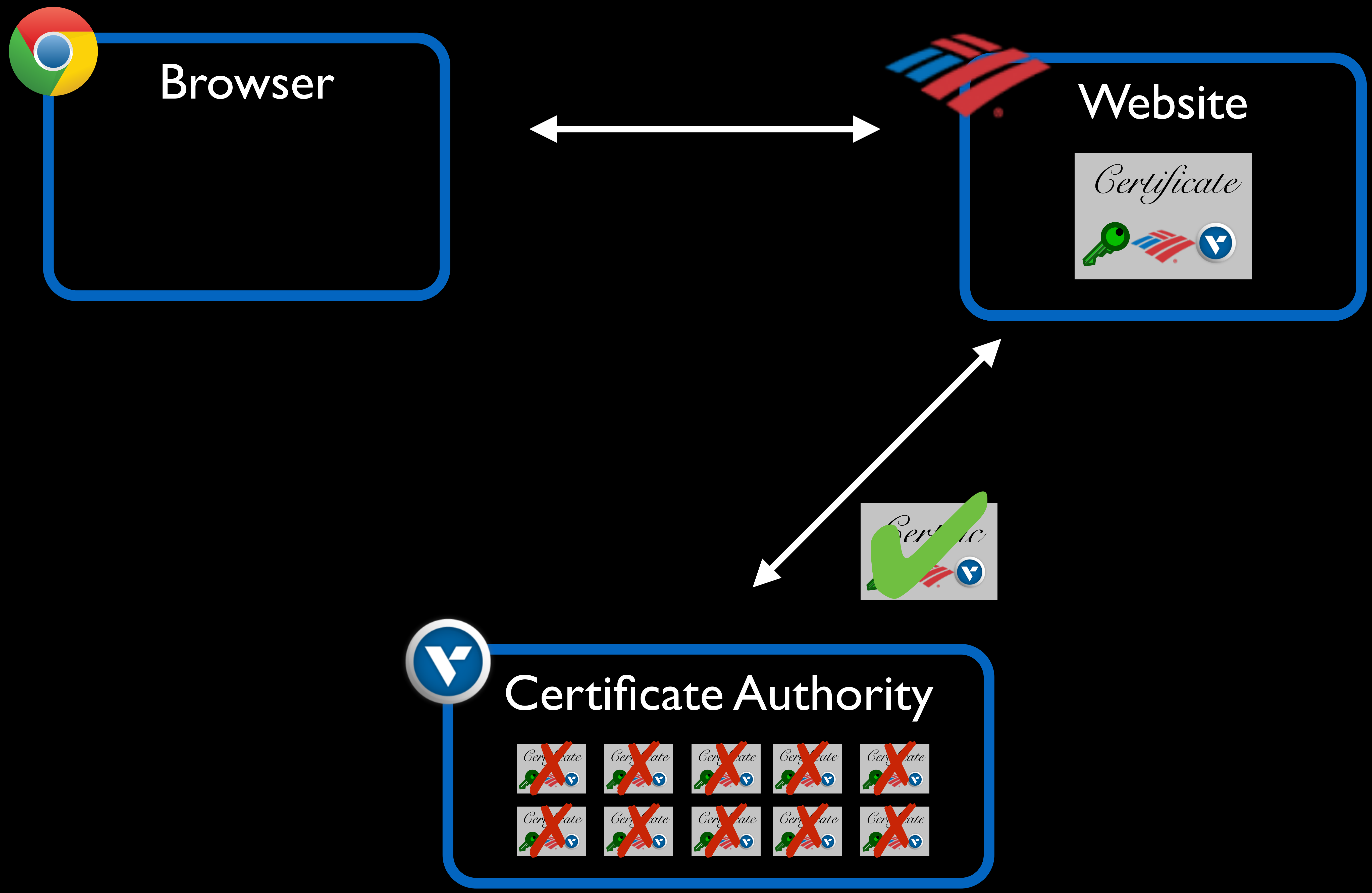
Security is an economic concern



Browsers face tension between security and page load times

CAs face tension between security and bandwidth costs

OCSP Stapling



But OCSP Stapling rarely activated by admins:

Our scan: 3% of normal certs; 2% of EV certs

Testing browser behavior

Revocation protocols

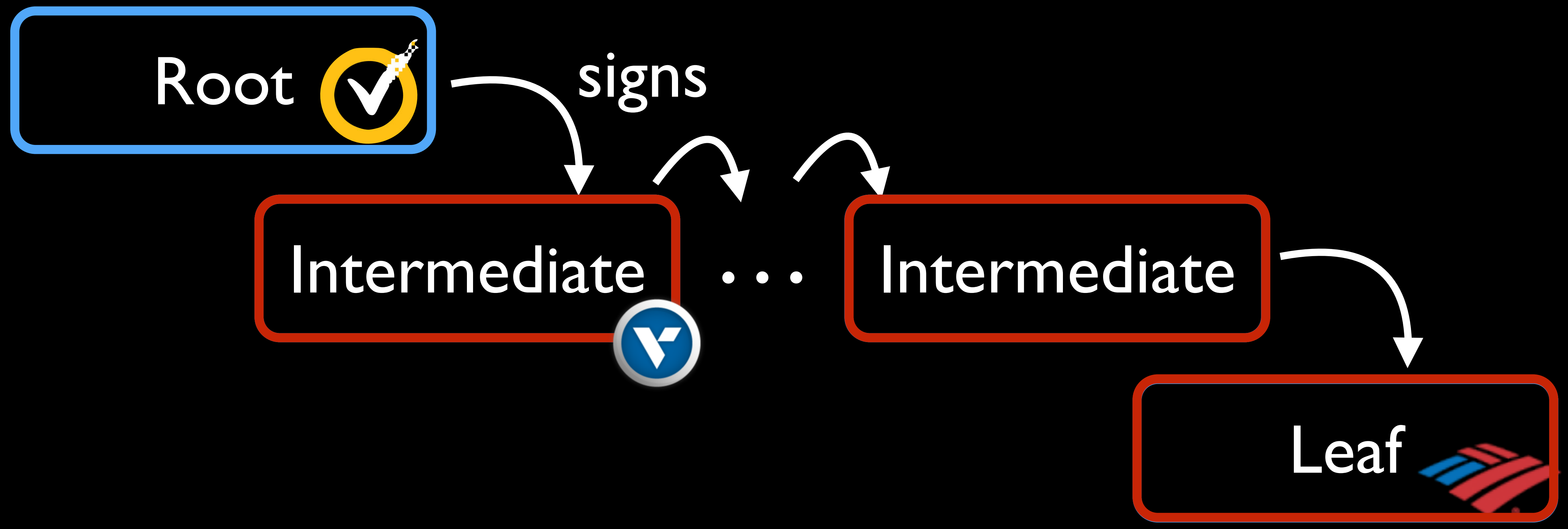
- Browsers *should* support all major protocols
    - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
    - E.g., because the OCSP server is down

Chain lengths

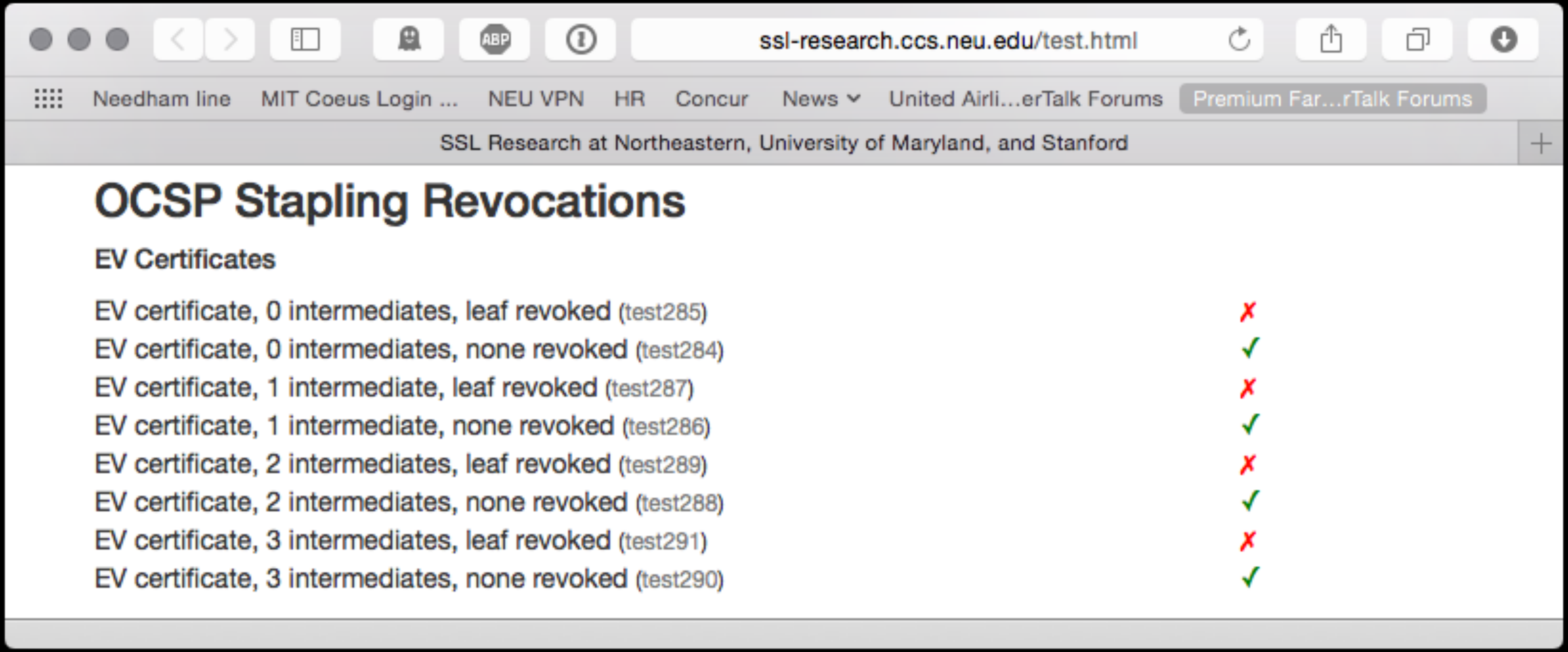
- Browsers *should* reject a cert if *any* on the chain fail
    - Leaf, intermediate(s), root



Test harness

Implemented 192 tests using fake root certificate + Javascript

- Unique DNS name, cert chain, CRL/OCSP responder, …



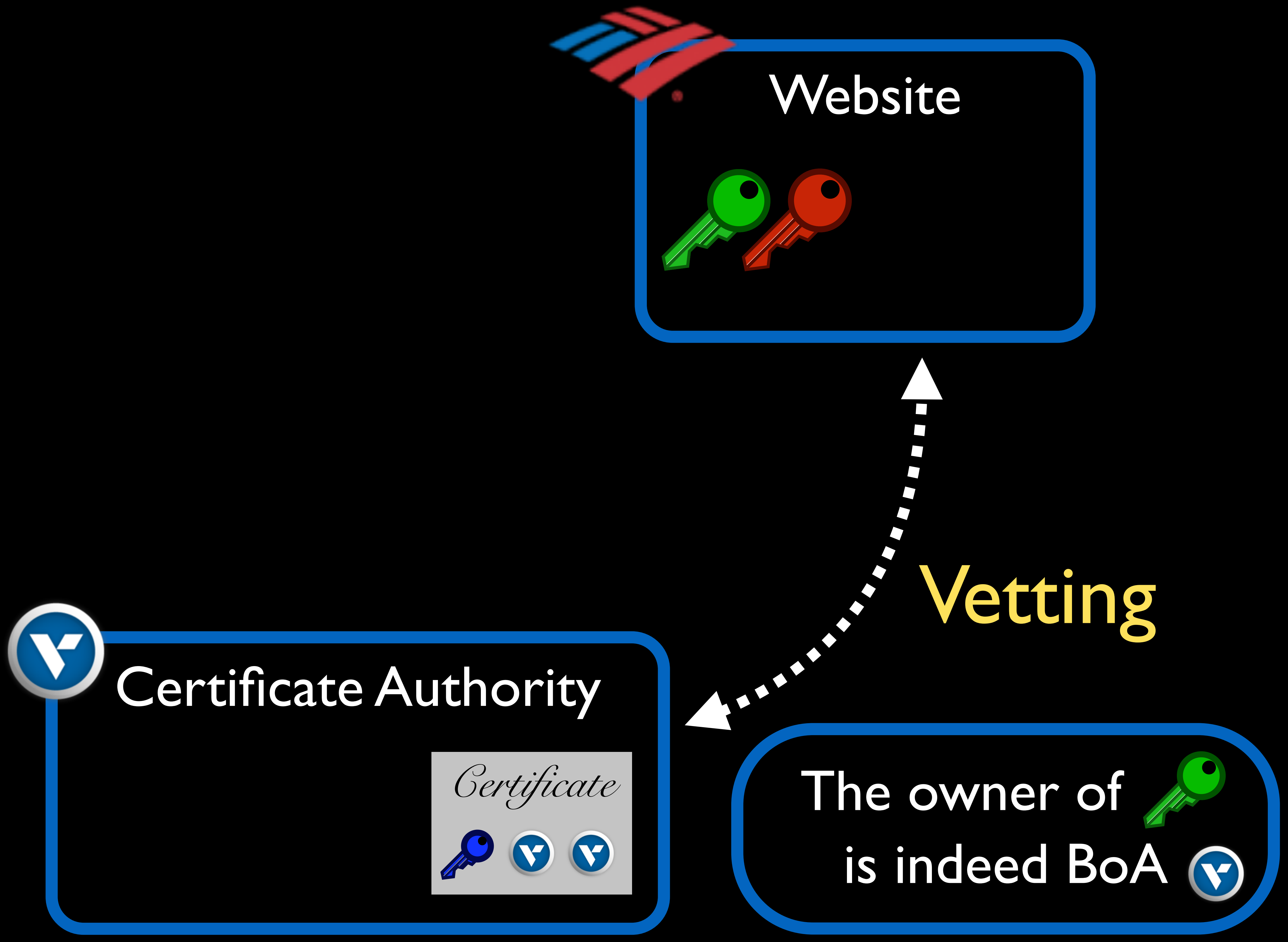
OCSP Stapling Revocations

EV Certificates

EV certificate, 0 intermediates, leaf revoked (test285)	✗
EV certificate, 0 intermediates, none revoked (test284)	✓
EV certificate, 1 intermediate, leaf revoked (test287)	✗
EV certificate, 1 intermediate, none revoked (test286)	✓
EV certificate, 2 intermediates, leaf revoked (test289)	✗
EV certificate, 2 intermediates, none revoked (test288)	✓
EV certificate, 3 intermediates, leaf revoked (test291)	✗
EV certificate, 3 intermediates, none revoked (test290)	✓

EV Certificates

More thorough vetting process of CAs and clients



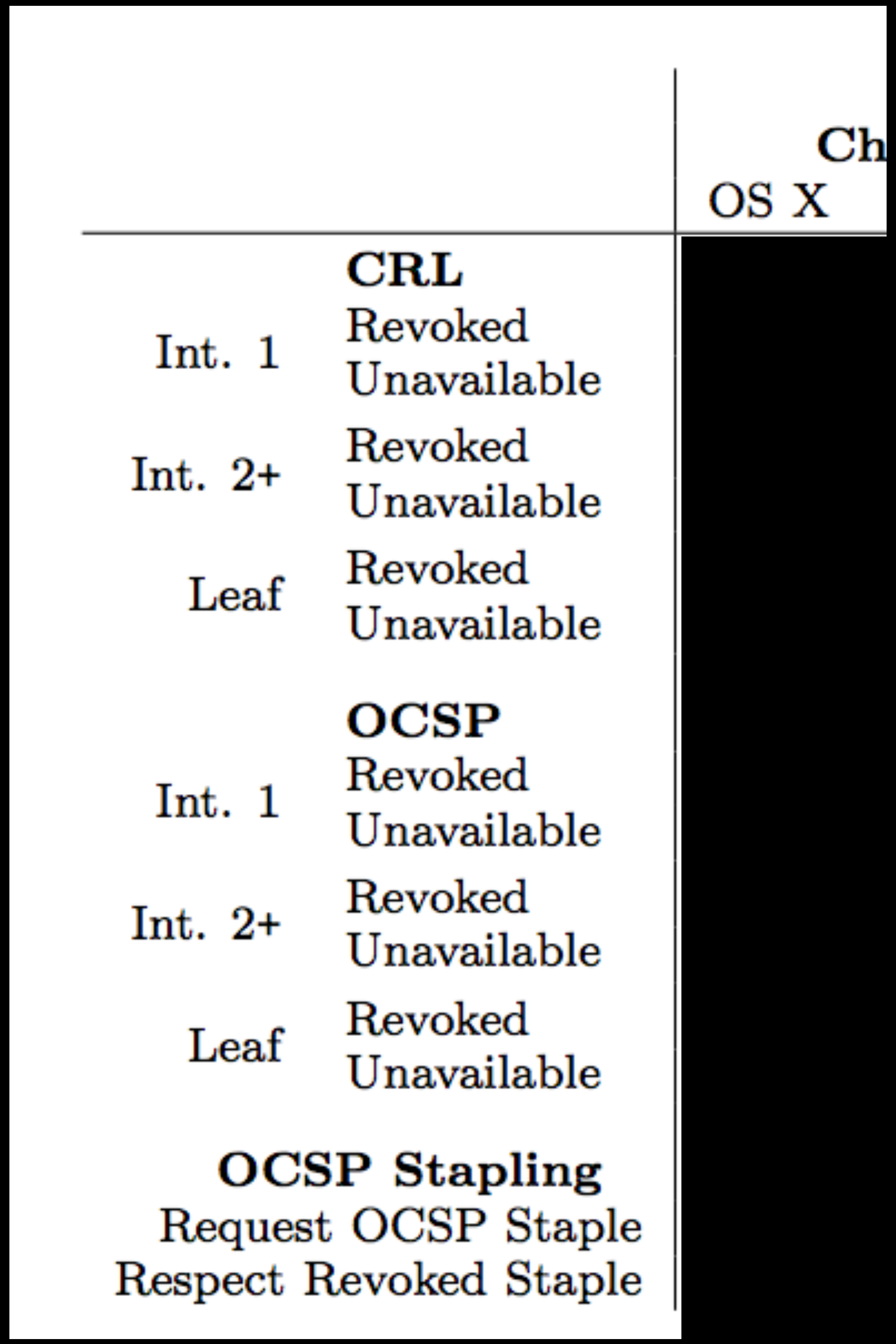
Website

Vetting

Certificate Authority

The owner of is indeed BoA

Results across all browsers



		Ch OS X
	CRL	
Int. 1	Revoked	
	Unavailable	
Int. 2+	Revoked	
	Unavailable	
Leaf	Revoked	
	Unavailable	
	OCSP	
Int. 1	Revoked	
	Unavailable	
Int. 2+	Revoked	
	Unavailable	
Leaf	Revoked	
	Unavailable	
OCSP Stapling		
Request OCSP Staple		
Respect Revoked Staple		

Mobile Browsers

✔ Passes test

✗ Fails test

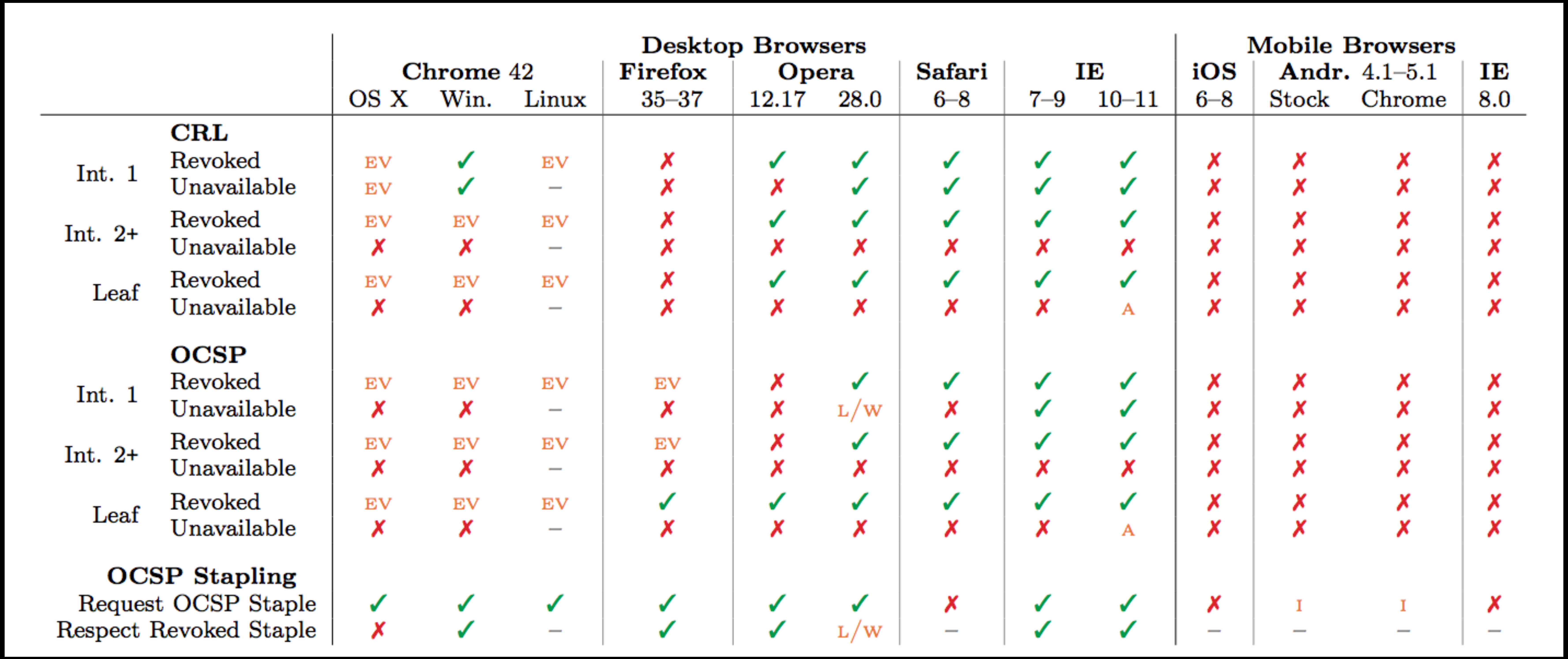
EV Passes for EV certs

I Ignores OCSP Staple

A Pops up alert to user

L/W Passes on Linux/Win.

Results across all browsers



		Desktop Browsers									Mobile Browsers			
		Chrome 42			**Firefox**	**Opera**		**Safari**	**IE**		**iOS**	**Andr.** 4.1–5.1		**IE**
		OS X	Win.	Linux	35–37	12.17	28.0	6–8	7–9	10–11	6–8	Stock	Chrome	8.0
	CRL													
Int. 1	Revoked	EV	✓	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	EV	✓	–	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
Int. 2+	Revoked	EV	EV	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Leaf	Revoked	EV	EV	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	A	✗	✗	✗	✗
	OCSP													
Int. 1	Revoked	EV	EV	EV	EV	✗	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	L/W	✗	✓	✓	✗	✗	✗	✗
Int. 2+	Revoked	EV	EV	EV	EV	✗	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Leaf	Revoked	EV	EV	EV	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	A	✗	✗	✗	✗
OCSP Stapling														
Request OCSP Staple		✓	✓	✓	✓	✓	✓	✗	✓	✓	✗	I	I	✗
Respect Revoked Staple		✗	✓	–	✓	✓	L/W	–	✓	✓	–	–	–	–

Browser developers are not
doing what the PKI needs them to do

✓ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Certificate revocation
is a critical part of any PKI

Administrators must revoke and reissue
as quickly as possible

Browsers/OSes should obtain revocations
as quickly as possible

Certificate revocation

is a critical part of any PKI

Administrators must revoke and reissue as quickly as possible

Browsers/OSes should obtain revocations as quickly as possible

CDNs should… what are they doing here?

CDNs should… what are they doing here?

Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

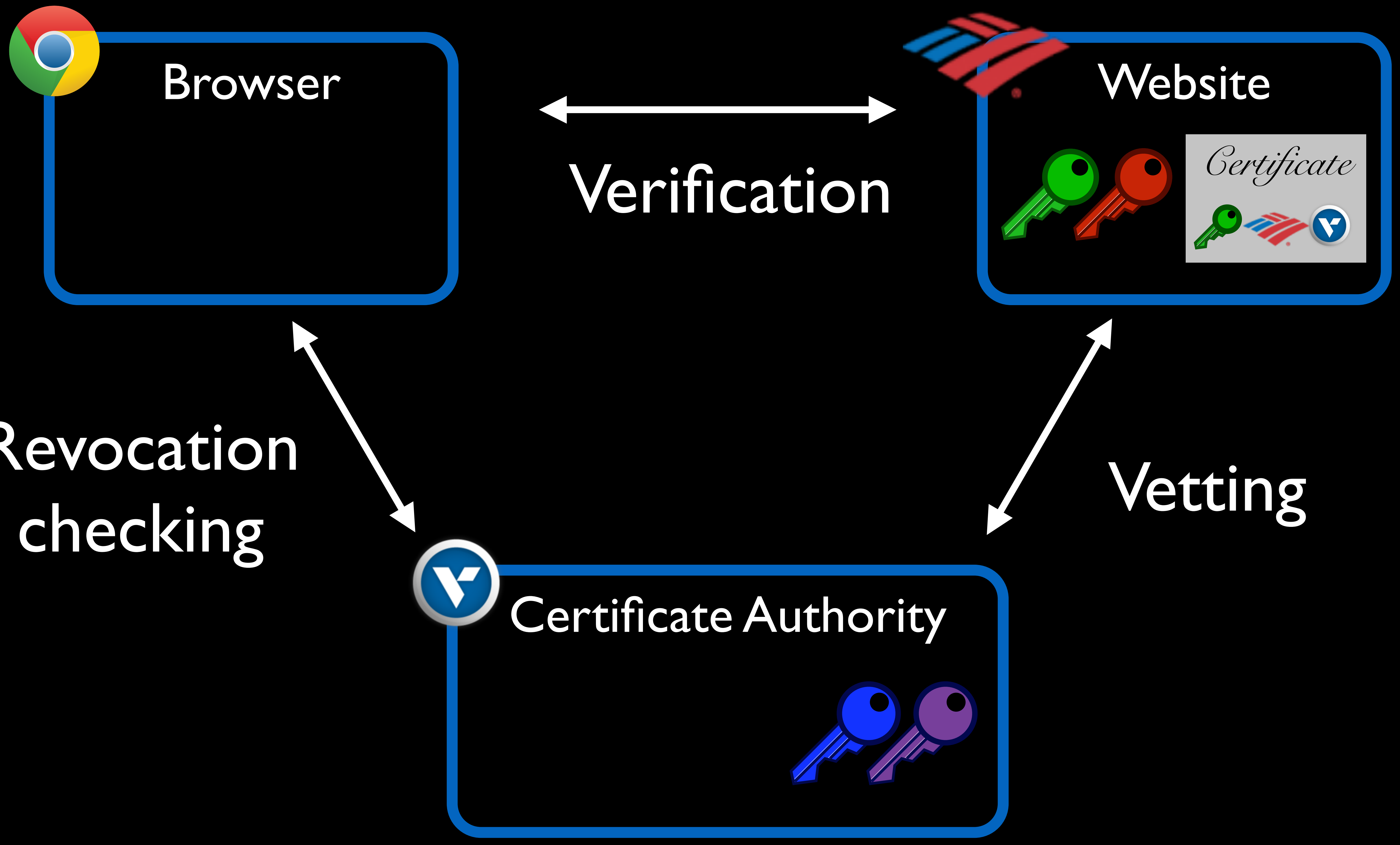
Frank Cangialosi, Taejoong Chung, David Choffnes,
Dave Levin, Bruce M. Maggs, Alan Mislove, Christo Wilson

CCS 2016

Public Key Infrastructures (PKIs)

How can users truly know with whom they are communicating?

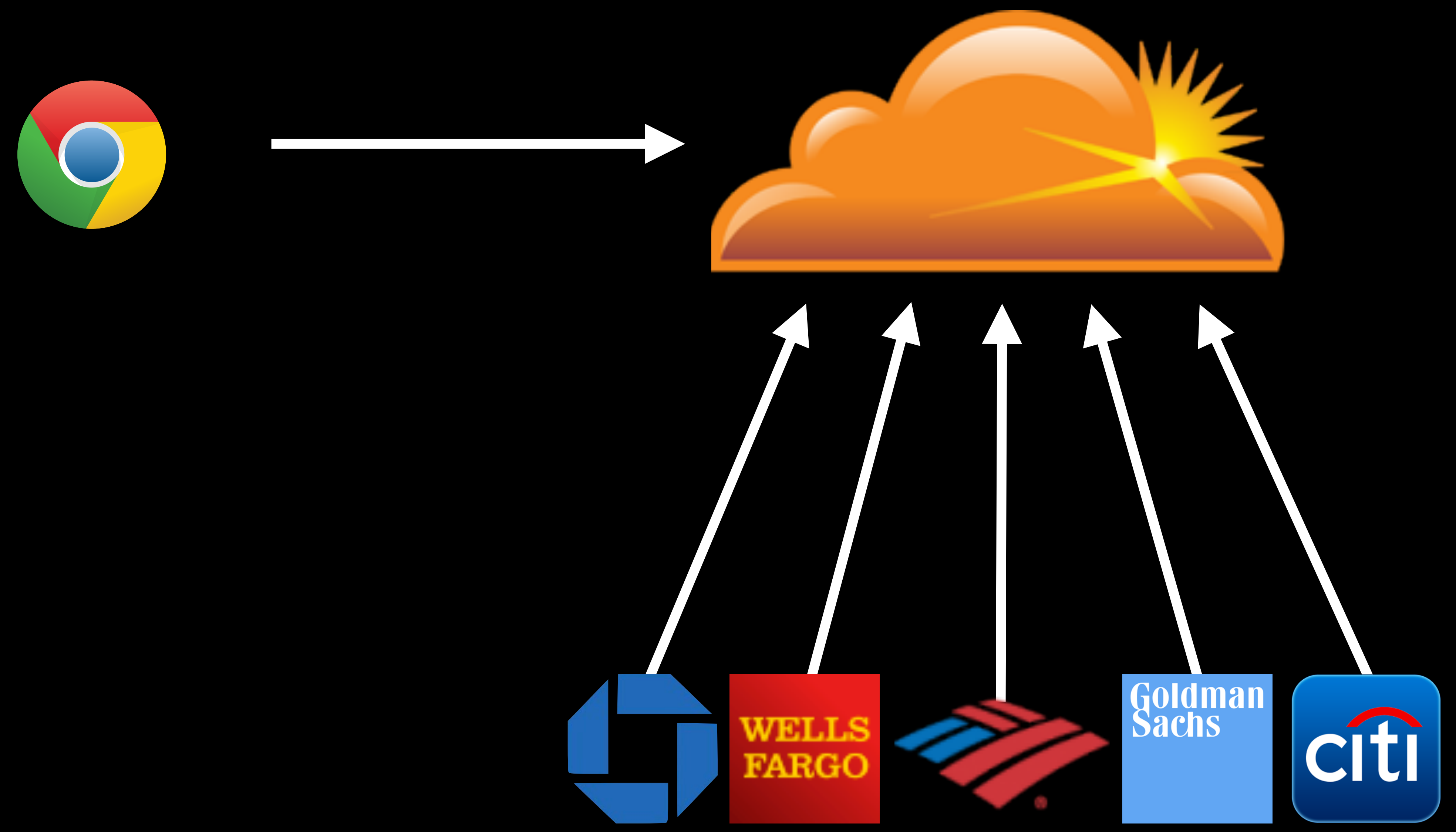
The only one who knows Alice's private key is Alice



The PKI in today’s web

Rare!

The PKI in today's web

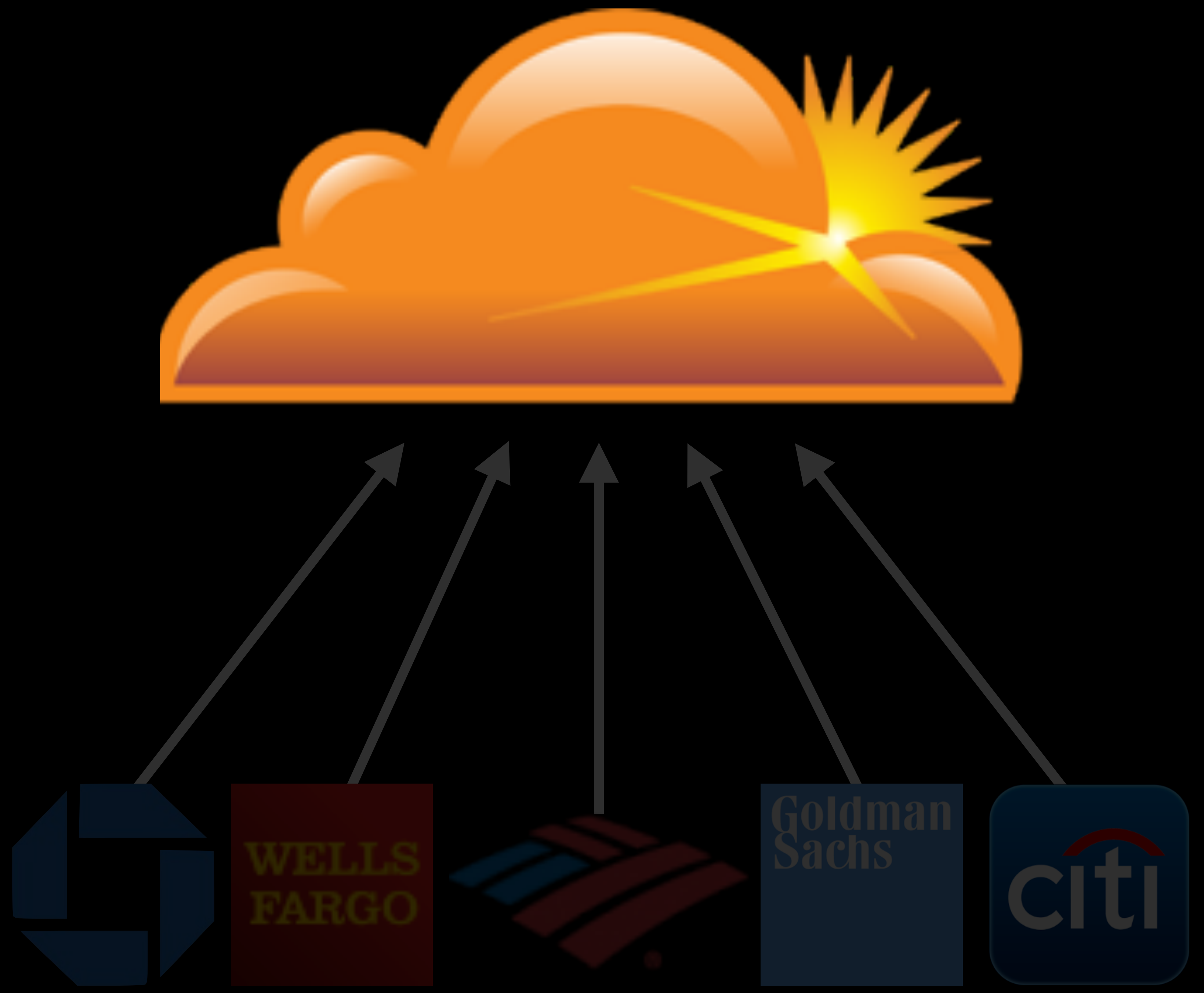


Third-party Hosting Providers

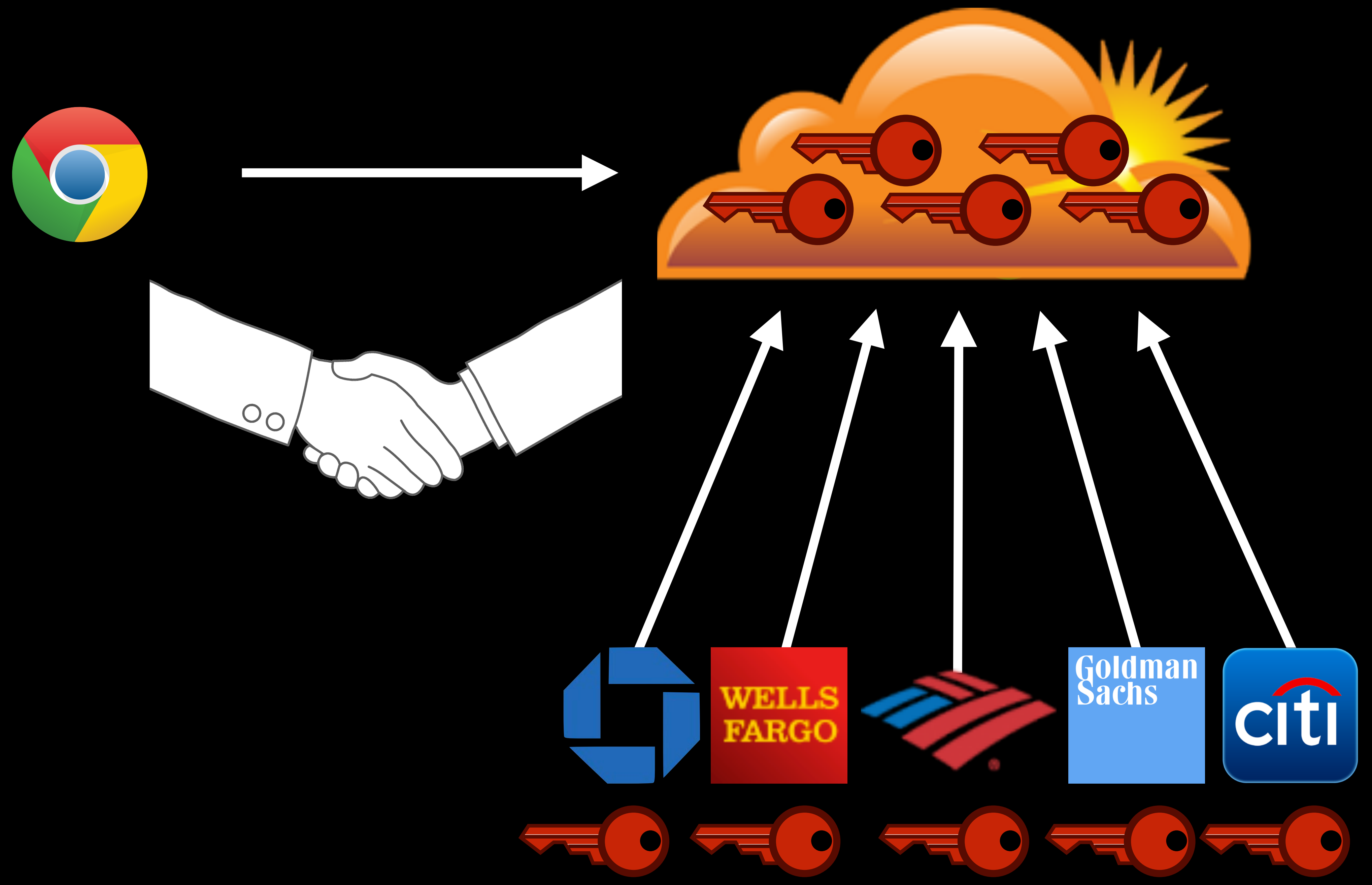
- Content delivery networks
- Web hosting services
- Cloud providers

Varying levels of involvement

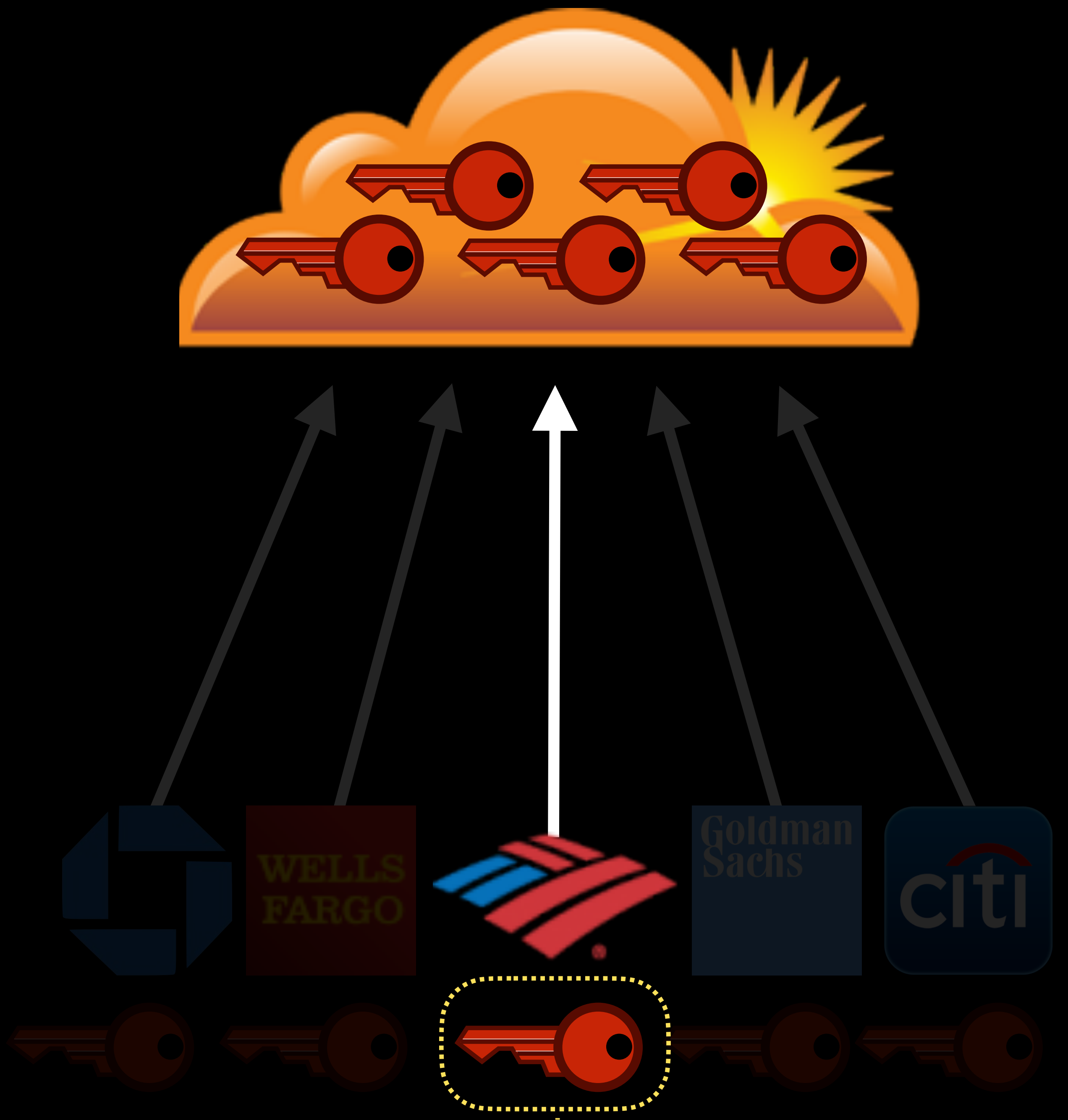
But all trusted to deliver content



The PKI in today's web



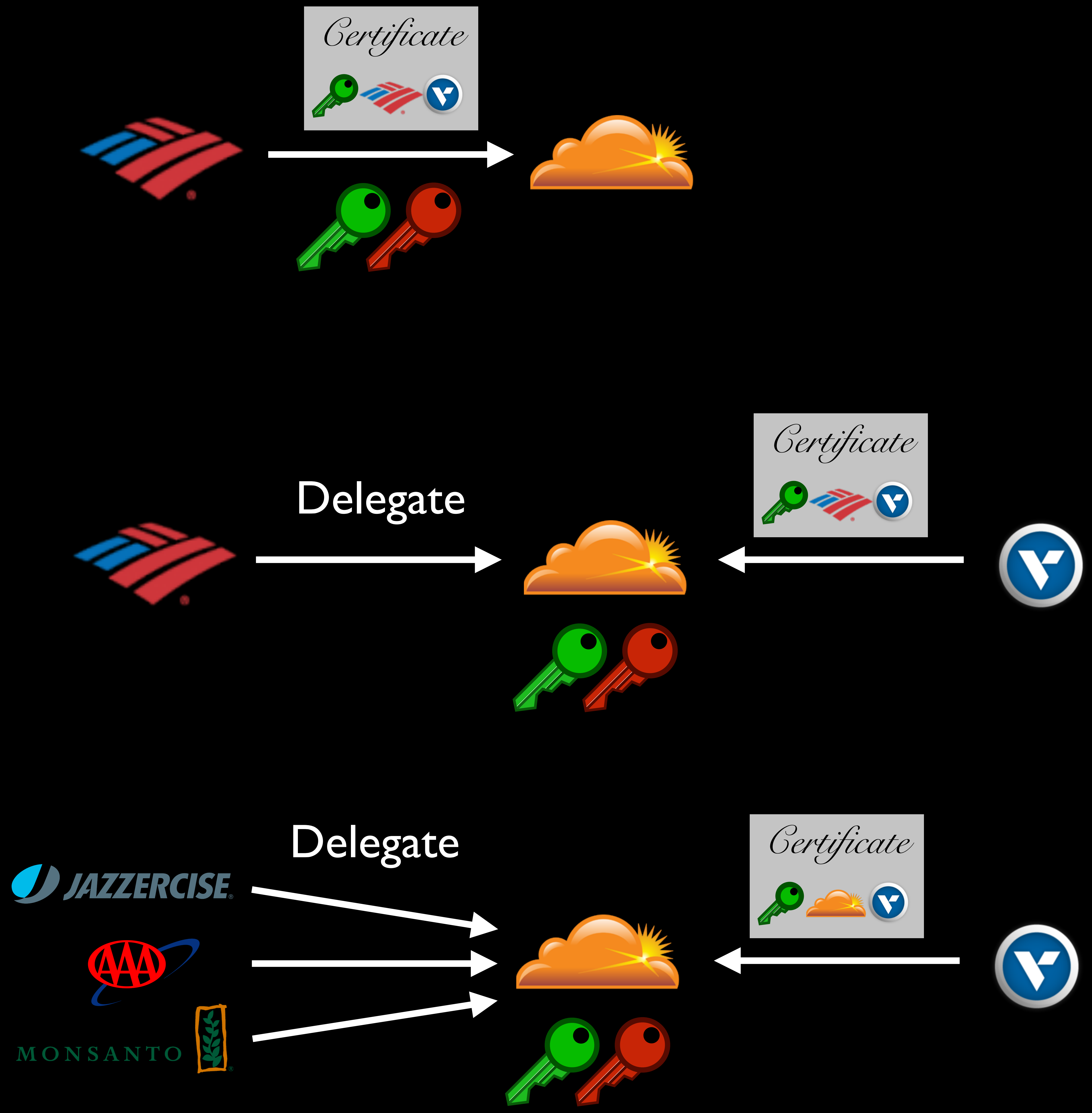
Third-party hosting providers
know their customers' private keys



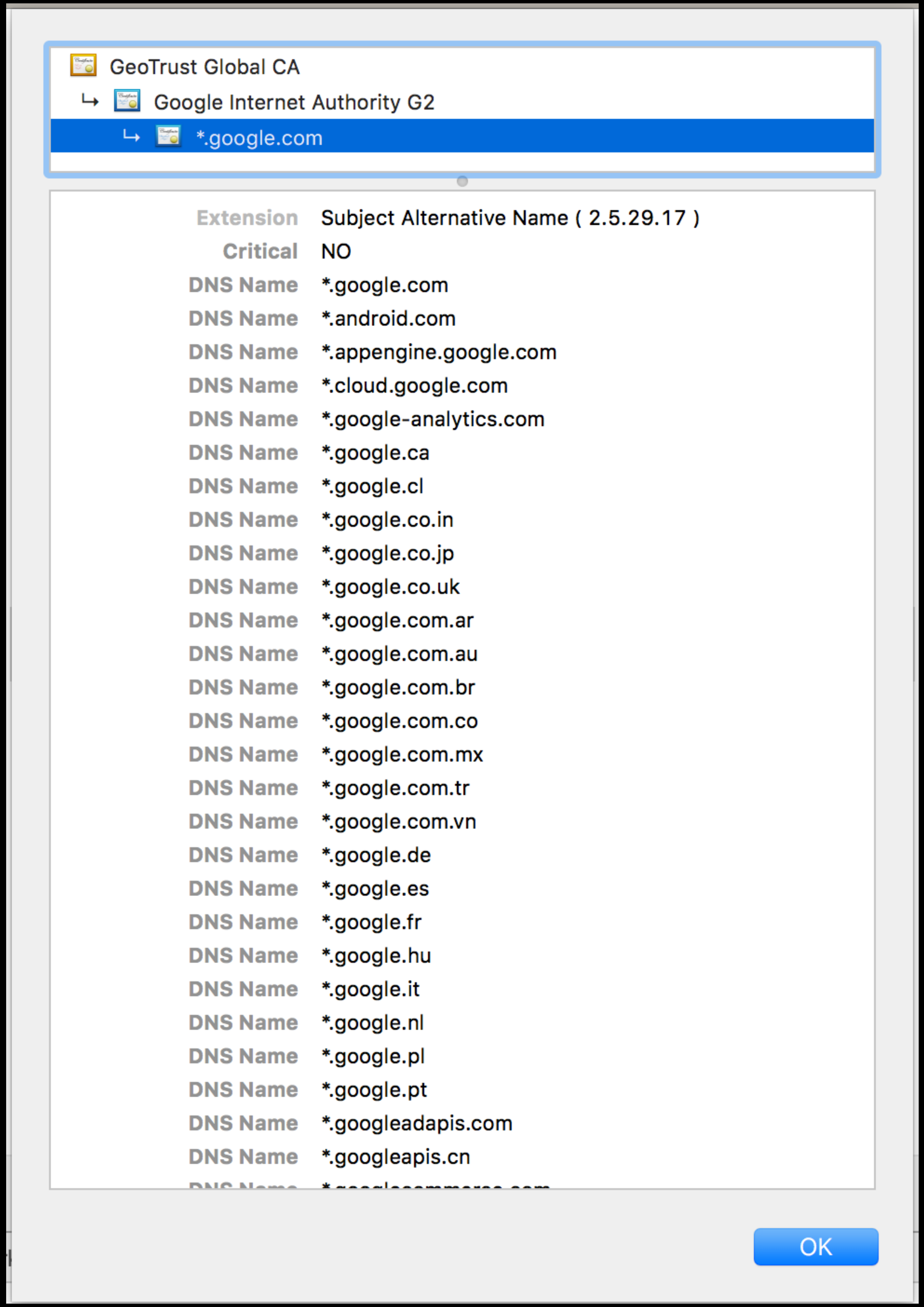
Authentication **fundamentally** assumes:

Only **knows** 🔑

How are keys shared?



Subject Alternate Name (SAN) Lists

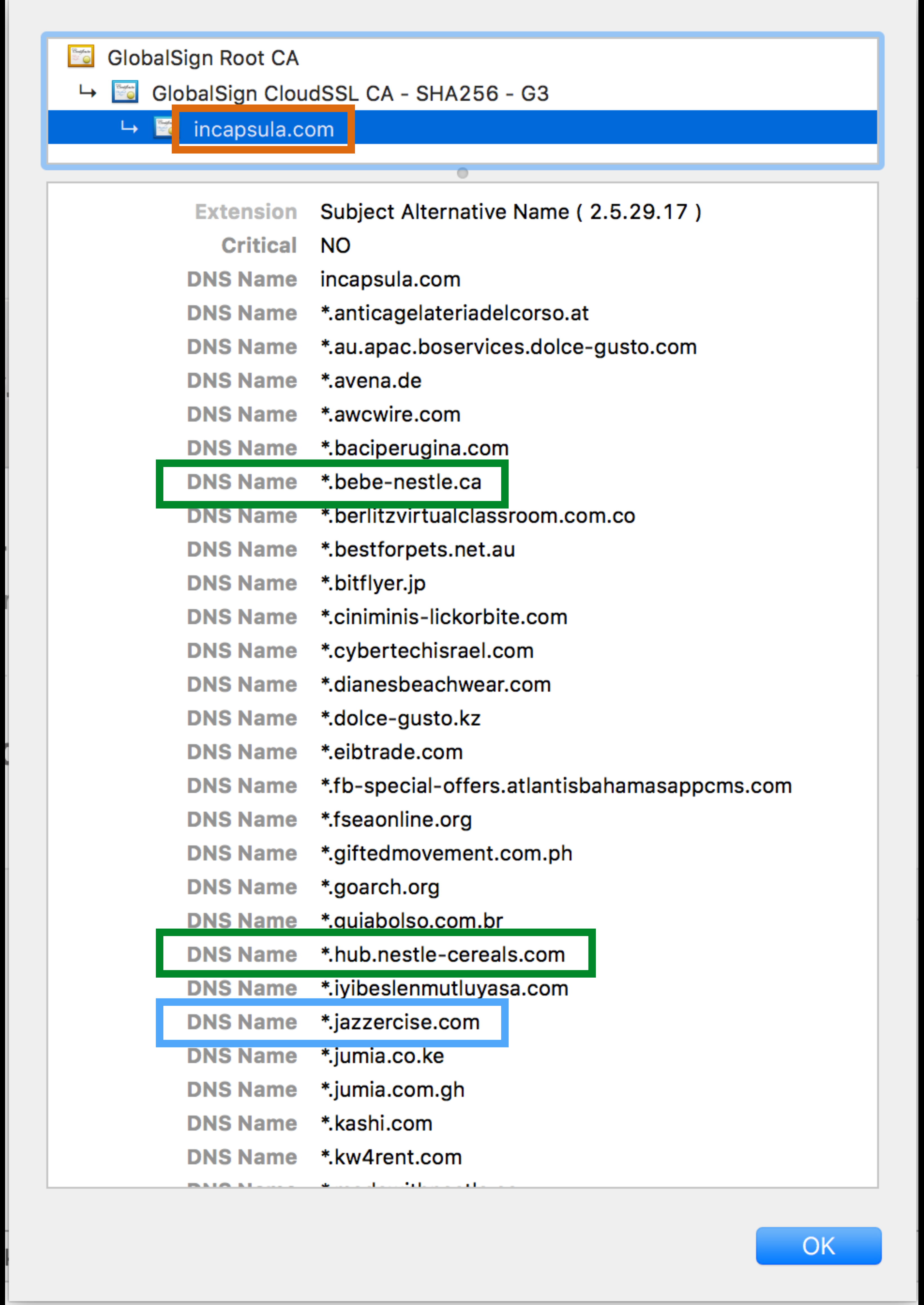


GeoTrust Global CA
↳ Google Internet Authority G2
↳ *.google.com

Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	*.google.com
DNS Name	*.android.com
DNS Name	*.appengine.google.com
DNS Name	*.cloud.google.com
DNS Name	*.google-analytics.com
DNS Name	*.google.ca
DNS Name	*.google.cl
DNS Name	*.google.co.in
DNS Name	*.google.co.jp
DNS Name	*.google.co.uk
DNS Name	*.google.com.ar
DNS Name	*.google.com.au
DNS Name	*.google.com.br
DNS Name	*.google.com.co
DNS Name	*.google.com.mx
DNS Name	*.google.com.tr
DNS Name	*.google.com.vn
DNS Name	*.google.de
DNS Name	*.google.es
DNS Name	*.google.fr
DNS Name	*.google.hu
DNS Name	*.google.it
DNS Name	*.google.nl
DNS Name	*.google.pl
DNS Name	*.google.pt
DNS Name	*.googleadapis.com
DNS Name	*.googleapis.cn

Spirit: Multiple names for the *same organization*

Subject Alternate Name (SAN) Lists



GlobalSign Root CA
↳ GlobalSign CloudSSL CA - SHA256 - G3
↳ incapsula.com

Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	incapsula.com
DNS Name	*.anticagelateriadelcorso.at
DNS Name	*.au.apac.boservices.dolce-gusto.com
DNS Name	*.avena.de
DNS Name	*.awcwire.com
DNS Name	*.baciperugina.com
DNS Name	*.bebe-nestle.ca
DNS Name	*.berlitzvirtualclassroom.com.co
DNS Name	*.bestforpets.net.au
DNS Name	*.bitflyer.jp
DNS Name	*.ciniminis-lickorbite.com
DNS Name	*.cybertechisrael.com
DNS Name	*.dianesbeachwear.com
DNS Name	*.dolce-gusto.kz
DNS Name	*.eibtrade.com
DNS Name	*.fb-special-offers.atlantisbahamasappcms.com
DNS Name	*.fseaonline.org
DNS Name	*.giftedmovement.com.ph
DNS Name	*.goarch.org
DNS Name	*.guiabolso.com.br
DNS Name	*.hub.nestle-cereals.com
DNS Name	*.iyibeslenmutluyasa.com
DNS Name	*.jazzercise.com
DNS Name	*.jumia.co.ke
DNS Name	*.jumia.com.gh
DNS Name	*.kashi.com
DNS Name	*.kw4rent.com

Spirit: Multiple names for the *same organization*

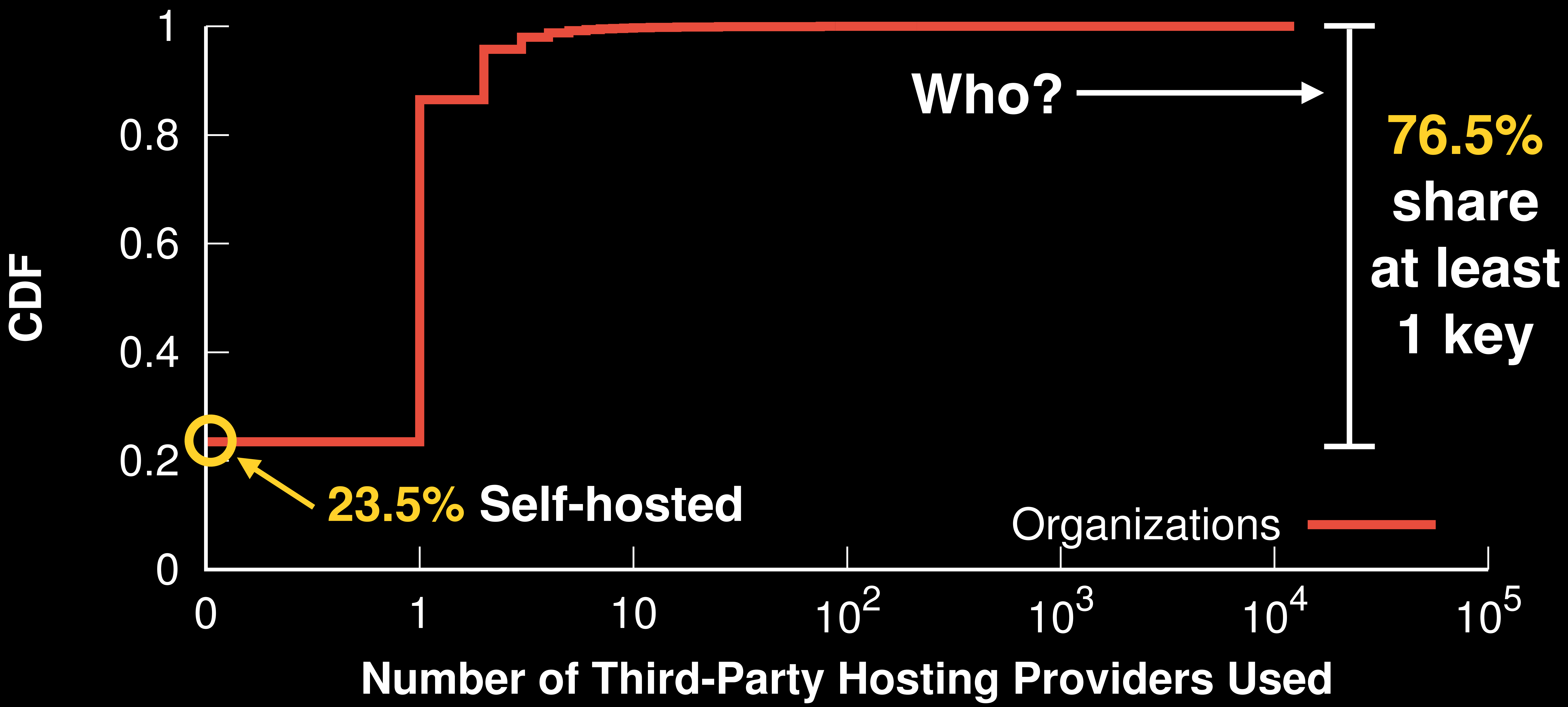
Practice: *Different organizations* lumped together

Cruise-liner Certificate

Who gets the private key?

Who manages it?

How prevalent is key sharing?

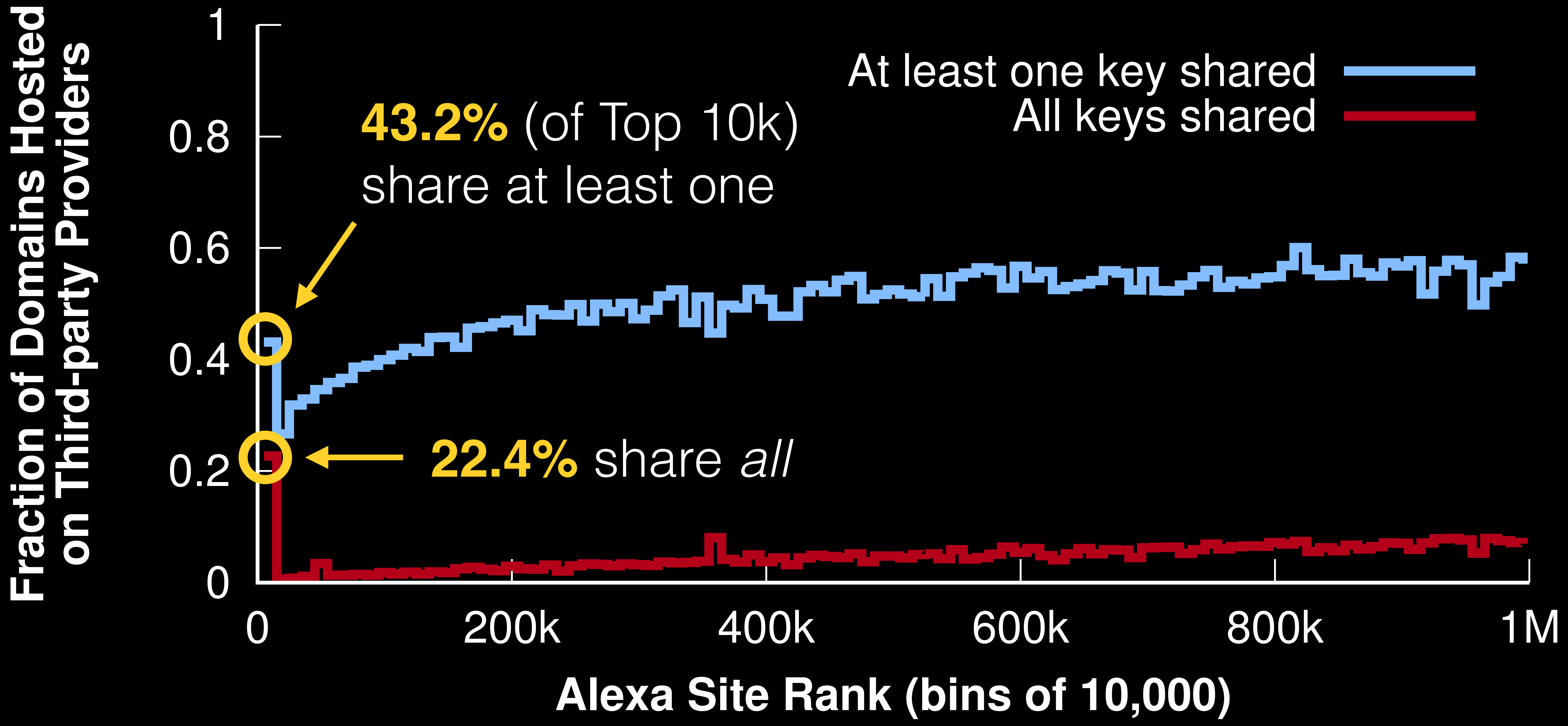


Who?

76.5% share at least 1 key

23.5% Self-hosted

Who shares?

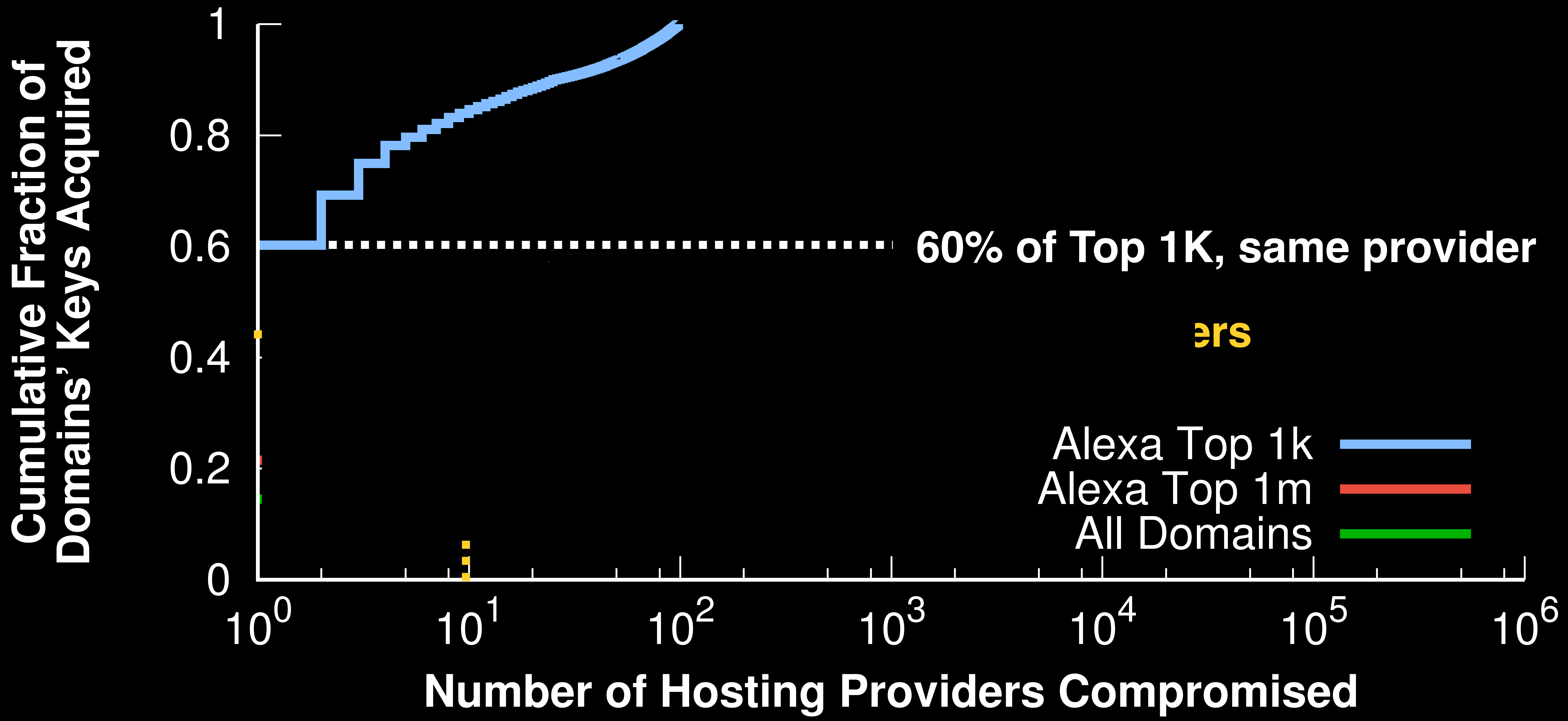


43.2% (of Top 10k) share at least one

22.4% share *all*

Key sharing is common across the Internet

Does key sharing make enticing attack targets?



Popular hosting services are prime targets for attack

POOR CERTIFICATE MANAGEMENT

Websites aren’t properly revoking their certificates

Browsers aren’t properly checking for revocations

Websites aren’t keeping their secret keys secret

Why?

CAs have incentive to introduce disincentives (bandwidth costs)

Websites have disincentive to do the right thing (CAs charge; key management hard)

Browsers have a disincentive to do the right thing (page load times)