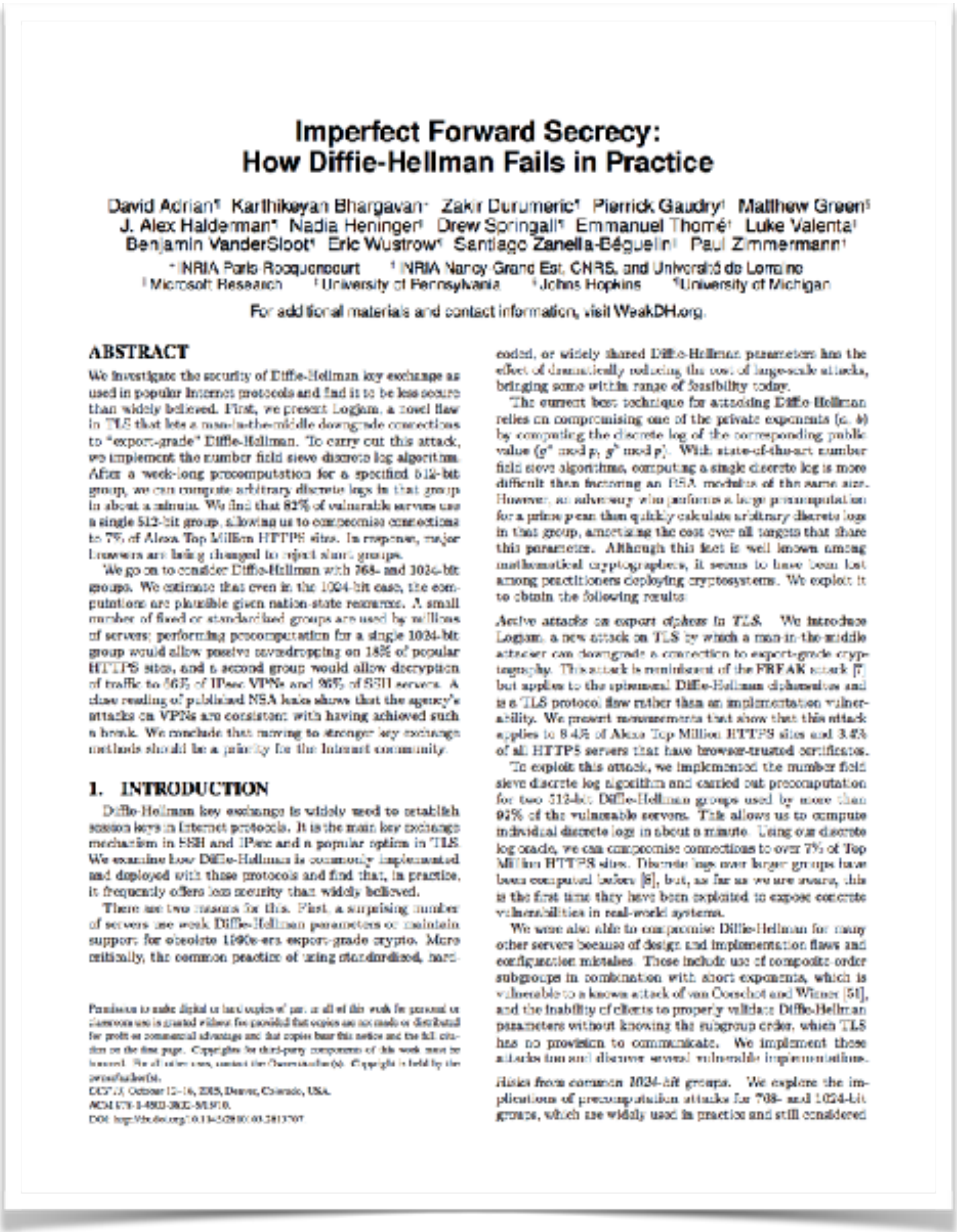
HOW CRYPTO FAILS IN PRACTICE

GRAD SEC

OCT 31 2017

TODAY'S PAPERS



Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

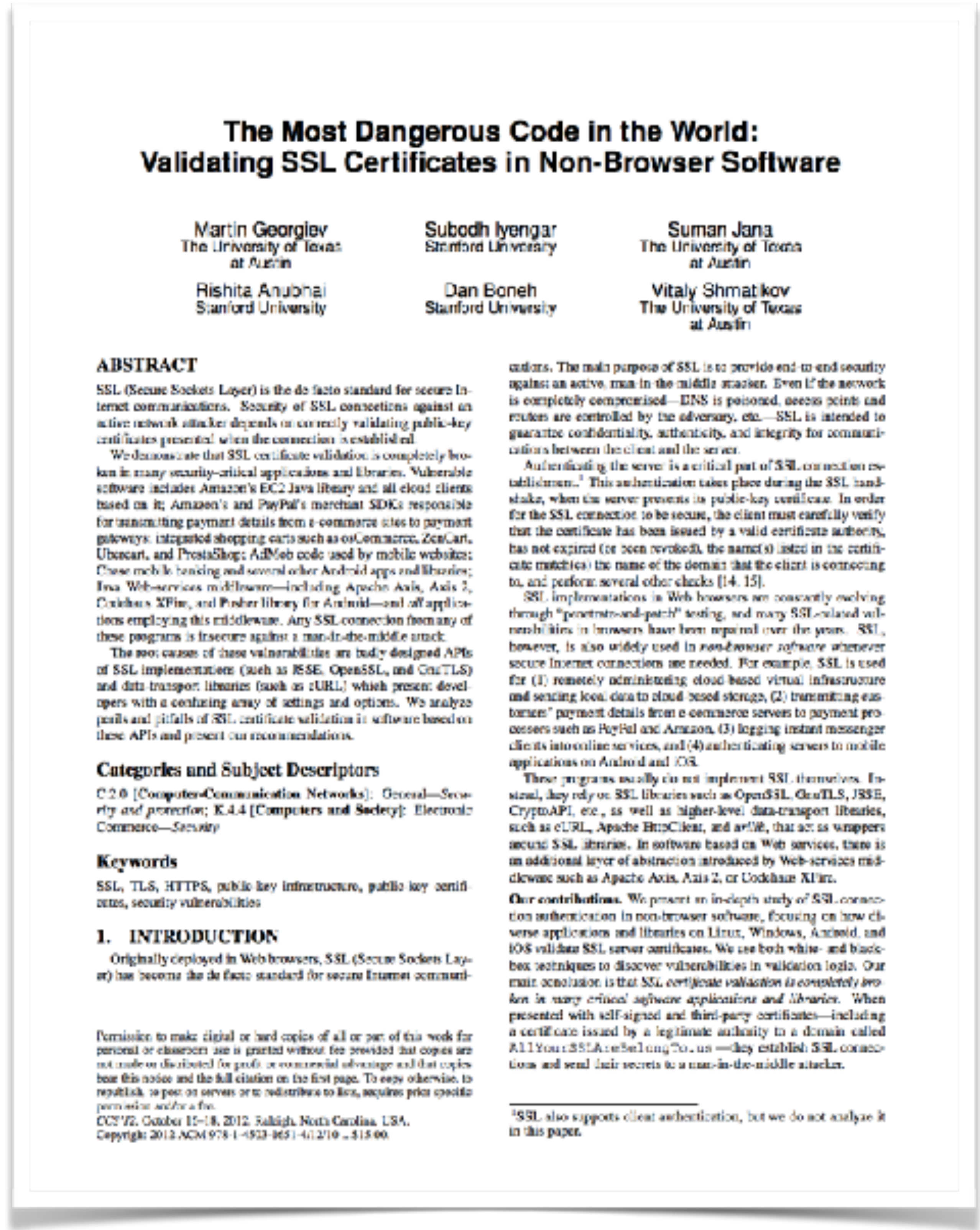
David Adrian Karthikeyan Bhargavan Zakir Durumeric Pierrick Gaudry Matthew Green J. Alex Halderman Nadia Heninger Drew Springall Emmanuel Thomé Luke Valenta Benjamin VanderSloot Eric Wustrow Santiago Zanella-Béguelin Paul Zimmermann

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

1. INTRODUCTION



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

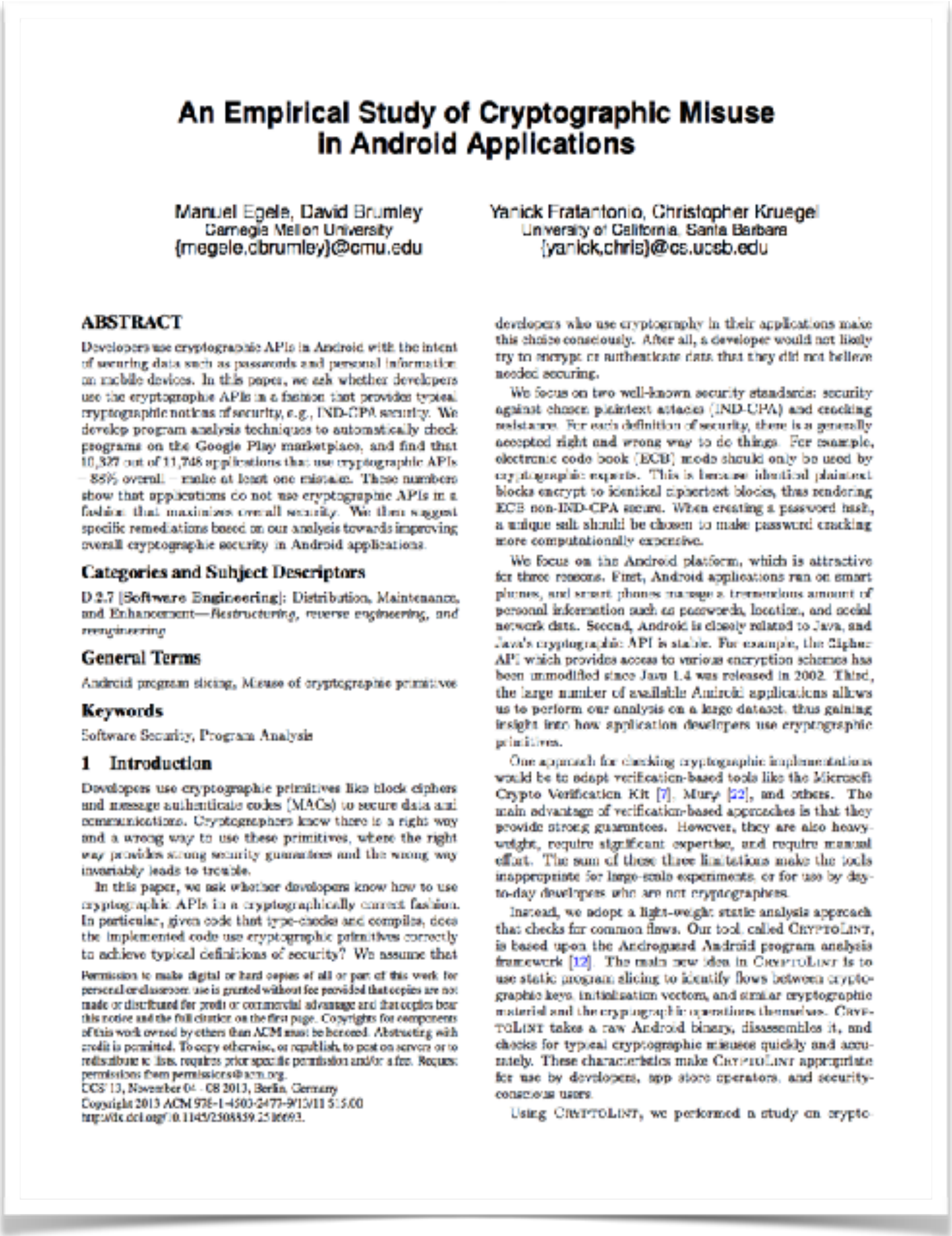
ABSTRACT

Categories and Subject Descriptors

Keywords

1. INTRODUCTION

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 adopt verification-based tools like the Microsoft Crypto Verification Kit [7], Mury [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-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS'13, November 04 - 08 2013, Berlin, Germany
Copyright 2013 ACM 978-1-4503-2477-9/13/11 $15.00
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

CRYPTO MISUSE IN ANDROID APPS

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

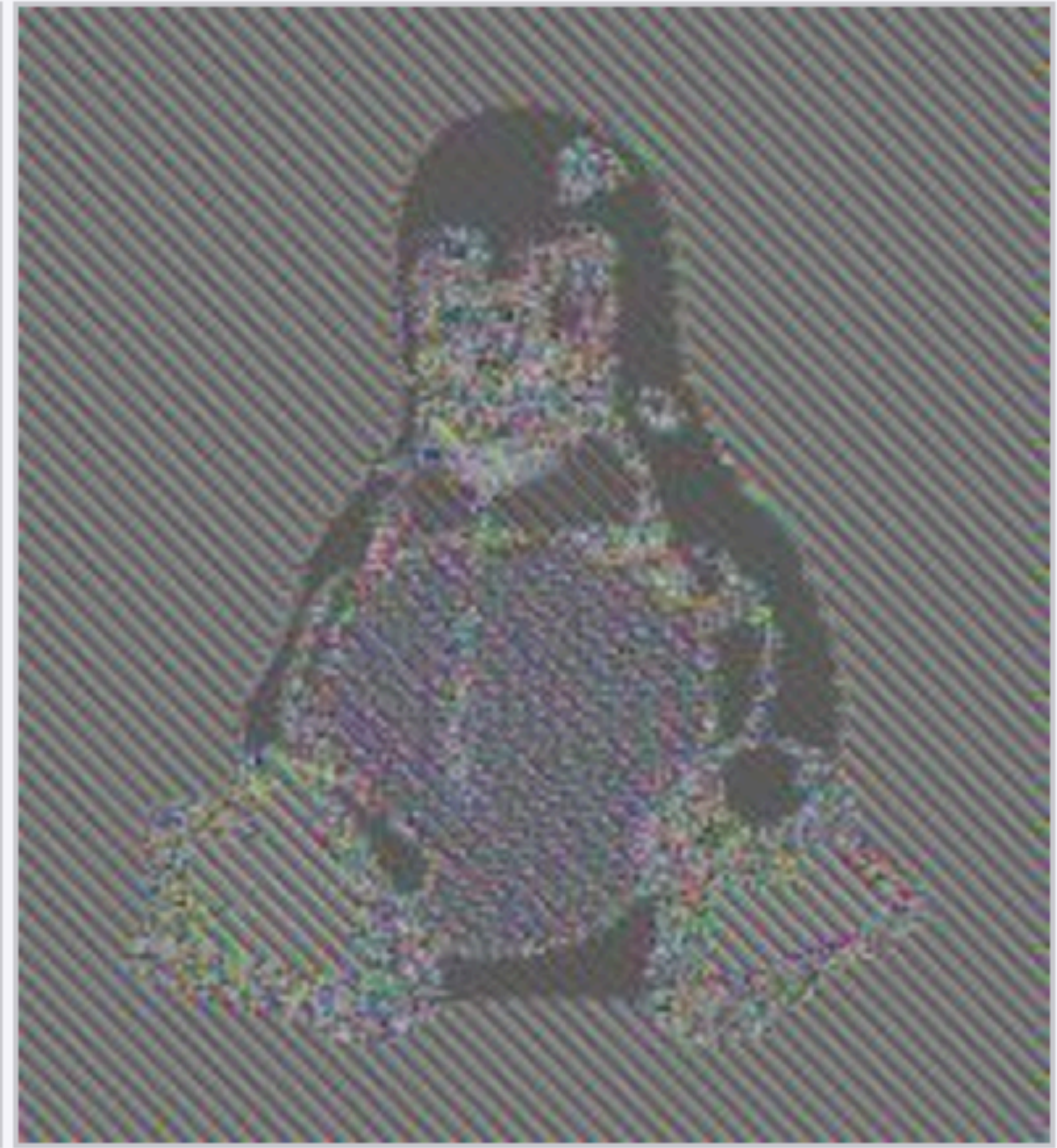
	# apps	violated rule
48%	5,656	Uses ECB (BouncyCastle default) (R1)
31%	3,644	Uses constant symmetric key (R3)
17%	2,000	Uses ECB (Explicit use) (R1)
16%	1,932	Uses constant IV (R2)
	1,636	Used iteration count < 1,000 for PBE(R5)
14%	1,629	Seeds SecureRandom with static (R6)
	1,574	Uses static salt for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

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.

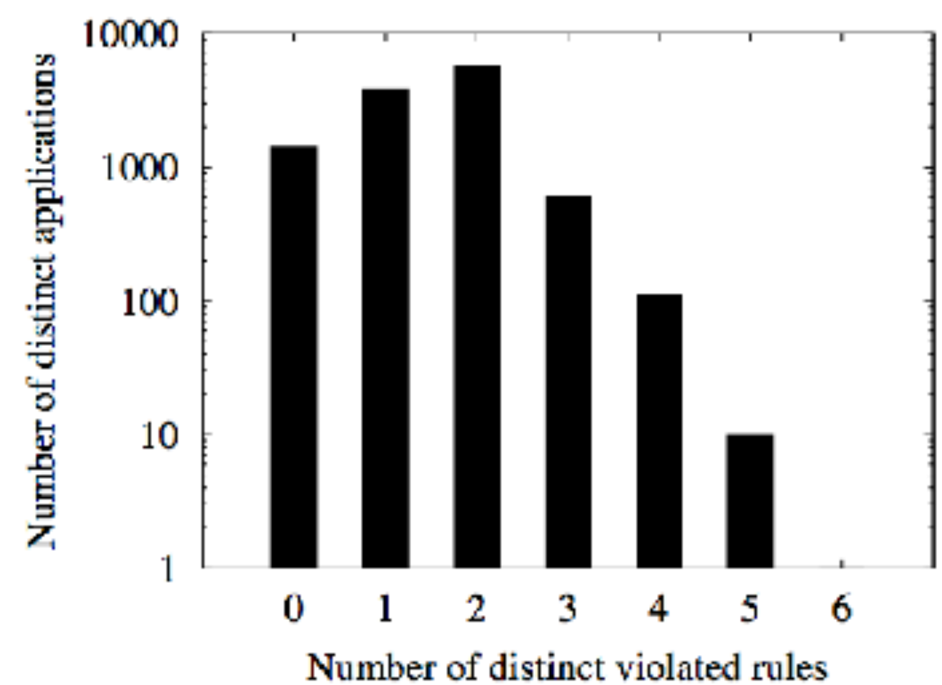
#Occurences	Symmetric encryption scheme
5878	AES/CBC/PKCS5Padding
4803	AES *
1151	DES/ECB/NoPadding
741	DES *
501	DESede *
473	DESede/ECB/PKCS5Padding
468	AES/CBC/NoPadding
443	AES/ECB/PKCS5Padding
235	AES/CBC/PKCS7Padding
221	DES/ECB/PKCS5Padding
220	AES/ECB/NoPadding
205	DES/CBC/PKCS5Padding
155	AES/ECB/PKCS7Padding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

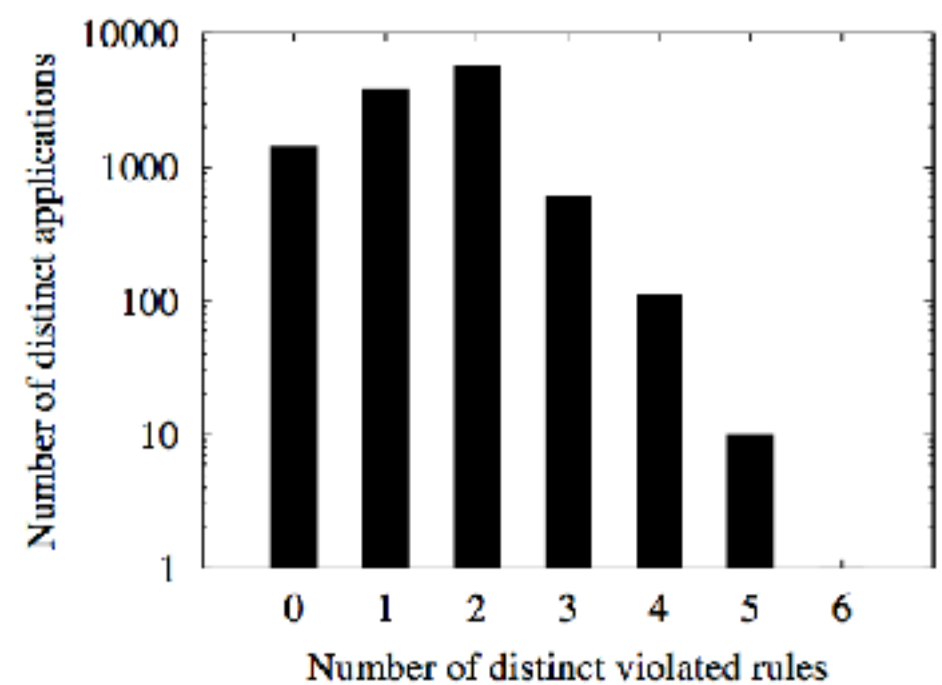
	# apps	violated rule
48%	5,656	Uses ECB (BouncyCastle default) (R1)
31%	3,644	Uses constant symmetric key (R3)
17%	2,000	Uses ECB (Explicit use) (R1)
16%	1,932	Uses constant IV (R2)
	1,636	Used iteration count < 1,000 for PBE(R5)
14%	1,629	Seeds SecureRandom with static (R6)
	1,574	Uses static salt for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



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:

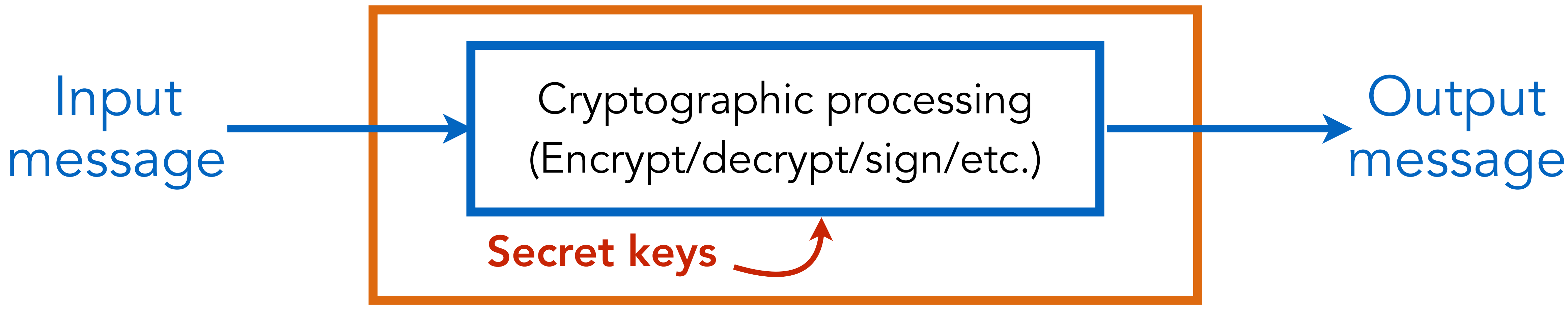
- Do not **roll your own** cryptographic mechanisms
  - Takes peer review
  - Apply Kerkhoff's principle
- Do not ***misuse*** existing crypto
- Do not even ***implement*** the underlying crypto

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?

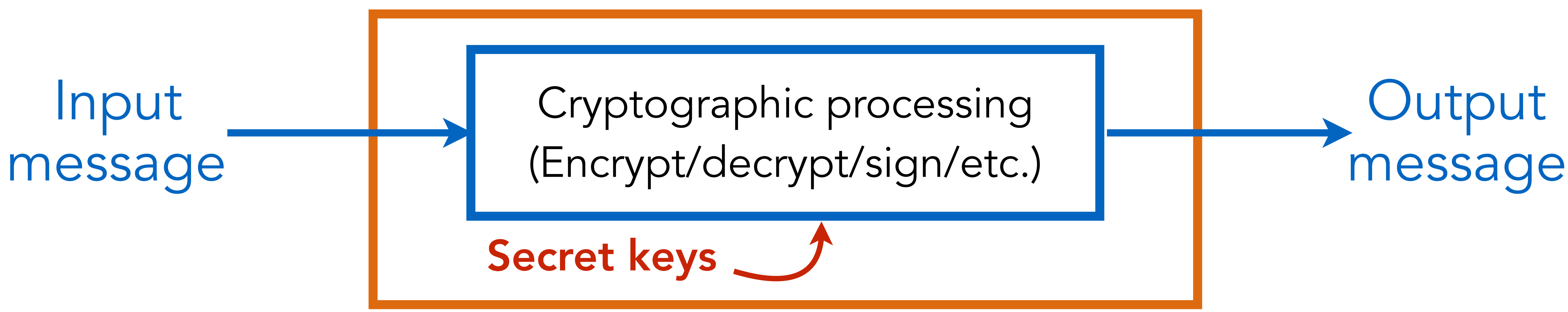
SIDE-CHANNEL ATTACKS

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



SIDE-CHANNEL ATTACKS

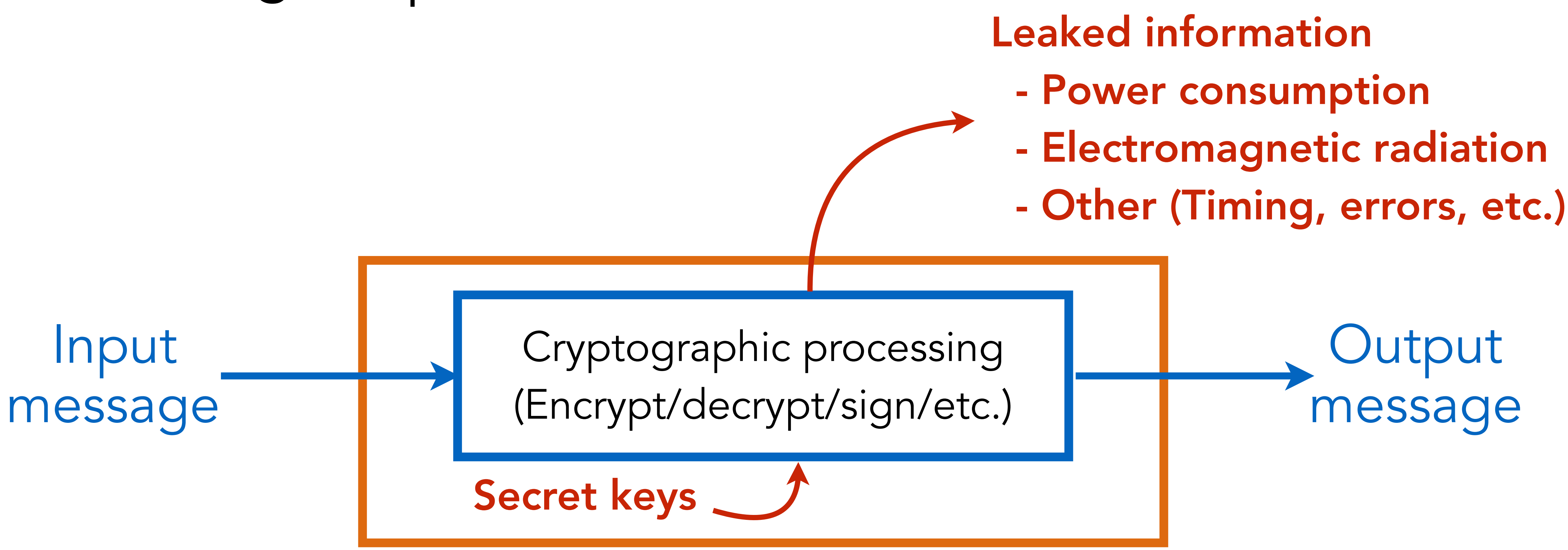
- Cryptography concerns the *theoretical* difficulty in breaking a cipher



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

SIDE-CHANNEL ATTACKS

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



- 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

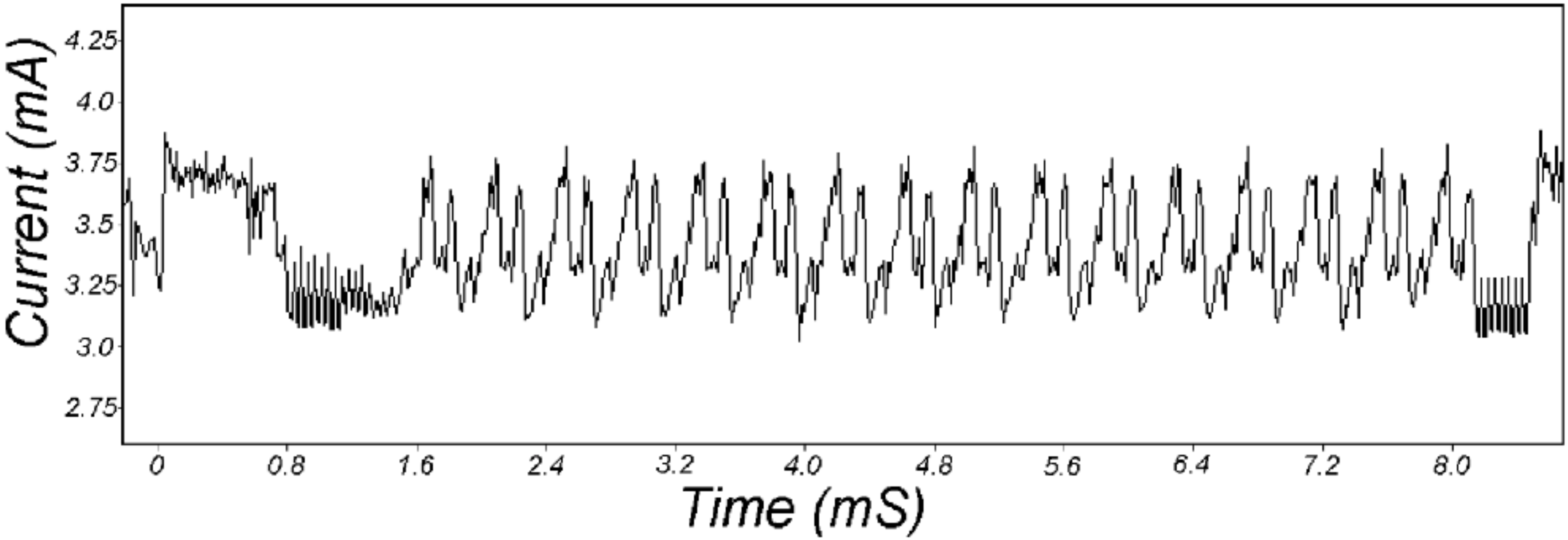


Figure 1: SPA trace showing an entire DES operation.

Overall operation clearly visible:
Can identify the **16 rounds of DES**

SPA ON DES

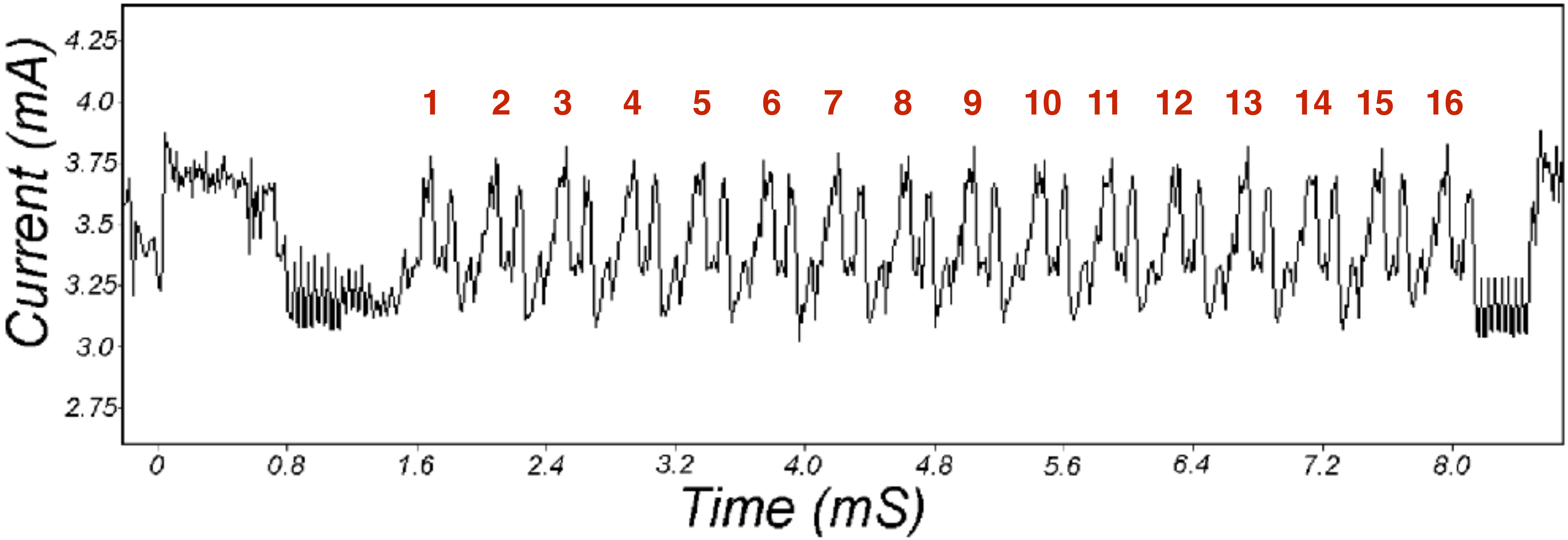


Figure 1: SPA trace showing an entire DES operation.

Overall operation clearly visible:

Can identify the **16 rounds of DES**

SPA ON DES

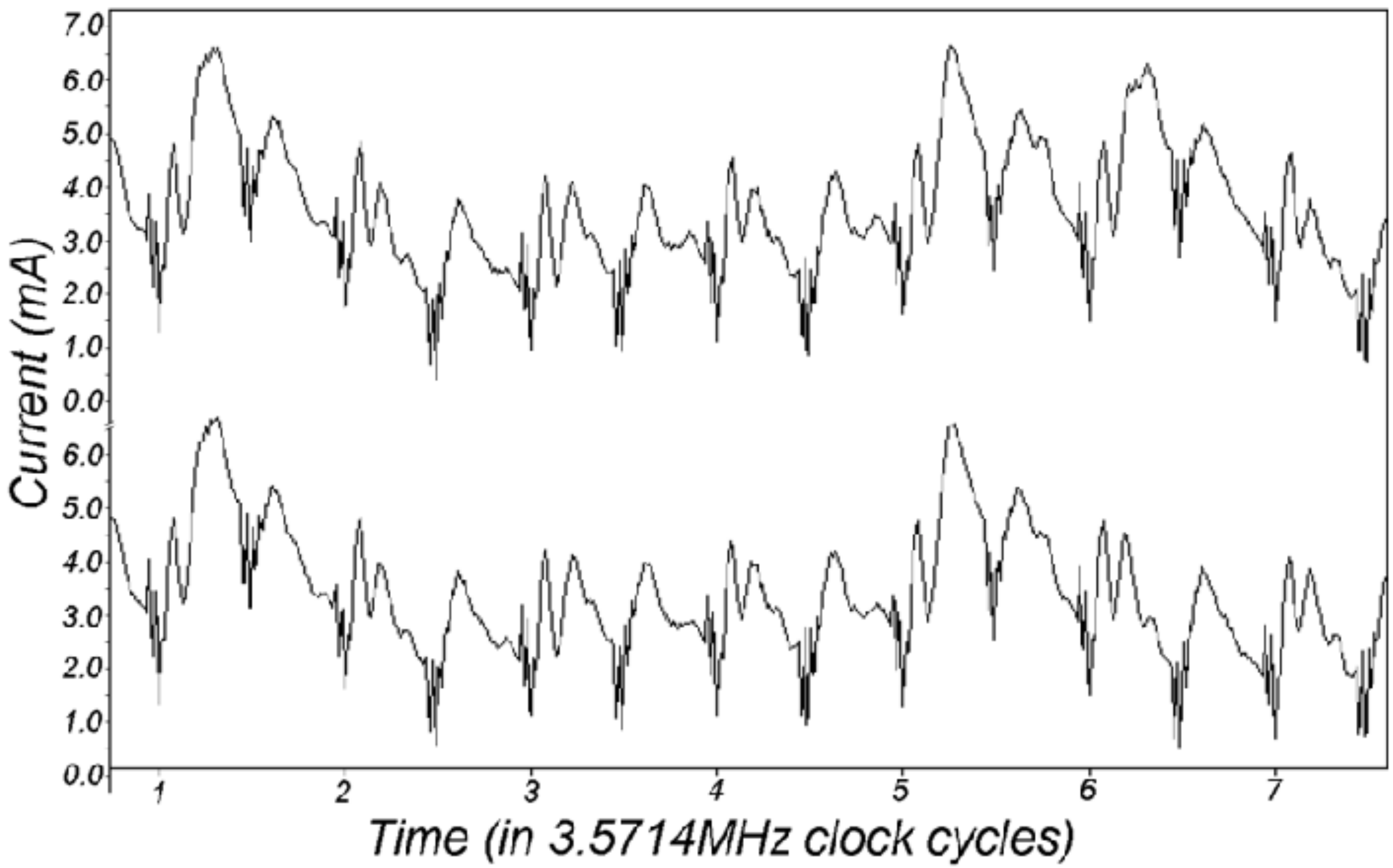


Figure 3: SPA trace showing individual clock cycles.

Specific **instructions** are also discernible

SPA ON DES

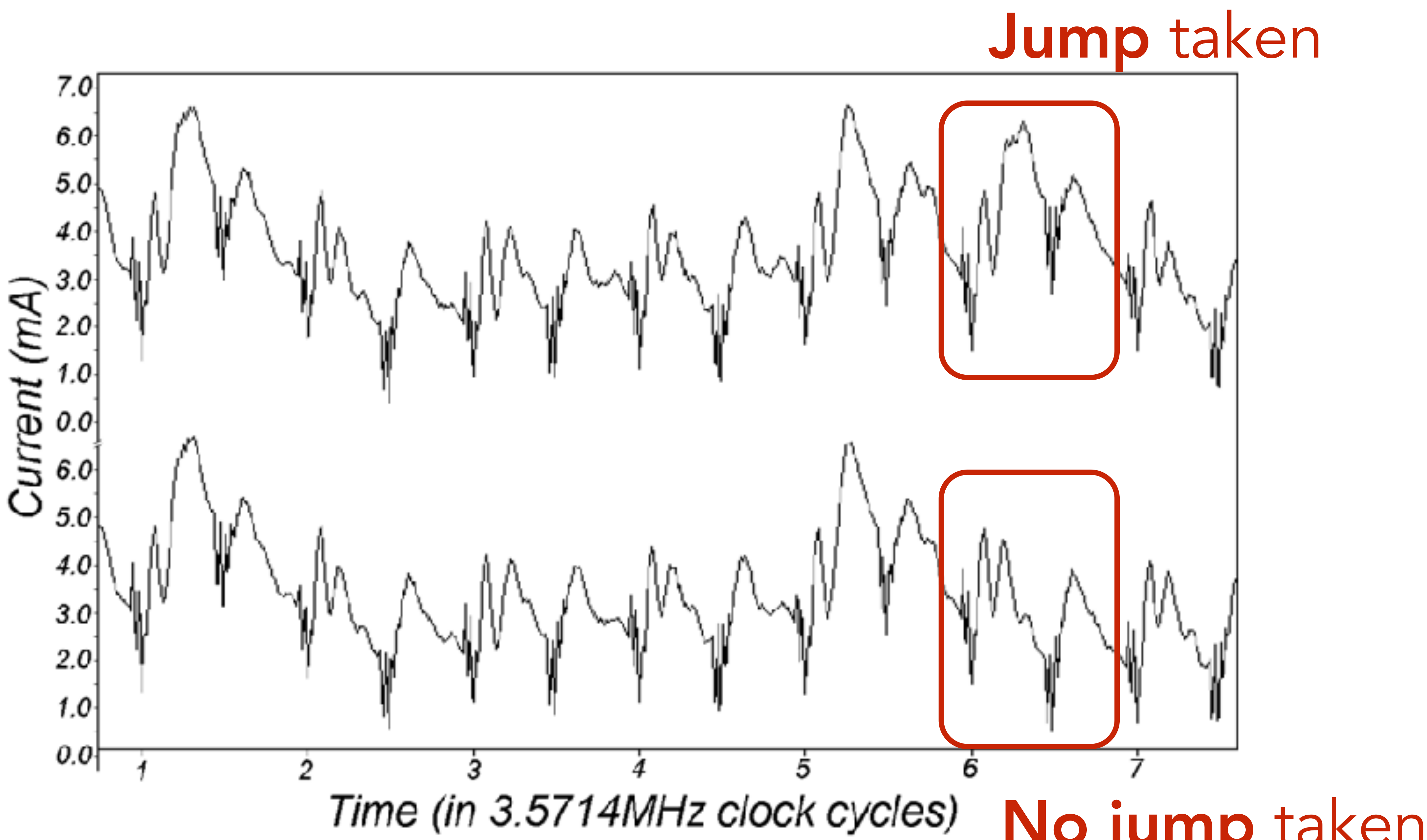


Figure 3: SPA trace showing individual clock cycles.

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
   }
}
```

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?

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?

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

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

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

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)

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

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?

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

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)

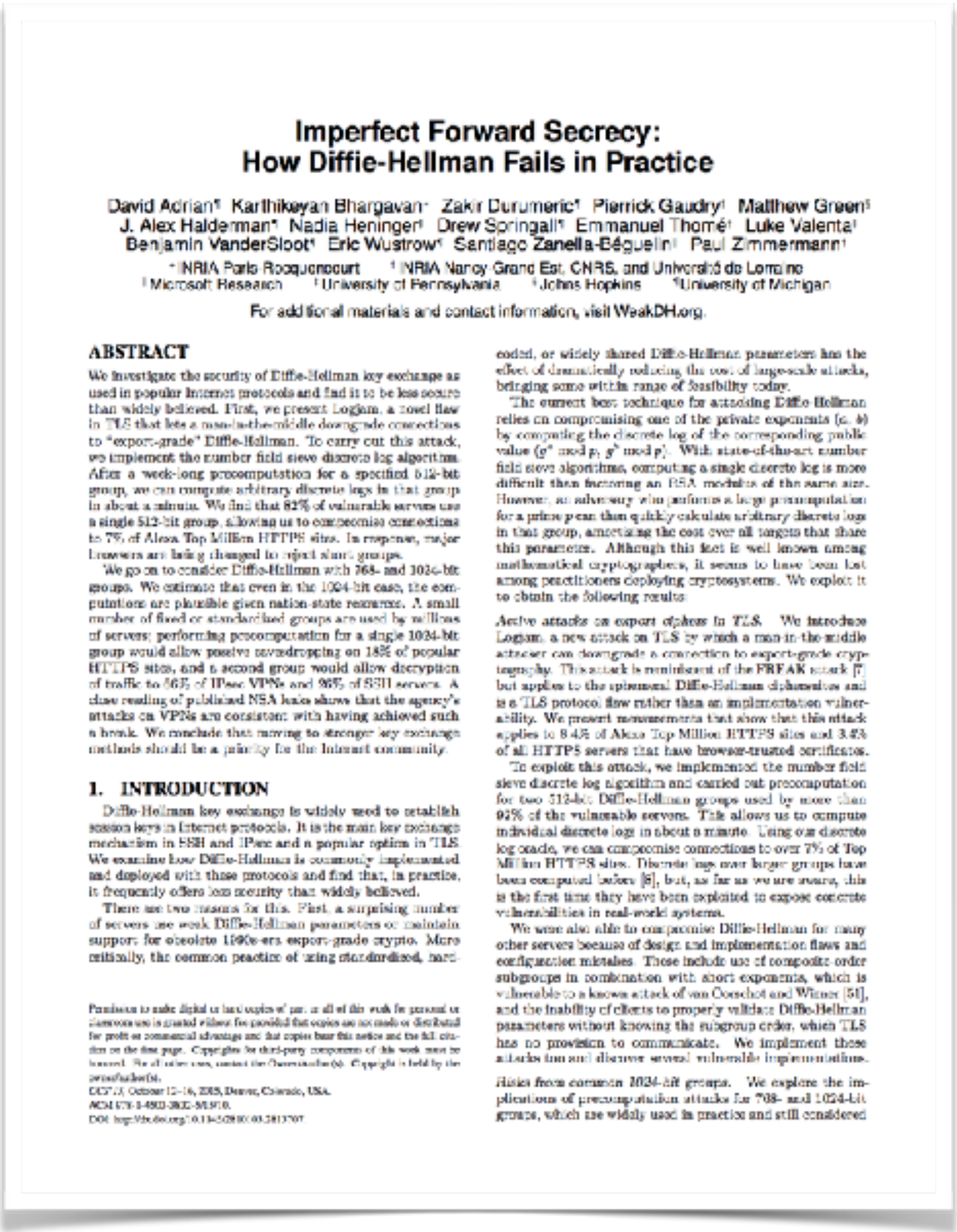
- SPA just visually inspects a single run
- DPA runs iteratively and reactively
  - Get multiple samples
  - Based on these, construct new plaintext messages as inputs, and repeat

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 FROM GOVERNMENTS



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.

Exploits export-grade encryption

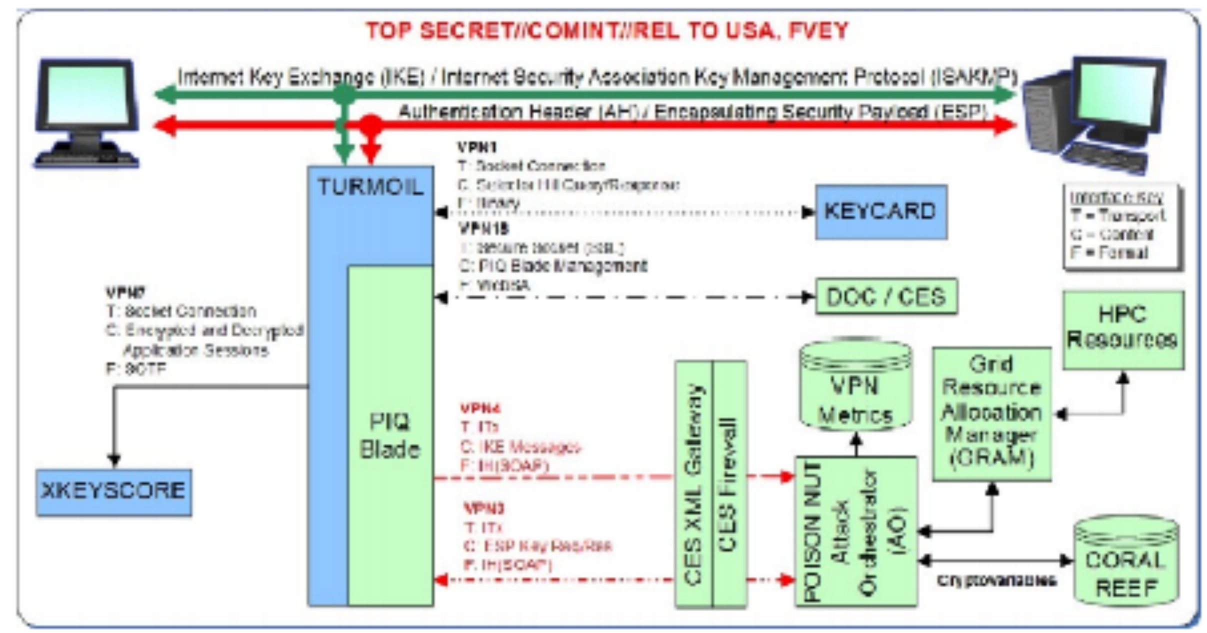


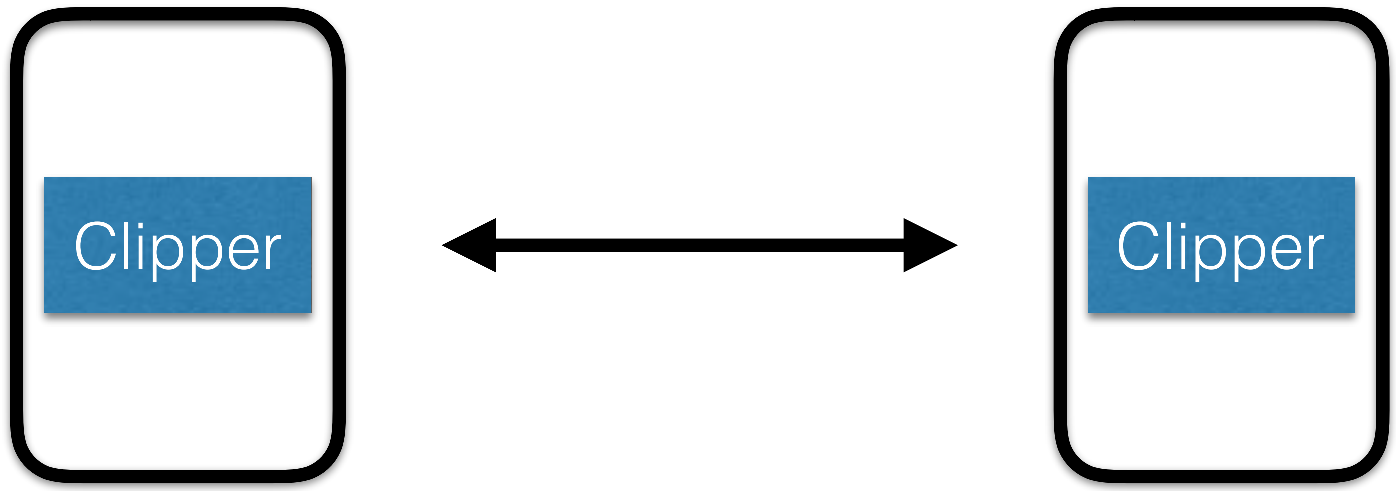
Figure 4: **NSA's VPN decryption infrastructure.** This classified illustration published by Der Spiegel [67] shows captured IKE handshake messages being passed to a high-performance computing system, which returns the symmetric keys for ESP session traffic. The details of this attack are consistent with an efficient break for 1024-bit Diffie-Hellman.

1024-bit and smaller feasibly broken

Logjam downgrades to export-grade (512)

Clipper chip

A lesson in poorly designed protocols



Goal: Confidentiality — Support encrypted communication between devices

Goal: Key escrow — Permit law enforcement to obtain “session keys” with a warrant

Clipper chip: Design

Tamper-proof hardware

Hardware that is difficult to introspect (e.g., extract keys), alter (change the algorithms), or impersonate

Skipjack
encryption algorithm

Skipjack Keys
Unit key
Global family key

Diffie-Hellman
key exchange

LEAF generation
& validation

Clipper chip: Design

Tamper-proof hardware

Skipjack
encryption algorithm

Skipjack Keys
Unit key
Global family key

Diffie-Hellman
key exchange

LEAF generation
& validation

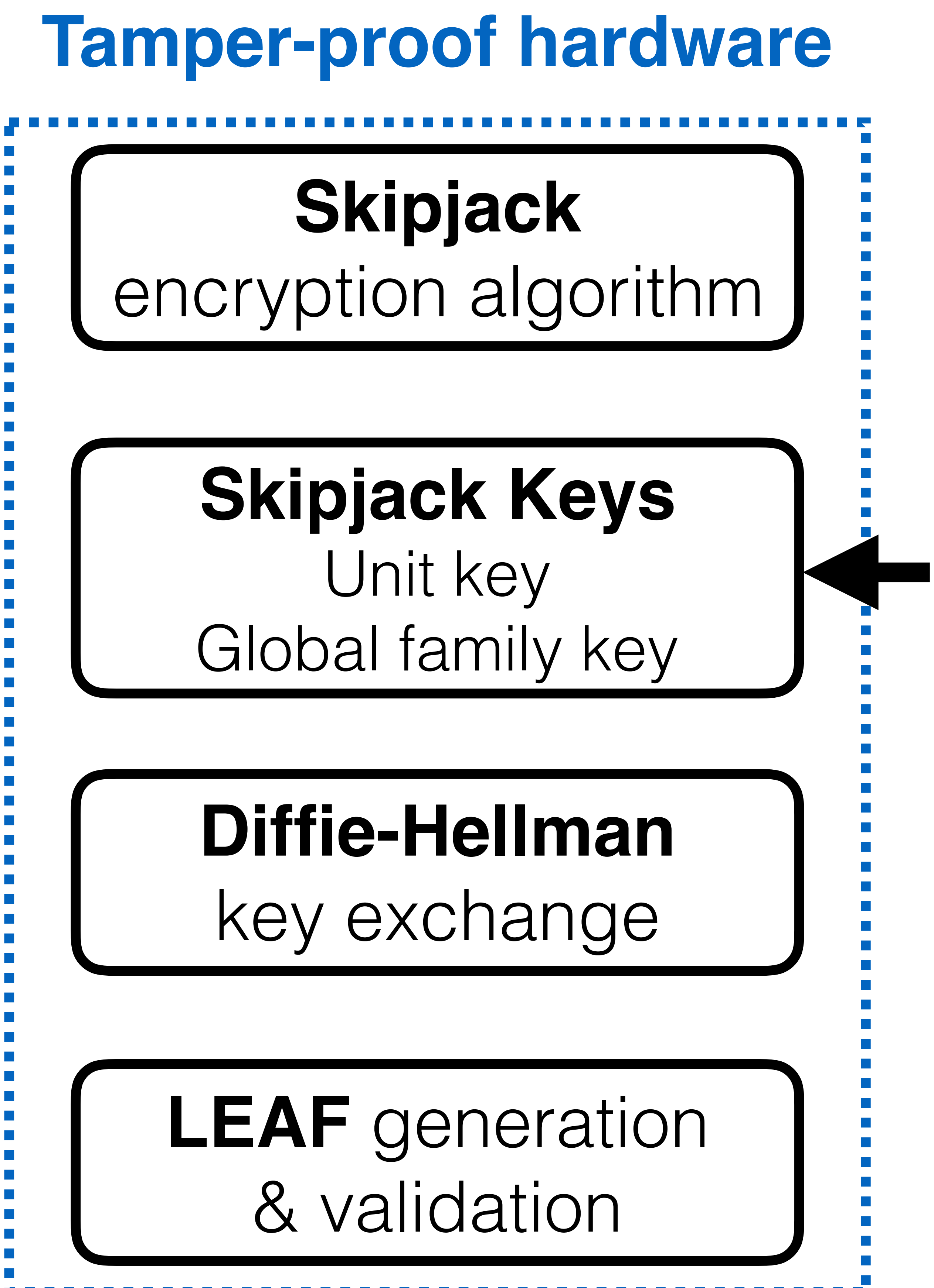
Block cipher designed by the NSA, originally classified SECRET.

(Violates Kirchhoff's principle)

Broken within *one day* of declassification.

80-bit key; similar algorithm to DES (also broken)

Clipper chip: Design



Assigned when the hardware is manufactured.

Unit key is unique to this unit in particular (each Clipper chip also has a *unit ID*).

Global family key is the same across many units.

Clipper chip: Design

Tamper-proof hardware

Skipjack
encryption algorithm

Skipjack Keys
Unit key
Global family key

Diffie-Hellman
key exchange

LEAF generation
& validation

Used for establishing a (symmetric) ***session key***

Session keys are ephemeral (e.g., last only for a given connection, transaction, etc.)

General properties about session keys:

- Compromising one session key does not compromise others
- Compromising a long-term key should not compromise past session keys (**forward secrecy**)

Clipper chip: Design

Tamper-proof hardware

Skipjack
encryption algorithm

Skipjack Keys
Unit key
Global family key

Diffie-Hellman
key exchange

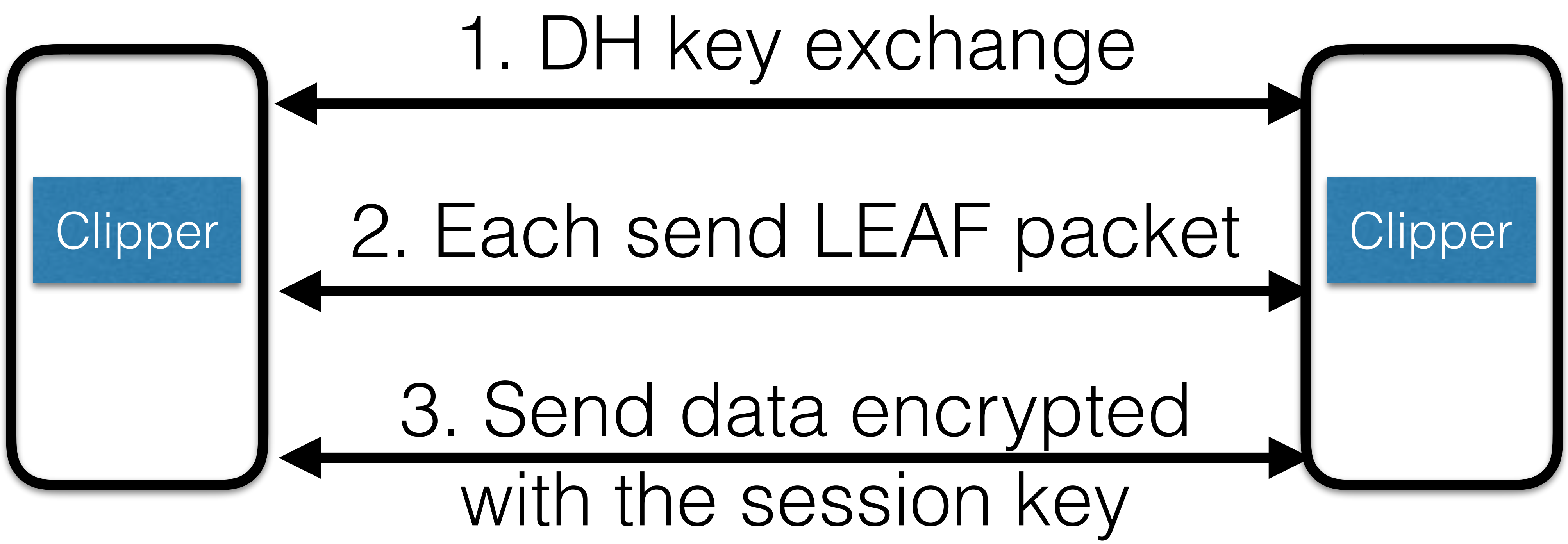
LEAF generation
& validation

LEAF
(Law Enforcement Access Field)

To permit wiretapping, law enforcement needs to be able to extract session keys, but only has access to what is sent during communication

Idea: send data that has enough info to allow law enforcement to extract keys (but not any other eavesdropper).

LEAF protocol design

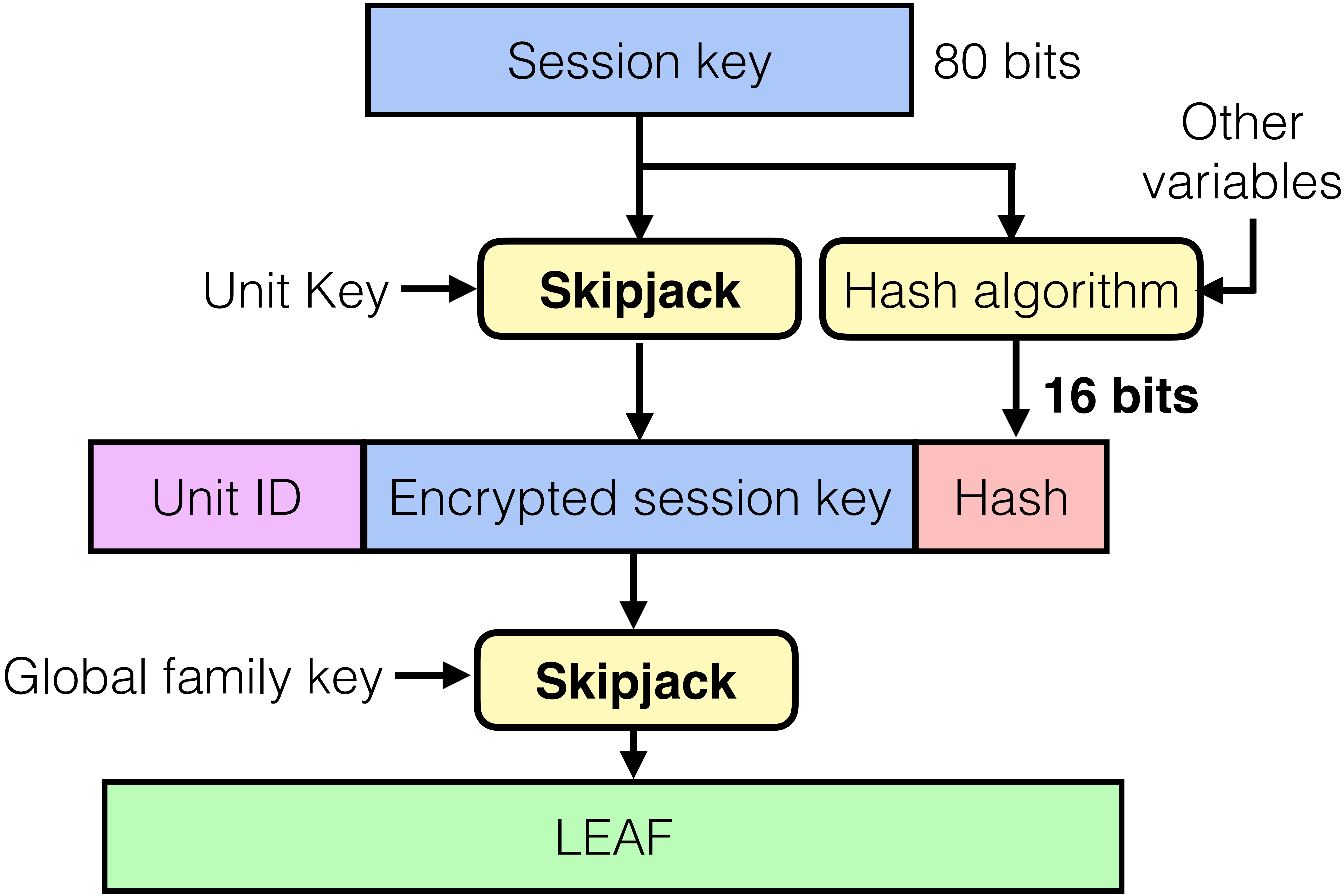


The Clipper chips will not decrypt until it has received a valid LEAF packet

Law enforcement sees all packets.

- Cannot infer key from DH key exchange
- *Can* infer it from the LEAF packet

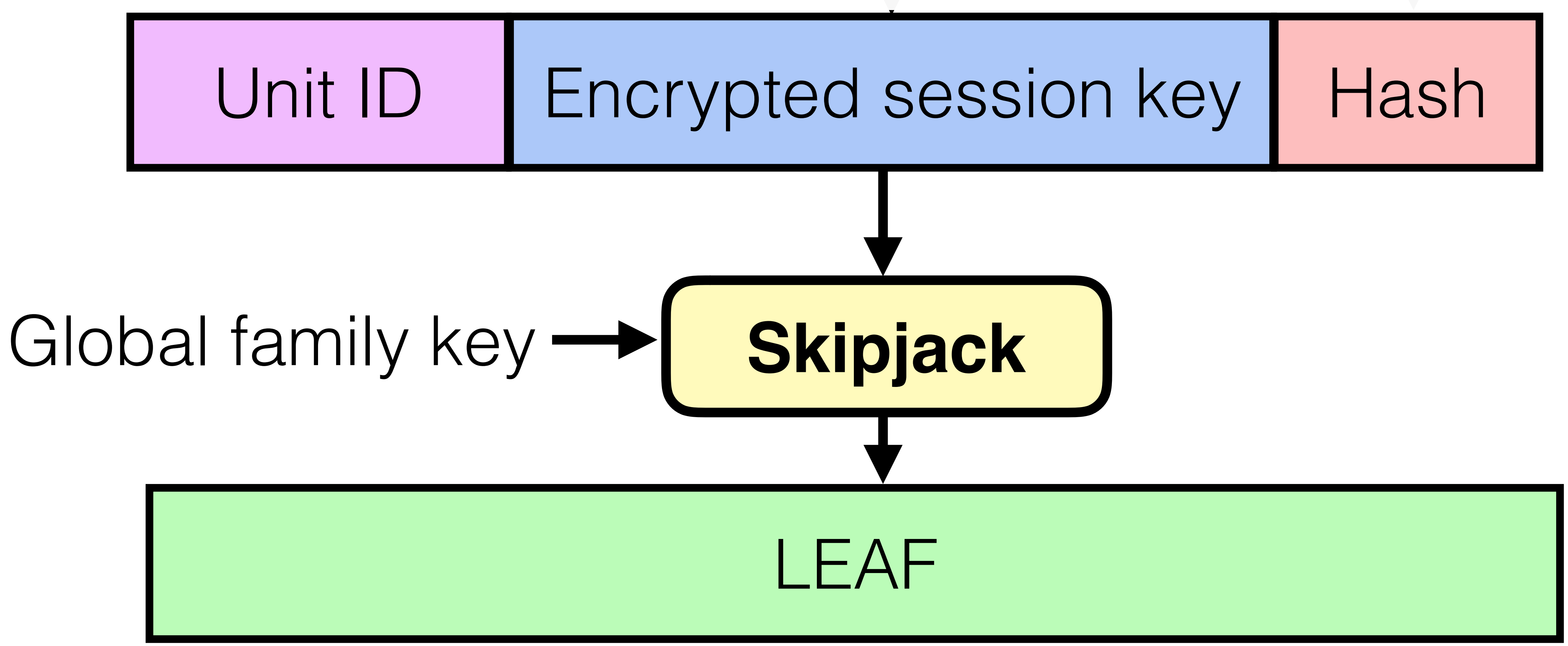
LEAF message structure



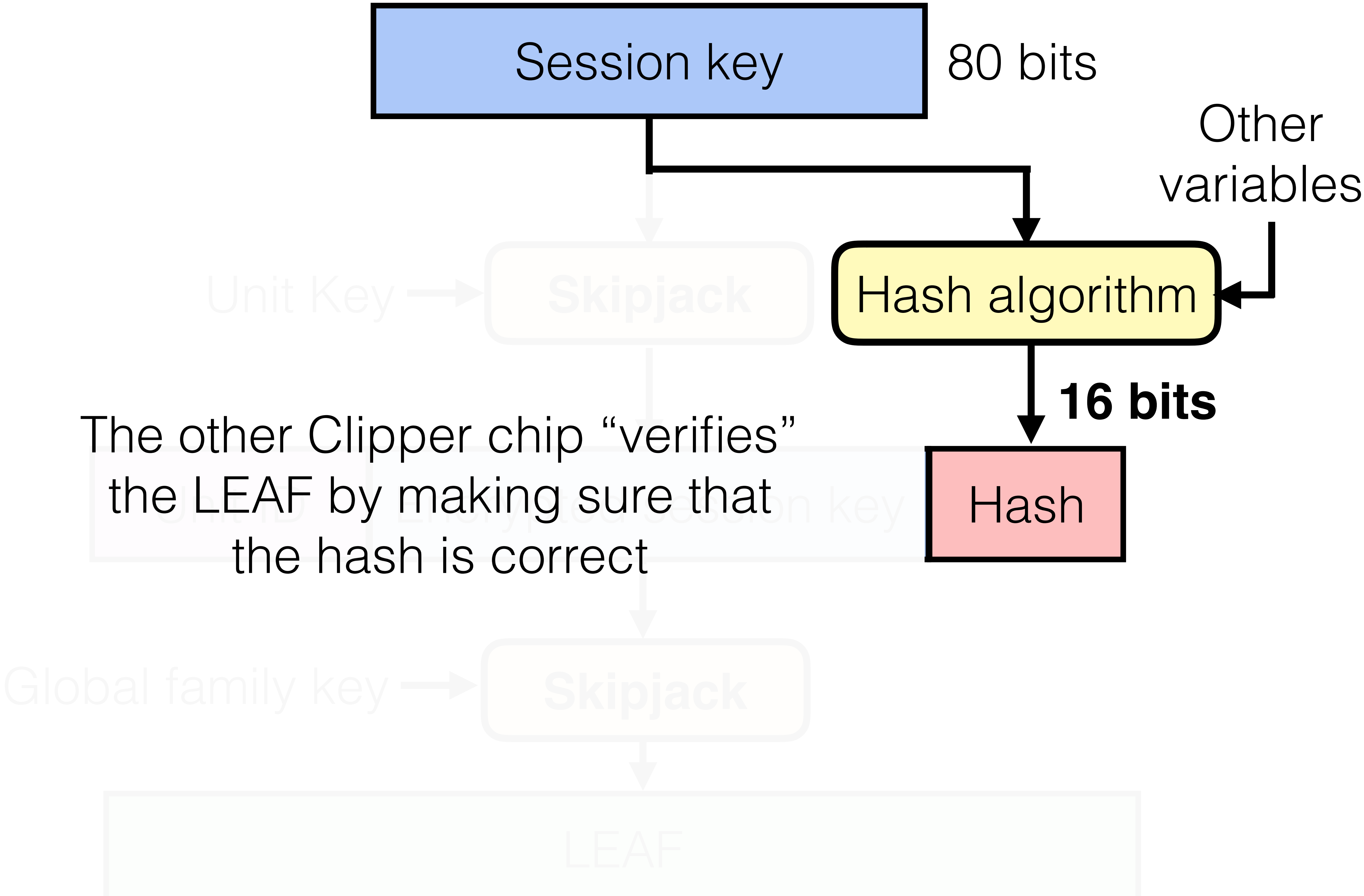
LEAF message structure

The other Clipper chip also has the Global Family key

=> Can decrypt the LEAF to obtain this triple



LEAF message structure

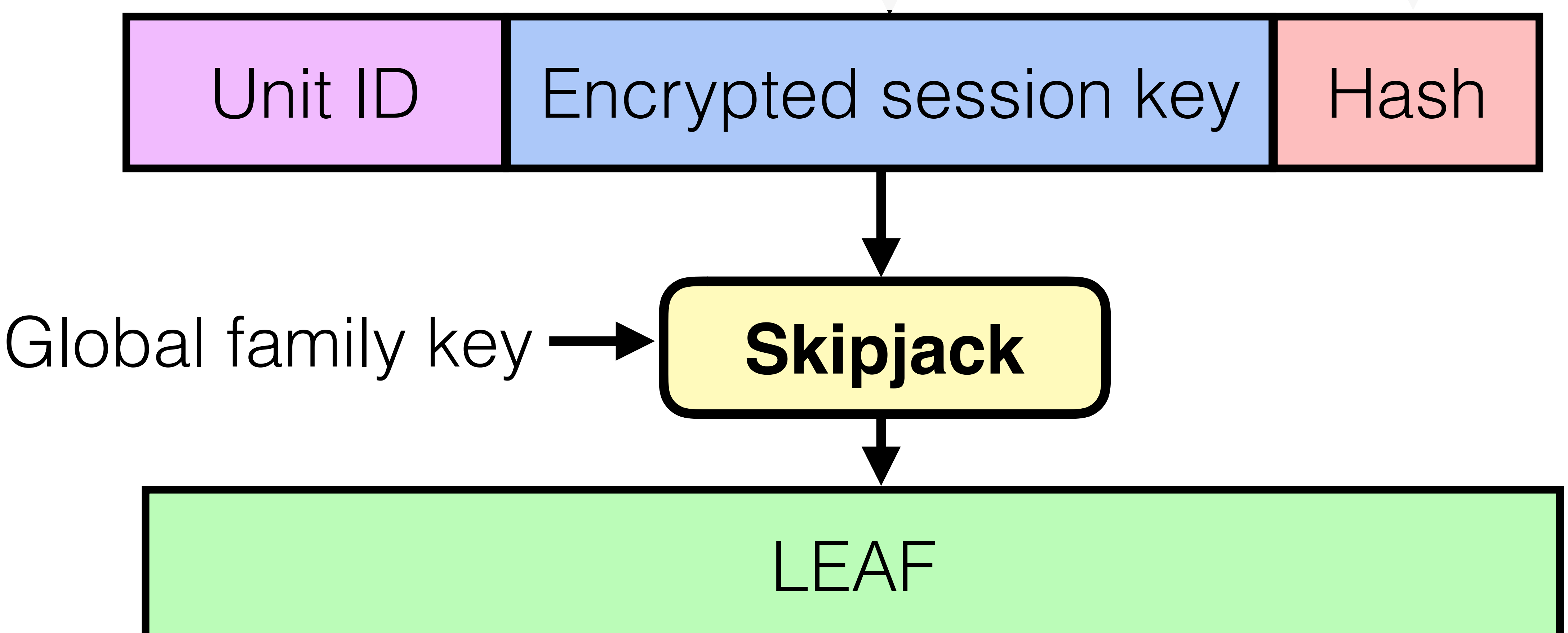


The other Clipper chip “verifies” the LEAF by making sure that the hash is correct

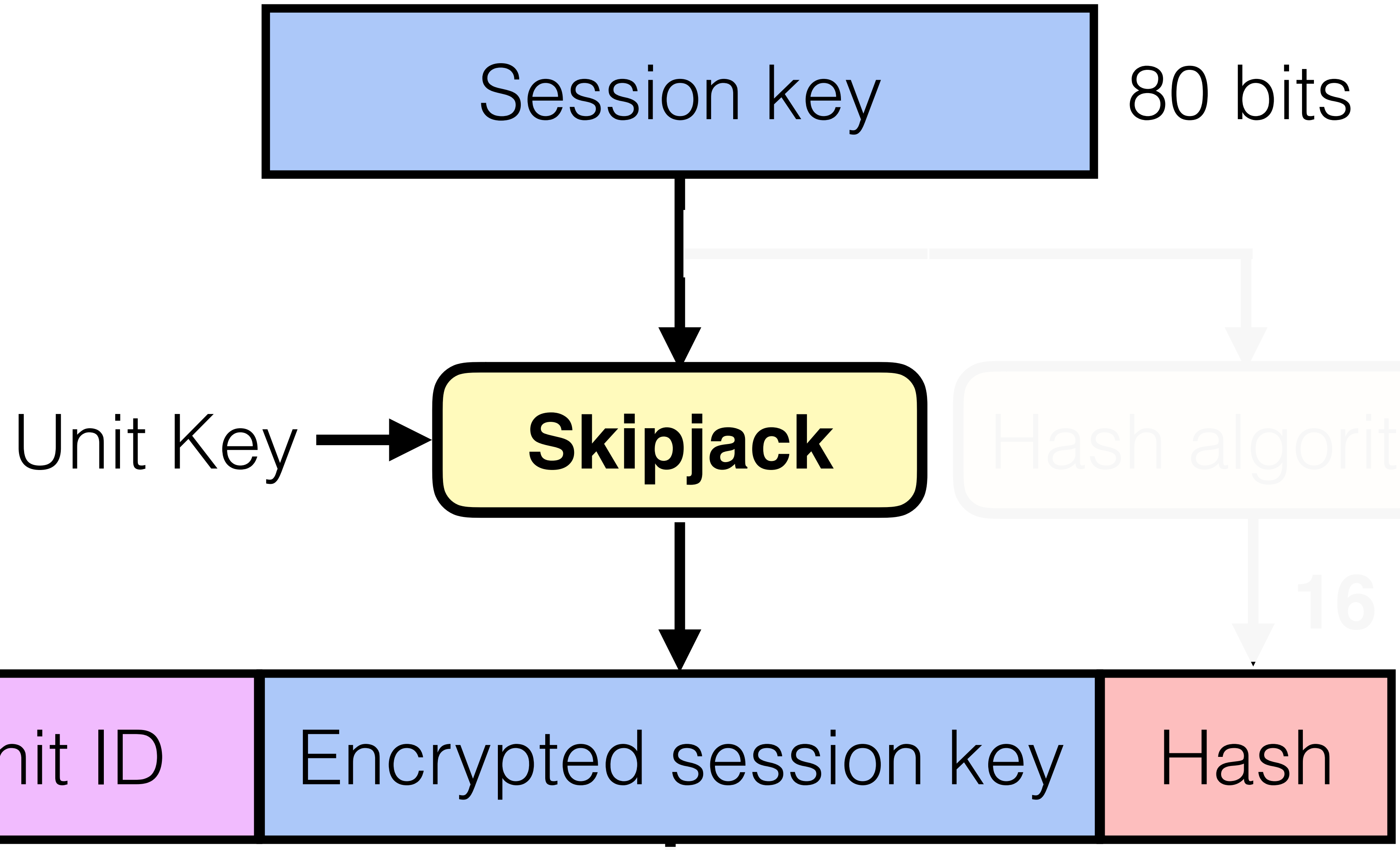
LEAF message structure

Law enforcement also has the Global Family Key

=> Can decrypt the LEAF to obtain this triple



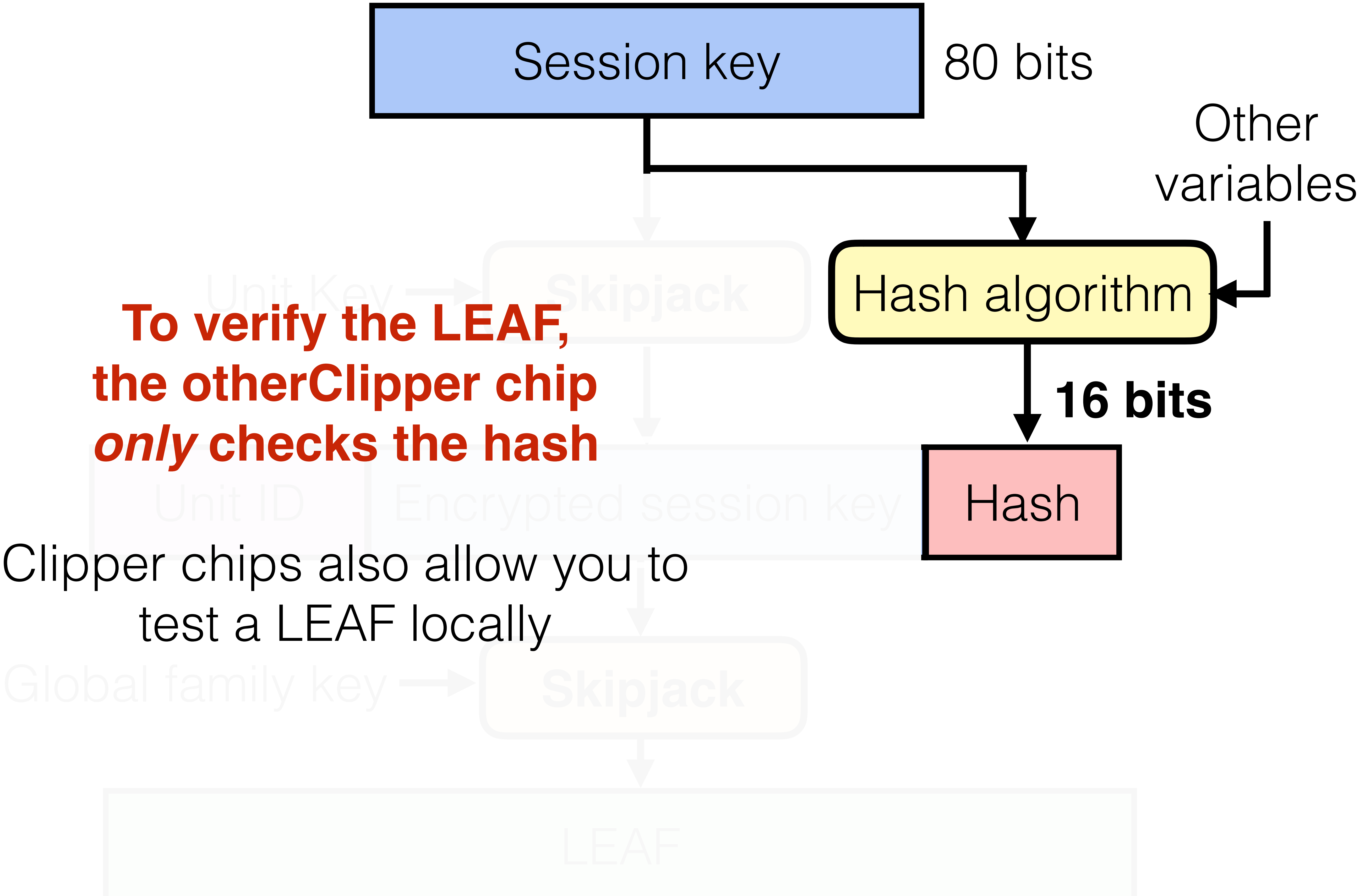
LEAF message structure



Law enforcement *does not* have direct access to all unit keys; needs a **warrant** to get them

Unit keys are split across two locations (one location gets a OTP, the other gets the XOR)

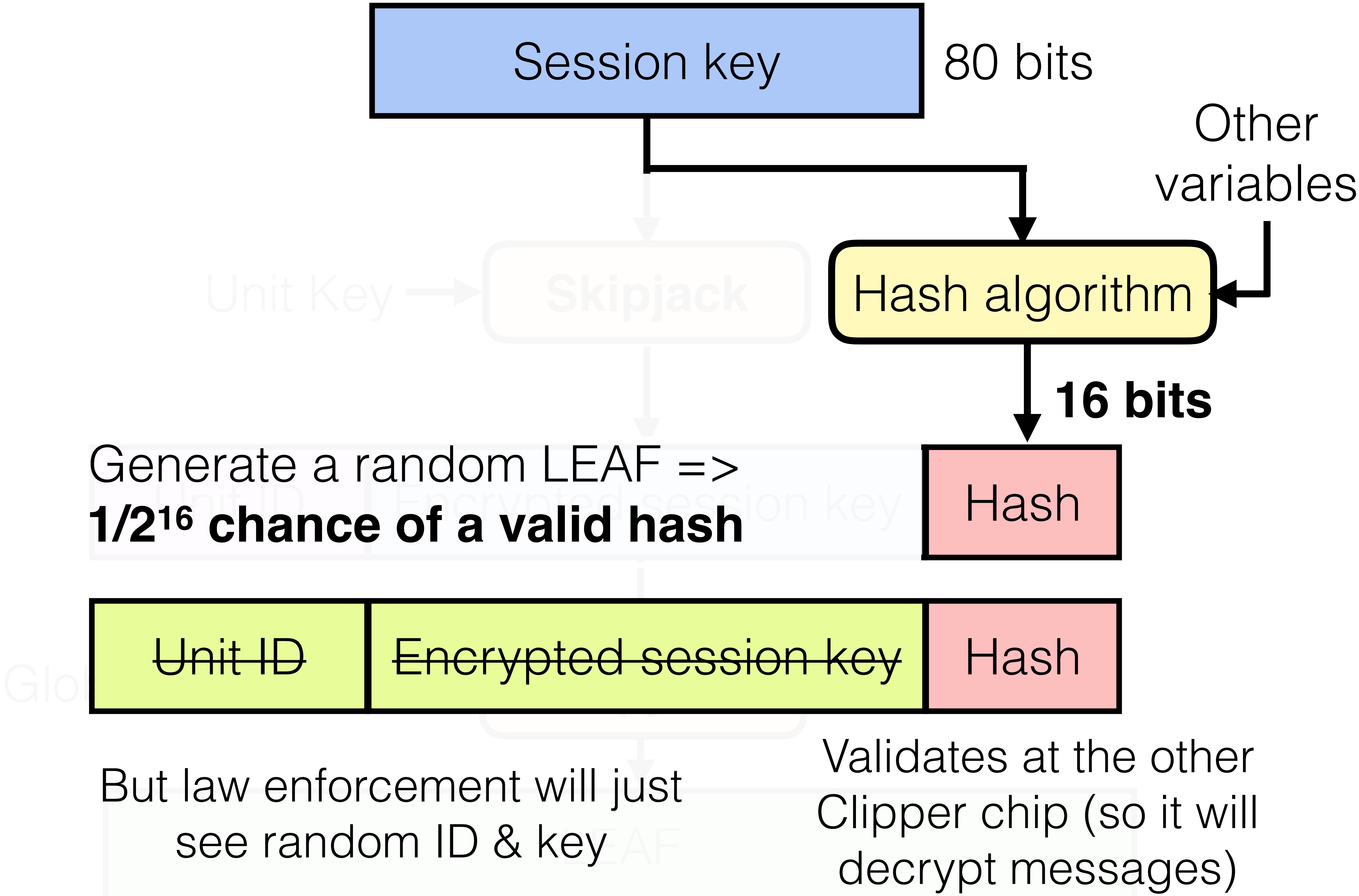
LEAF: failure



To verify the LEAF, the otherClipper chip *only* checks the hash

Clipper chips also allow you to test a LEAF locally

LEAF: failure



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 distinct 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

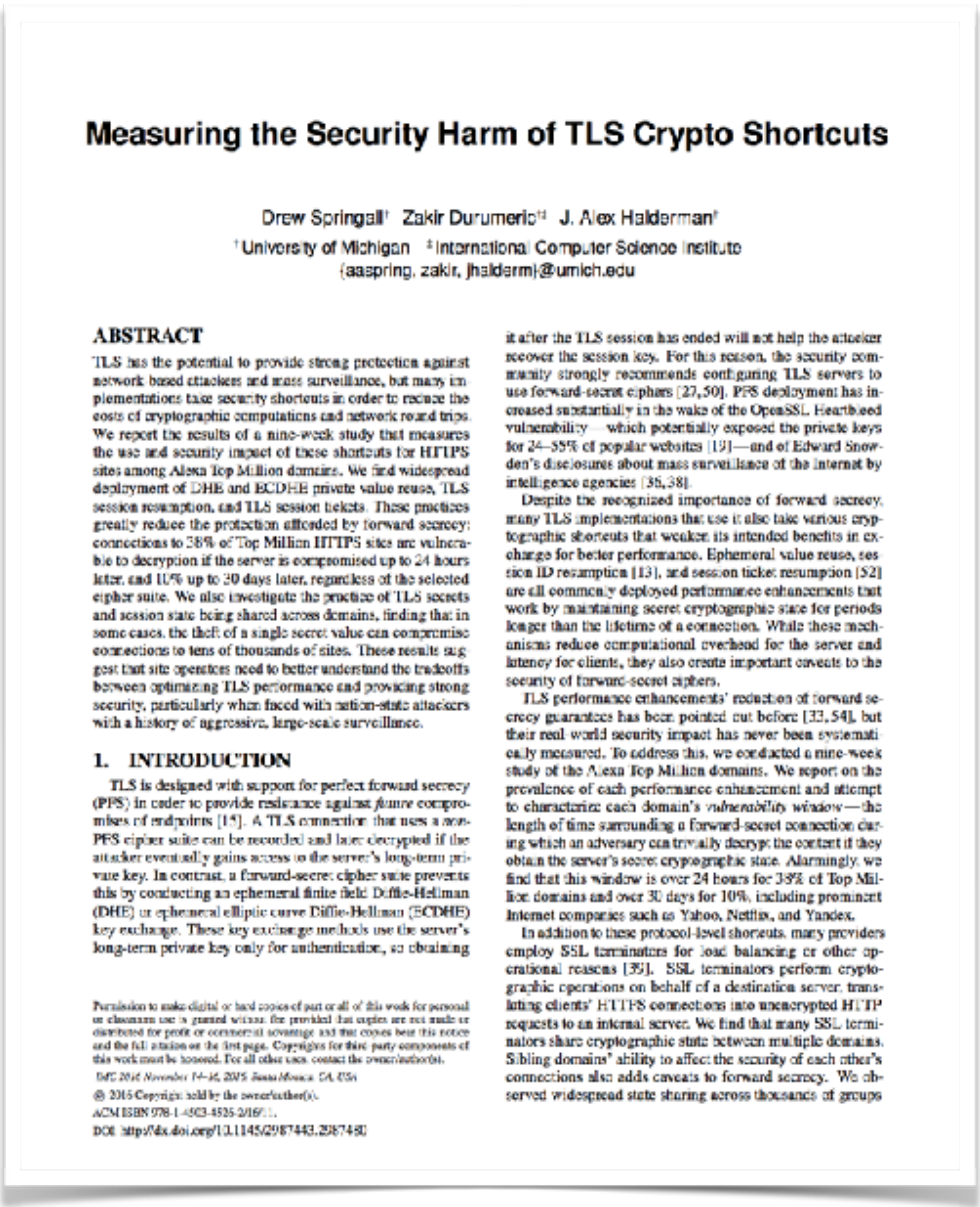
⟹ Highly parallelizable

Randomize addresses

⟹ Avoid takedown notices

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

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 [35]. 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

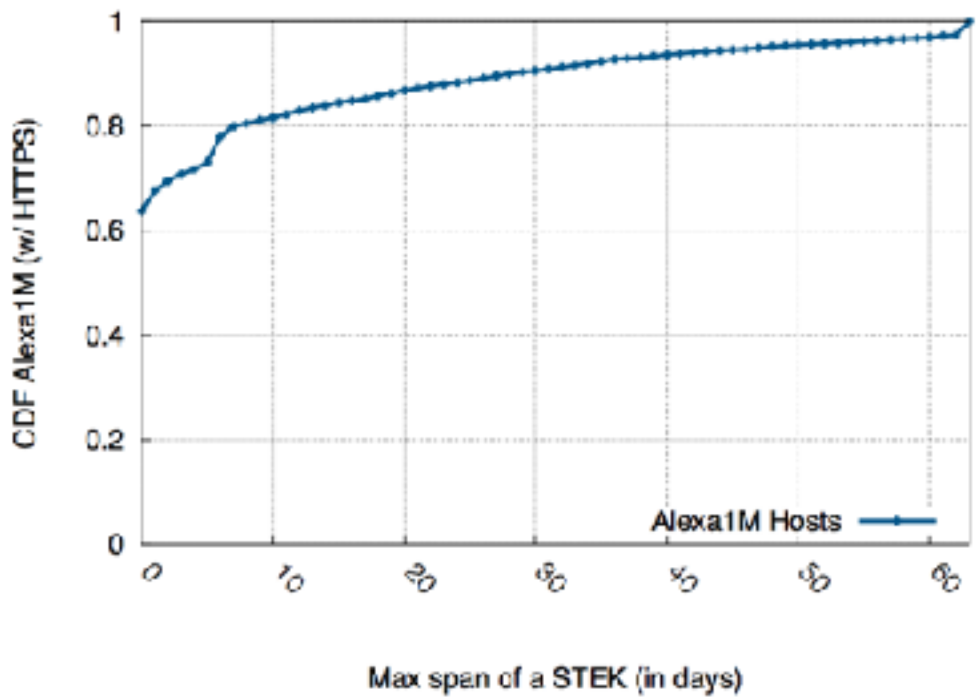


Figure 3: **STEK Lifetime**—TLS connections cannot achieve forward secrecy until the STEK (the key used by the server to encrypt the session ticket) is discarded.

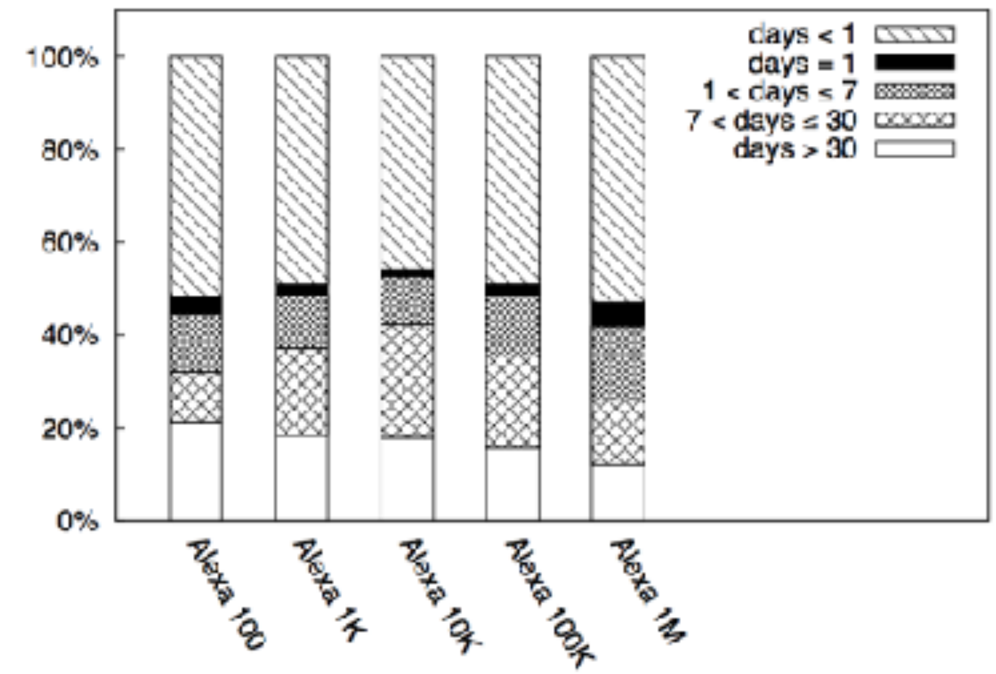


Figure 4: **STEK Lifetime by Alexa Rank**—We found 12 Alexa Top 100 sites that persisted STEKs for at least 30 days.

Incentive to hold onto STEKs (lower RTTs)

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

UNSAFE OPTIMIZATIONS

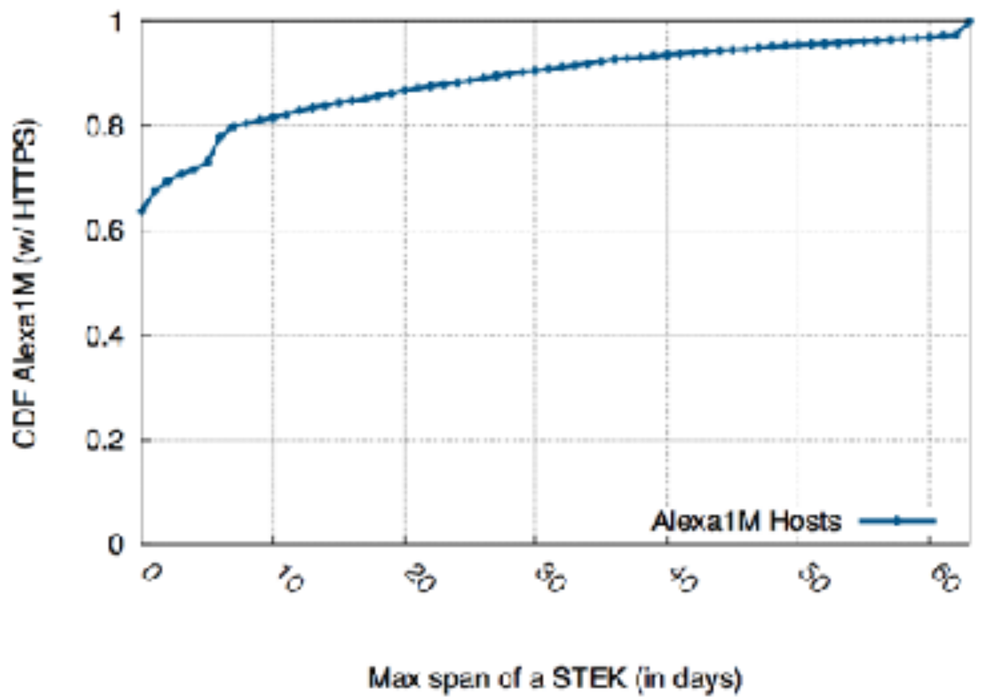


Figure 3: **STEK Lifetime**—TLS connections cannot achieve forward secrecy until the STEK (the key used by the server to encrypt the session ticket) is discarded.

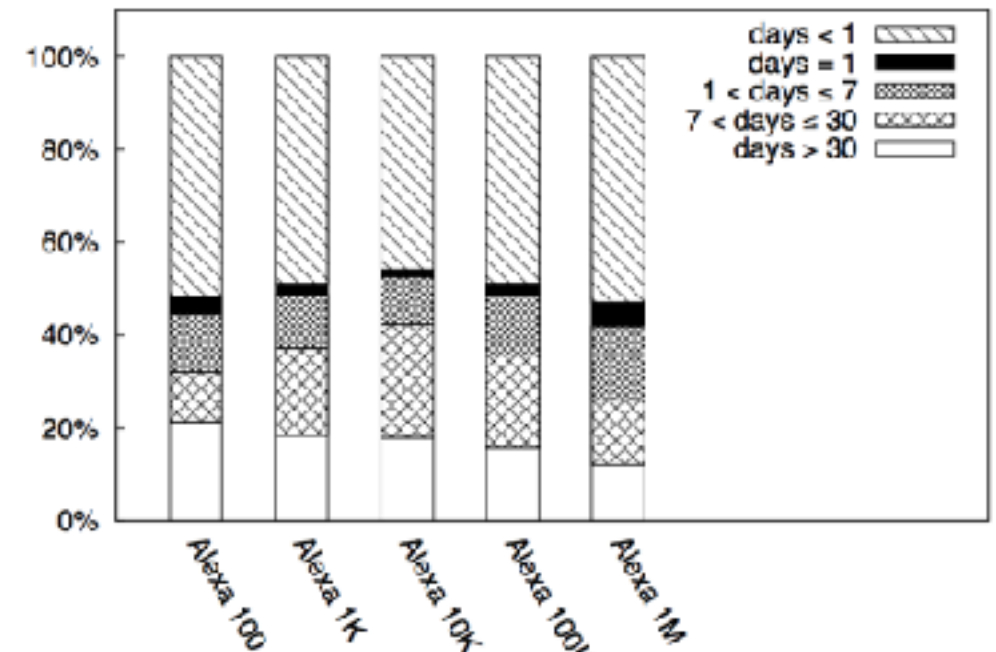


Figure 4: **STEK Lifetime by Alexa Rank**—We found 12 Alexa Top 100 sites that persisted STEKs for at least 30 days.

Incentive to hold onto STEKs (lower RTTs)

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

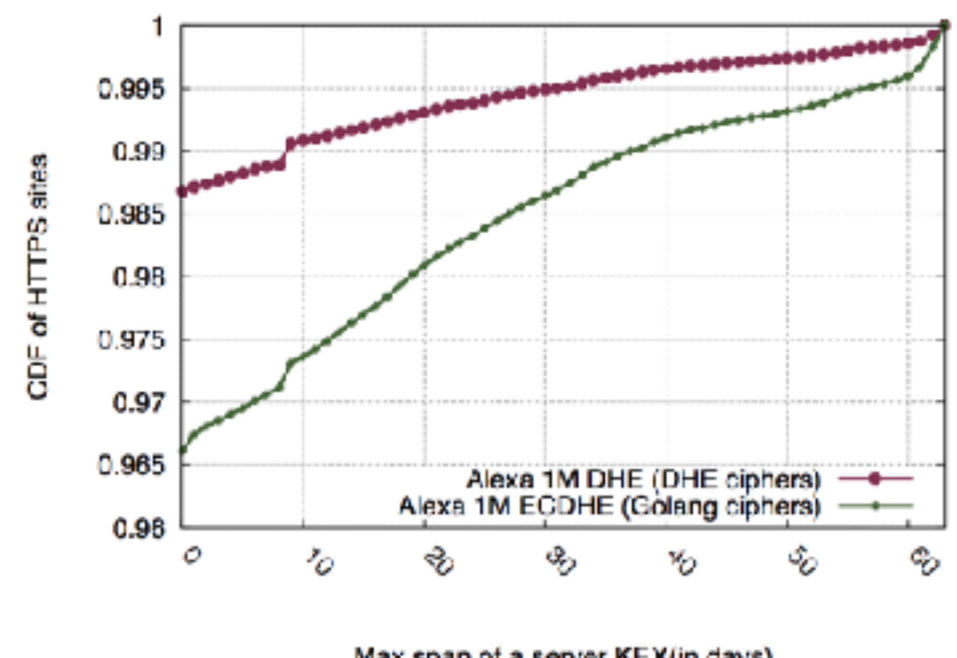
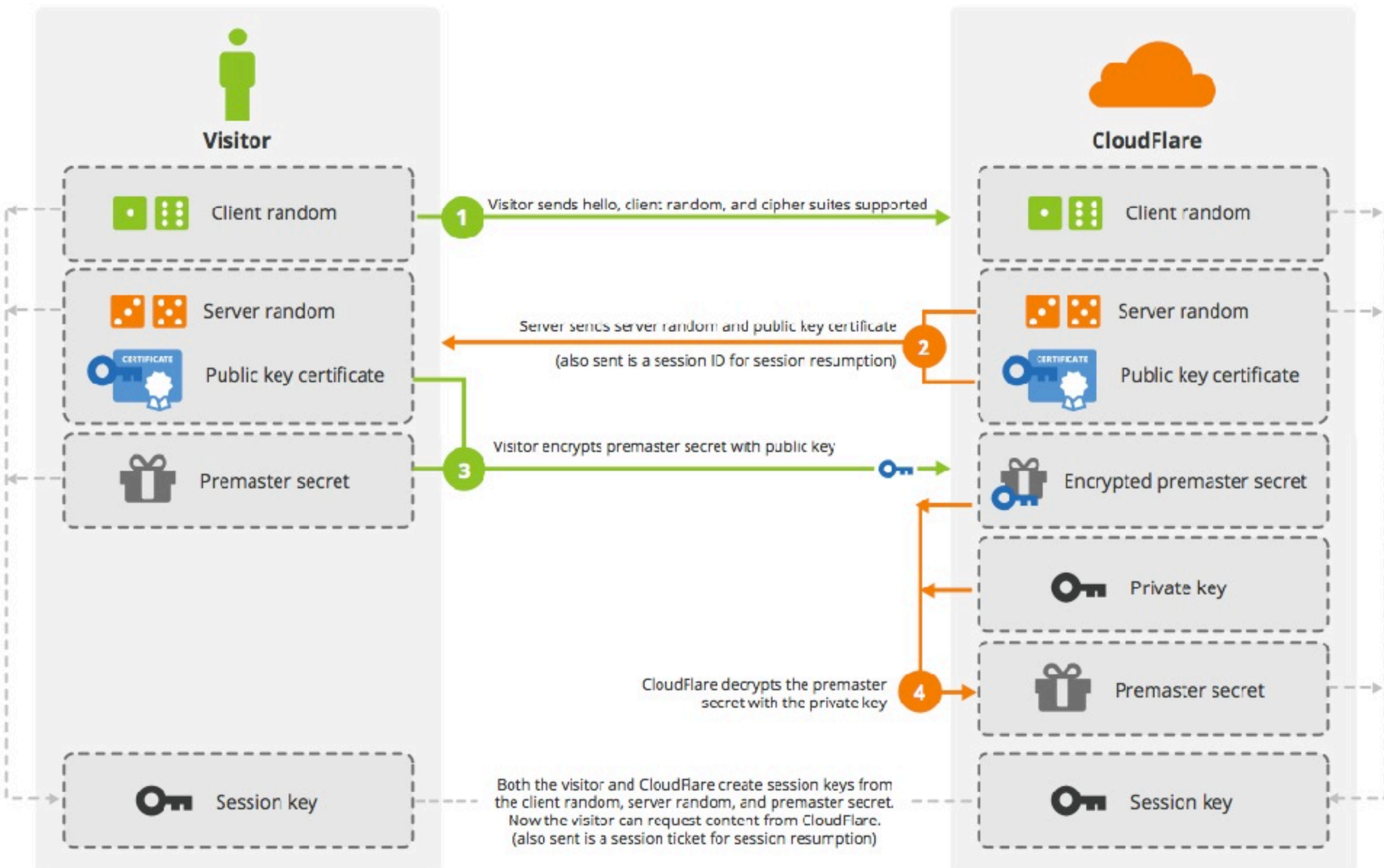


Figure 5: **Ephemeral Exchange Value Reuse**—We measured how long Alexa Top Million websites served identical DHE and ECDHE values (note vertical scale is cropped).

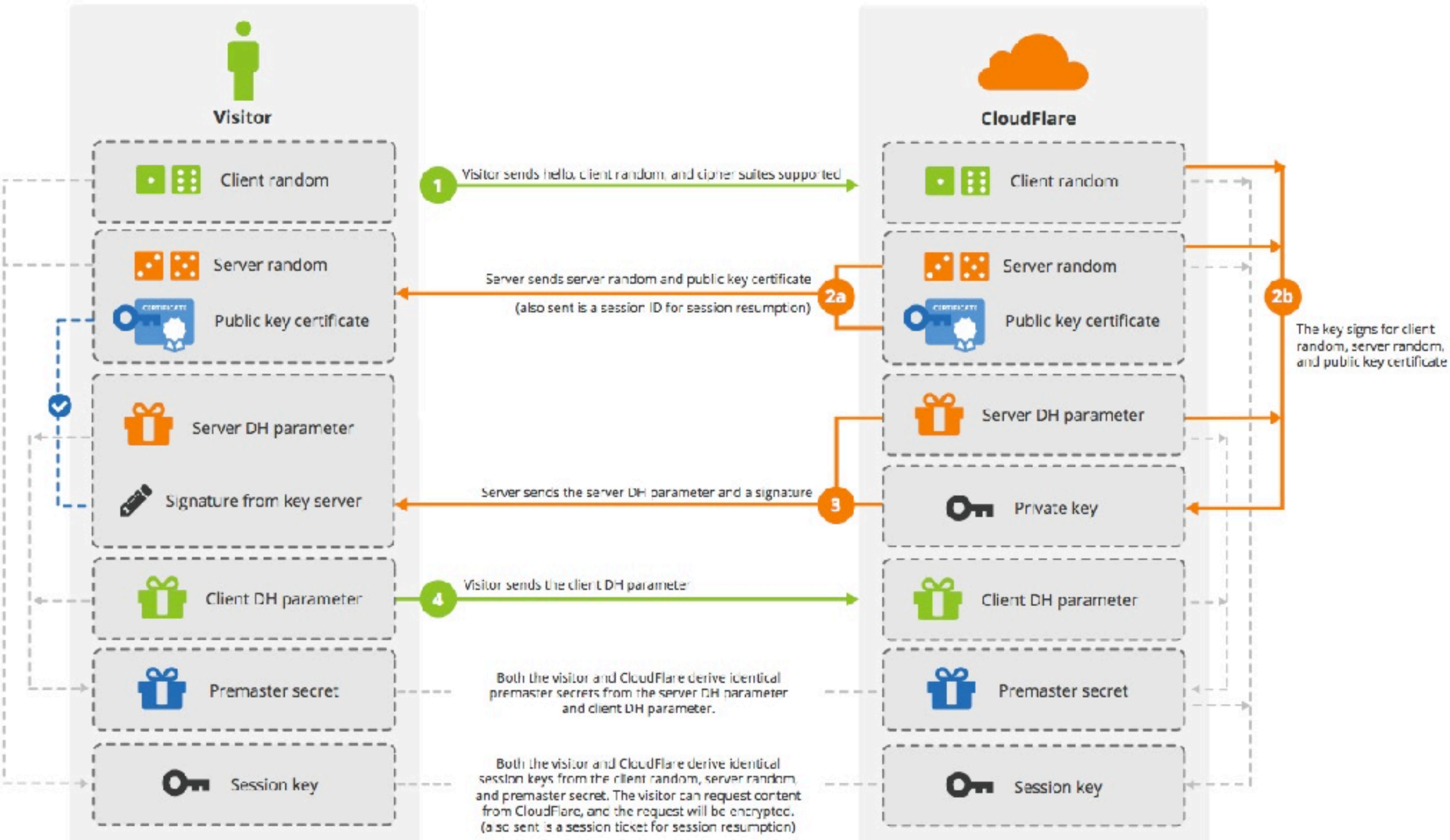
SSL Handshake (RSA) Without Keyless SSL

Handshake



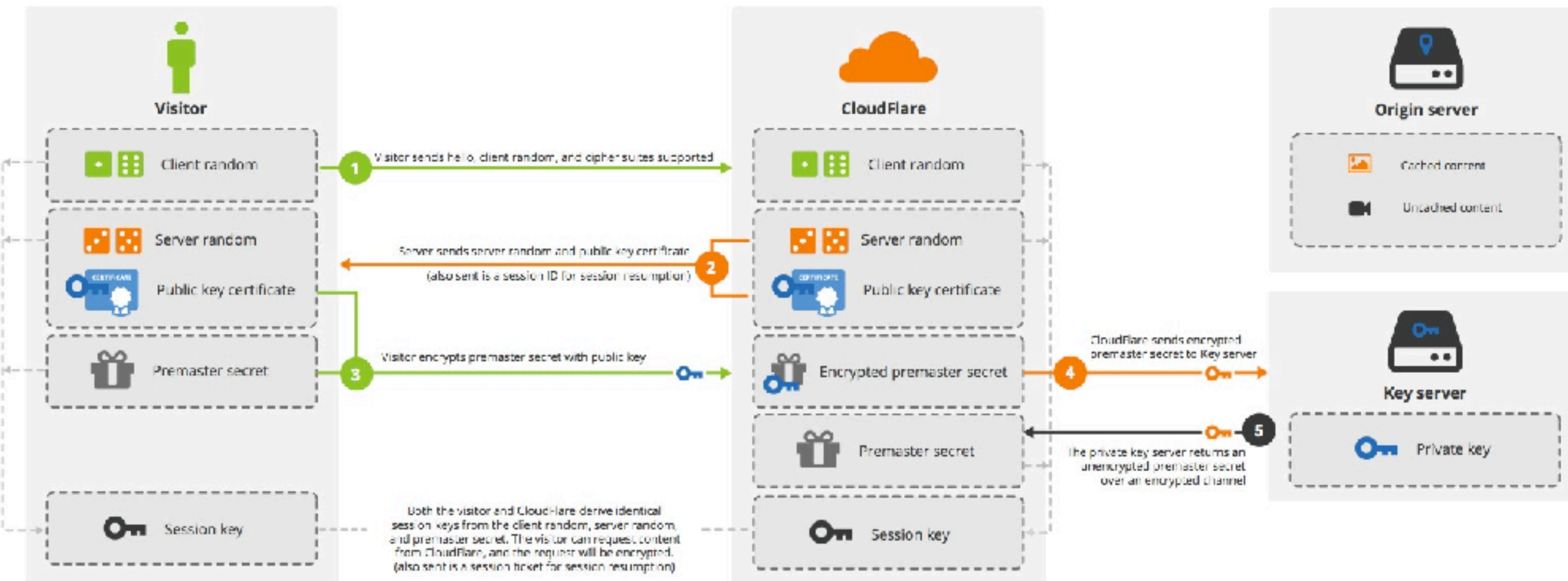
SSL Handshake (Diffie-Hellman) Without Keyless SSL

Handshake



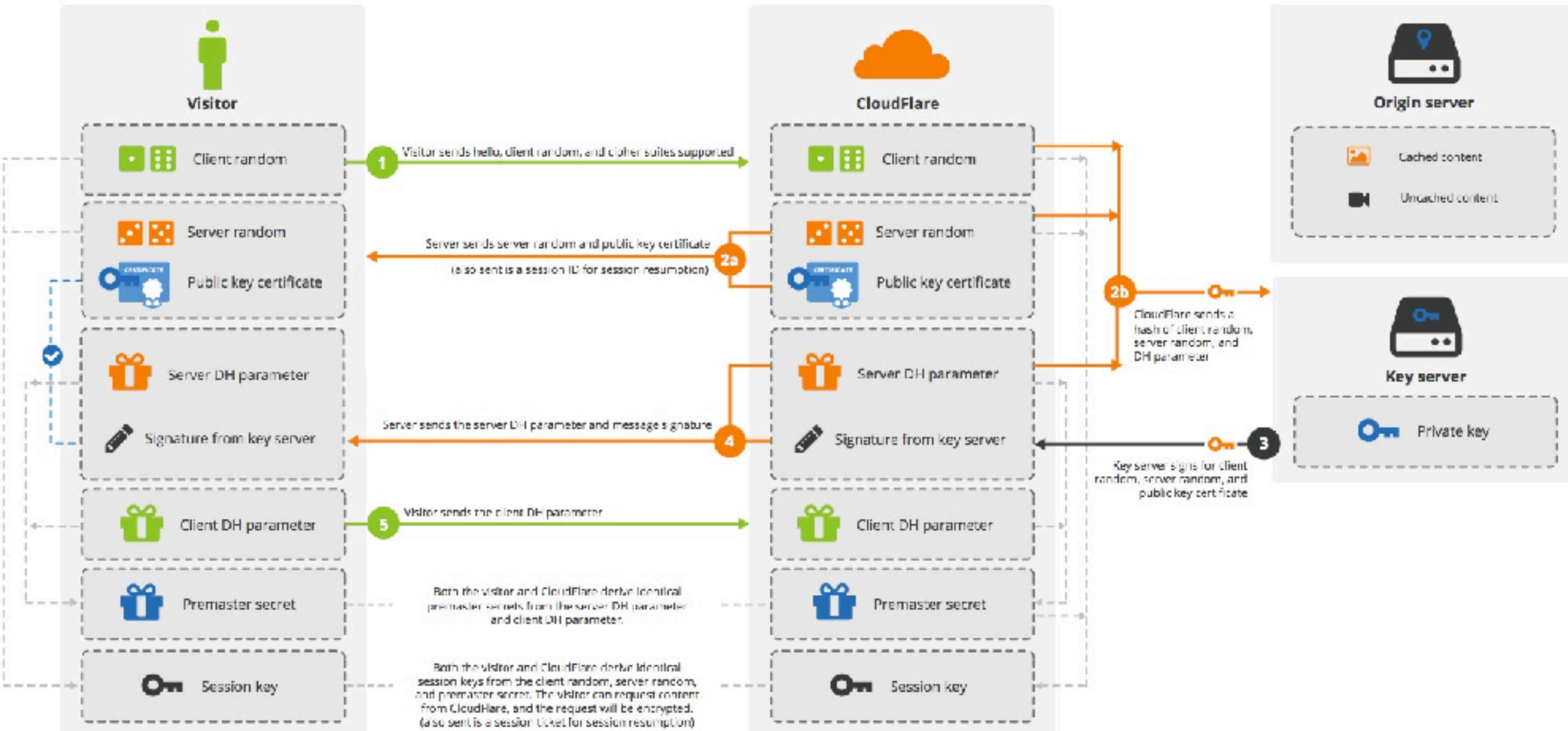
CloudFlare Keyless SSL (RSA)

Handshake

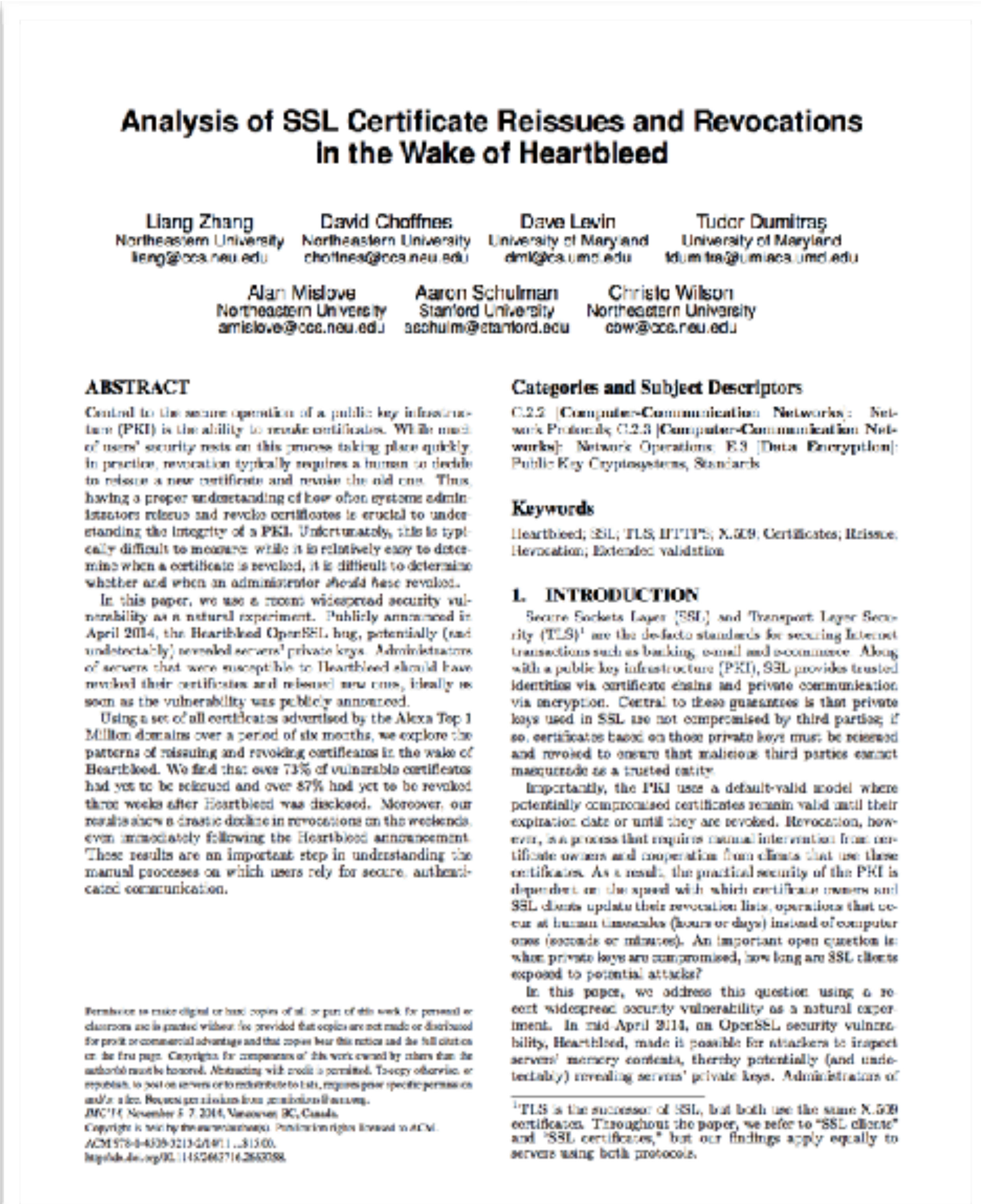


CloudFlare Keyless SSL (Diffie-Hellman)

Handshake



POOR CERTIFICATE MANAGEMENT



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

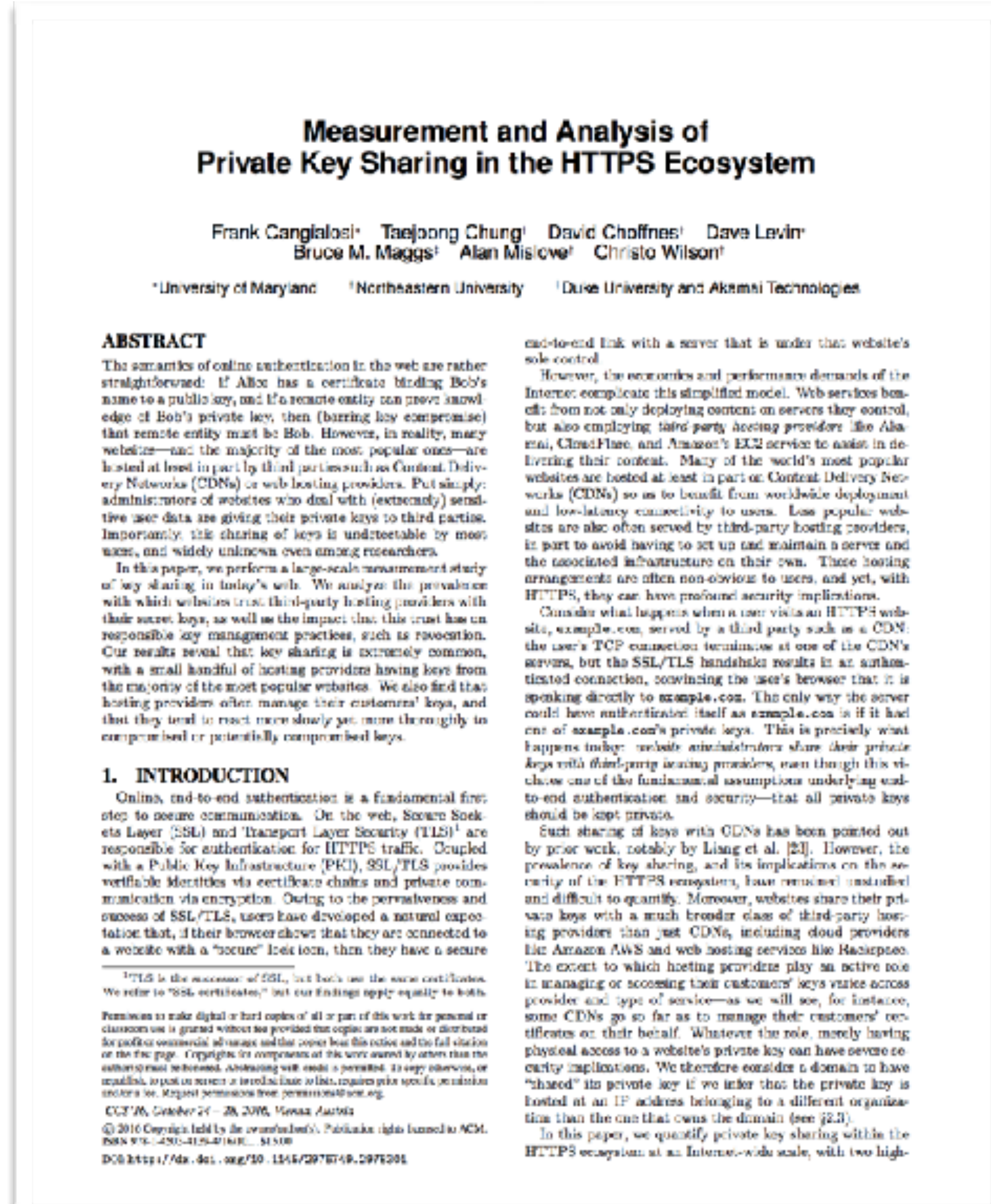
Liang Zhang, Northeastern University, liang@ccs.neu.edu
David Choffnes, Northeastern University, choffnes@ccs.neu.edu
Dave Levin, University of Maryland, dml@cs.umd.edu
Tudor Dumitraş, University of Maryland, tdumitra@umiacs.umd.edu
Alan Mislove, Northeastern University, amislove@ccs.neu.edu
Aaron Schulman, Stanford University, aschulm@stanford.edu
Christo Wilson, Northeastern University, cbw@ccs.neu.edu

ABSTRACT

Categories and Subject Descriptors

Keywords

1. INTRODUCTION



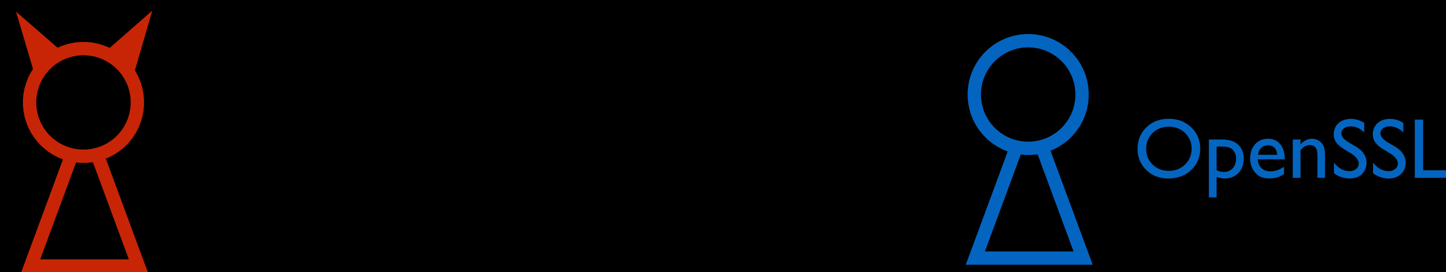
Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

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

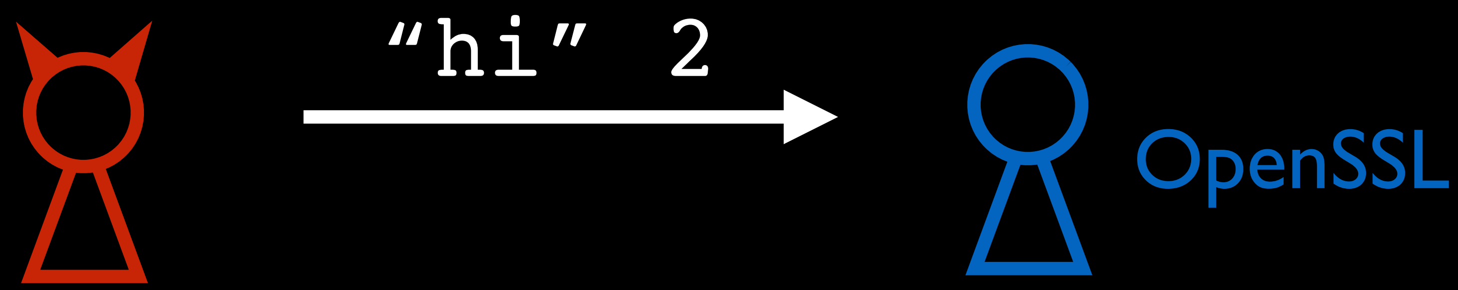
ABSTRACT

1. INTRODUCTION

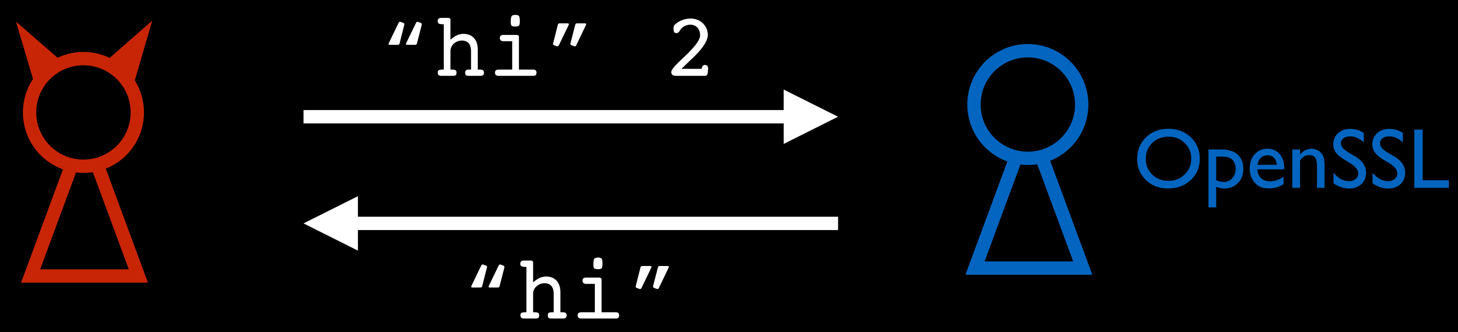
Heartbleed



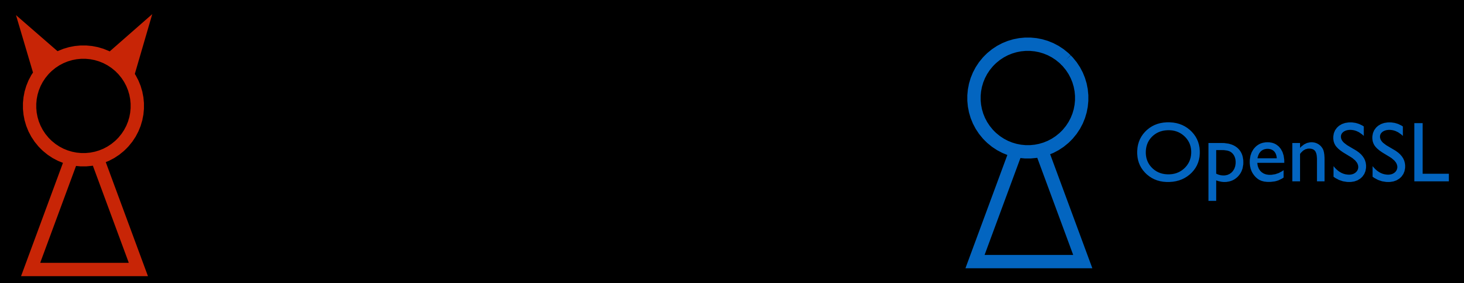
Heartbleed



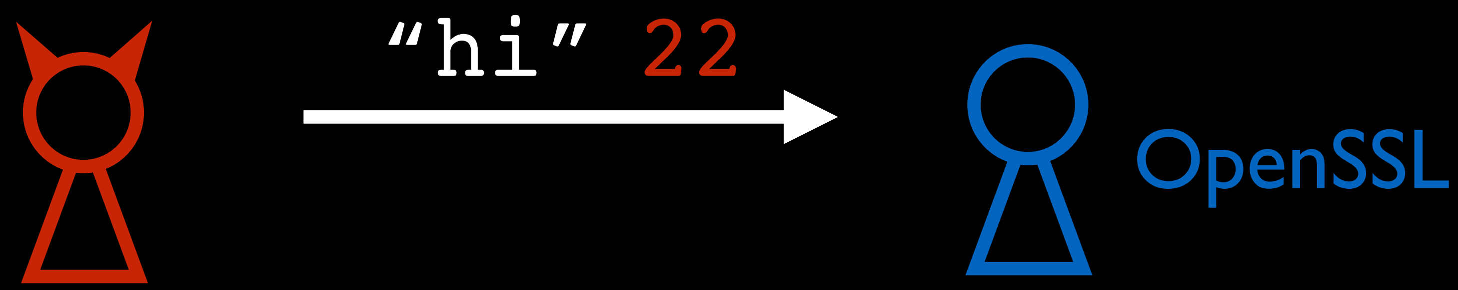
Heartbleed



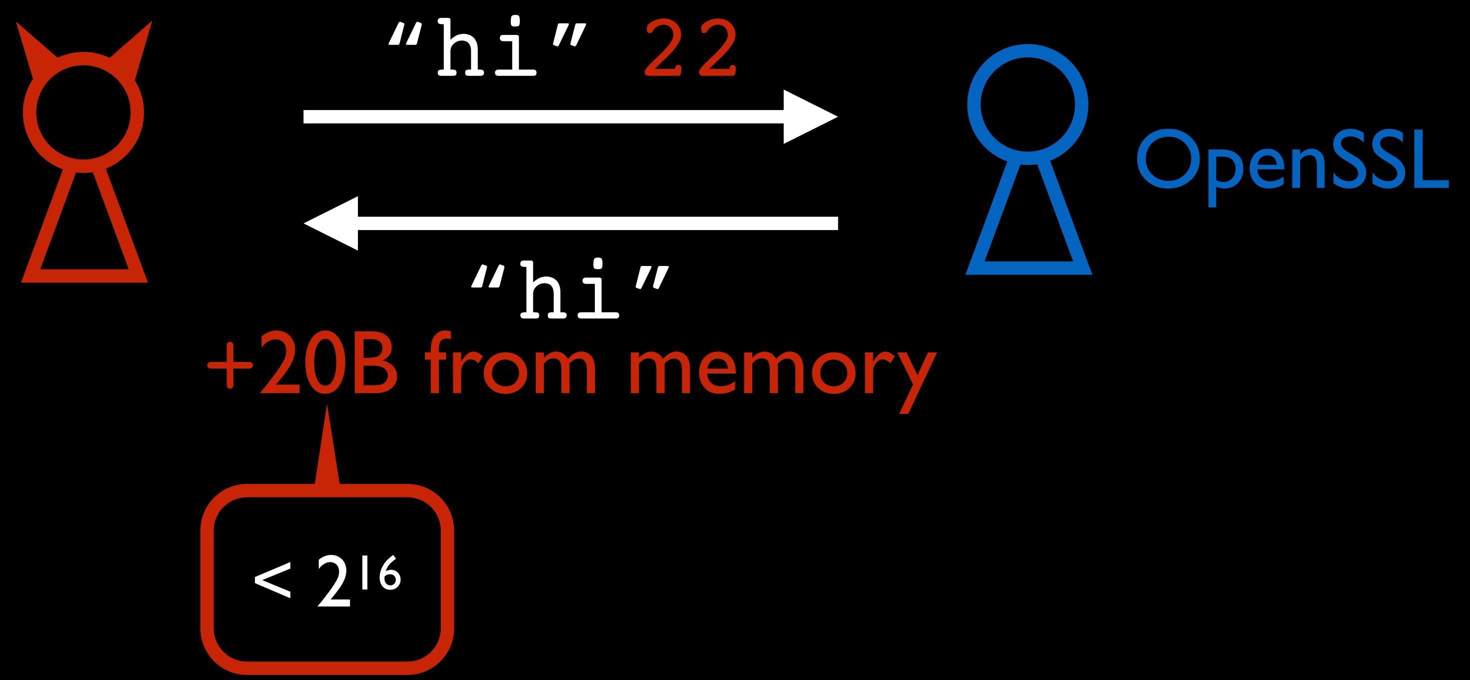
Heartbleed



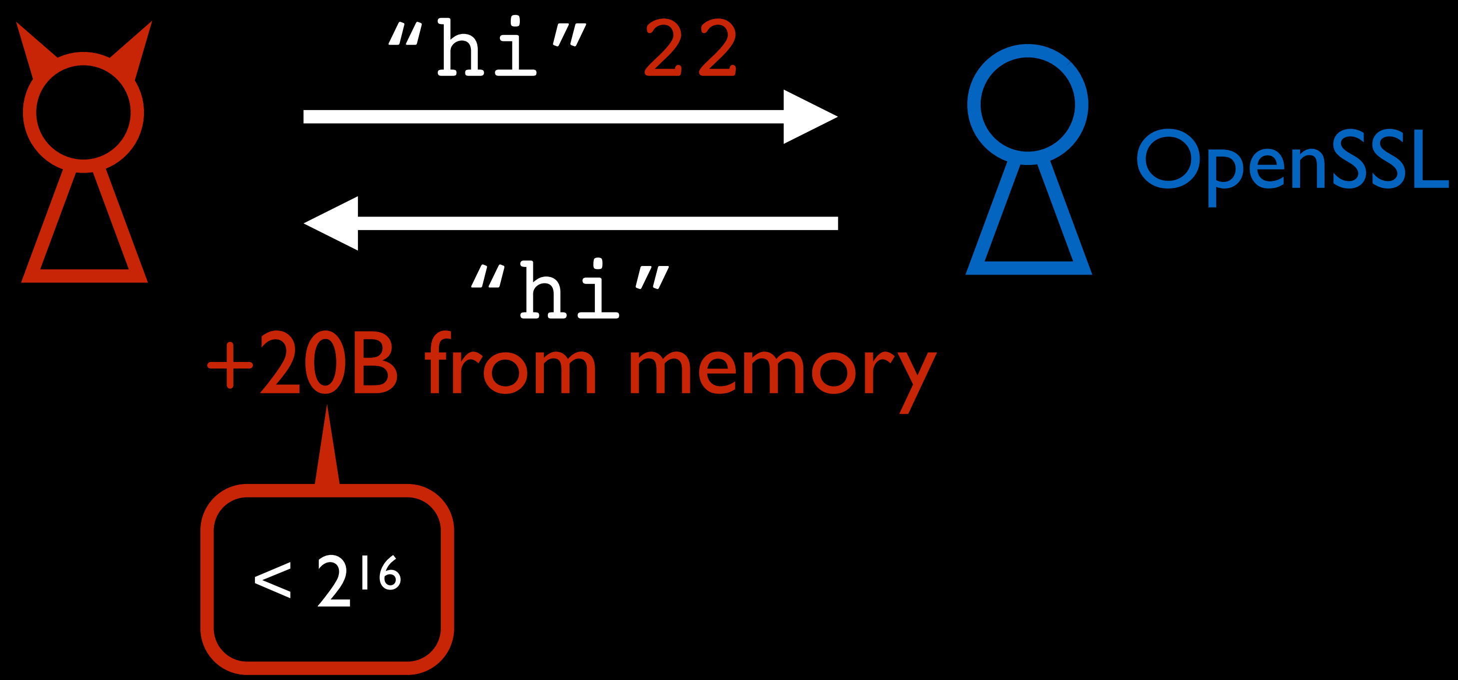
Heartbleed



Heartbleed



Heartbleed



Potentially reveals user data and private keys

Heartbleed exploits were undetectable

Why study Heartbleed?

Discovered	Akamai patched	Publicly announced
03/21	04/02	04/07

Why study Heartbleed?

Discovered	Akamai patched	Publicly announced
03/21	04/02	04/07

Every vulnerable website should have:

1. Patched
2. Revoked
3. Reissued

Why study Heartbleed?

Discovered	Akamai patched	Publicly announced
03/21	04/02	04/07

Every vulnerable website should have:

1. Patched
2. Revoked
3. Reissued

Heartbleed is a natural experiment:

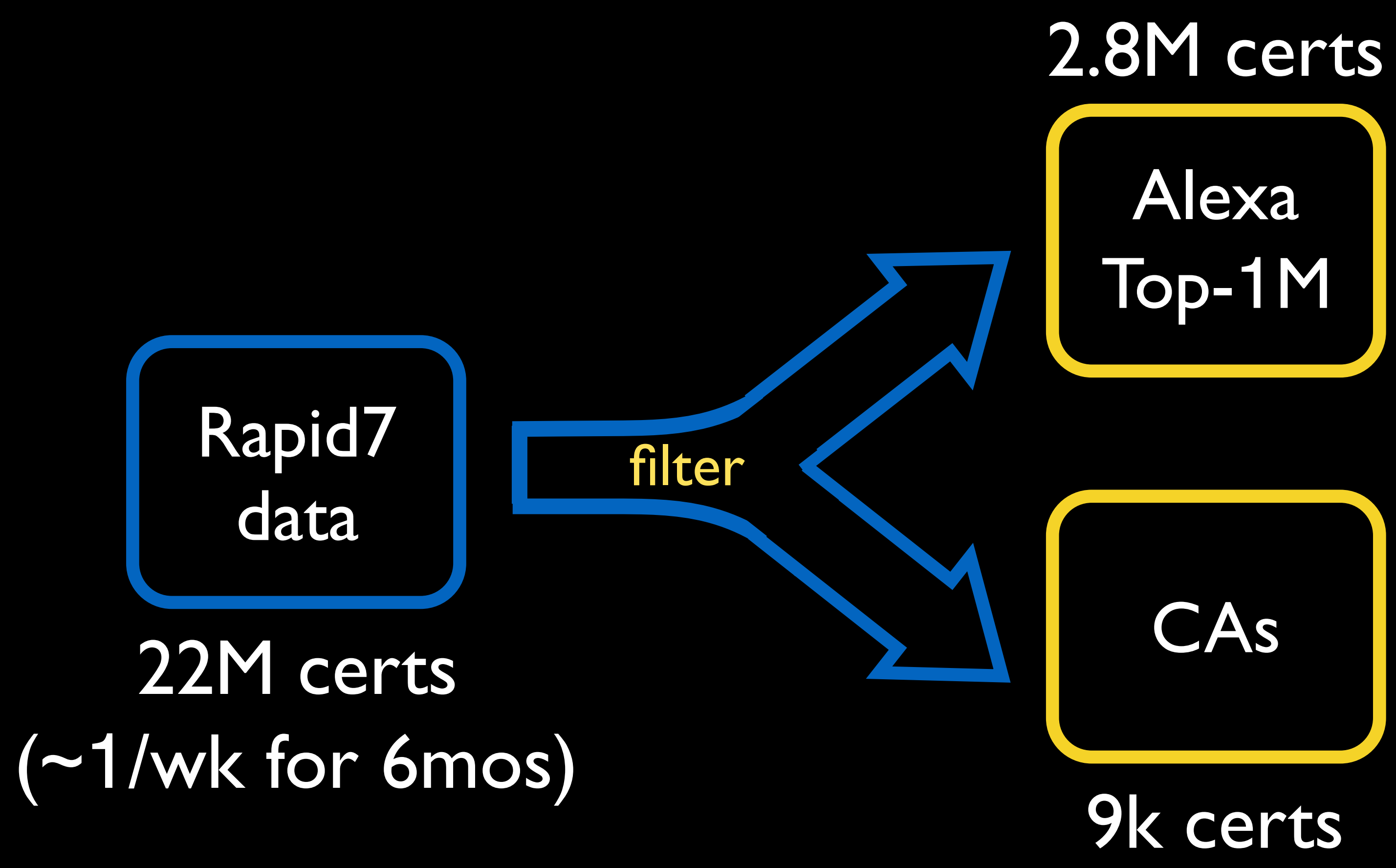
How quickly and thoroughly do administrators act?

Dataset

Rapid7
data

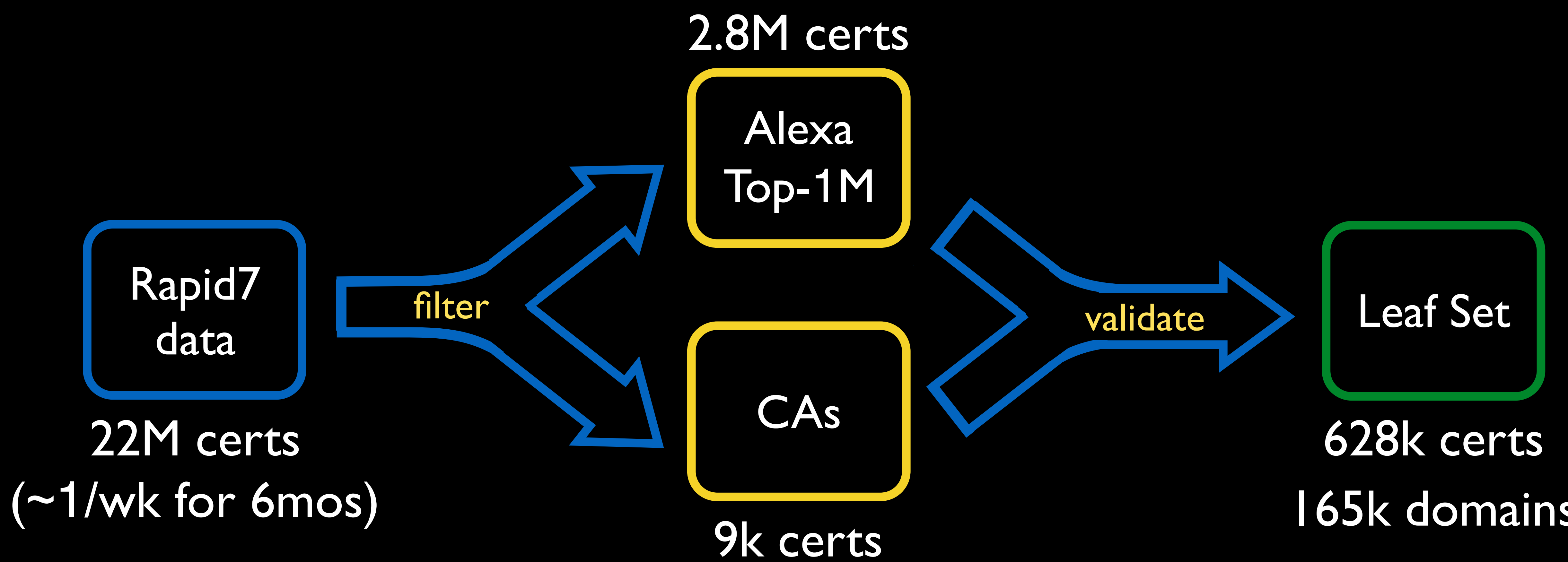
22M certs
(~1/wk for 6mos)

Dataset

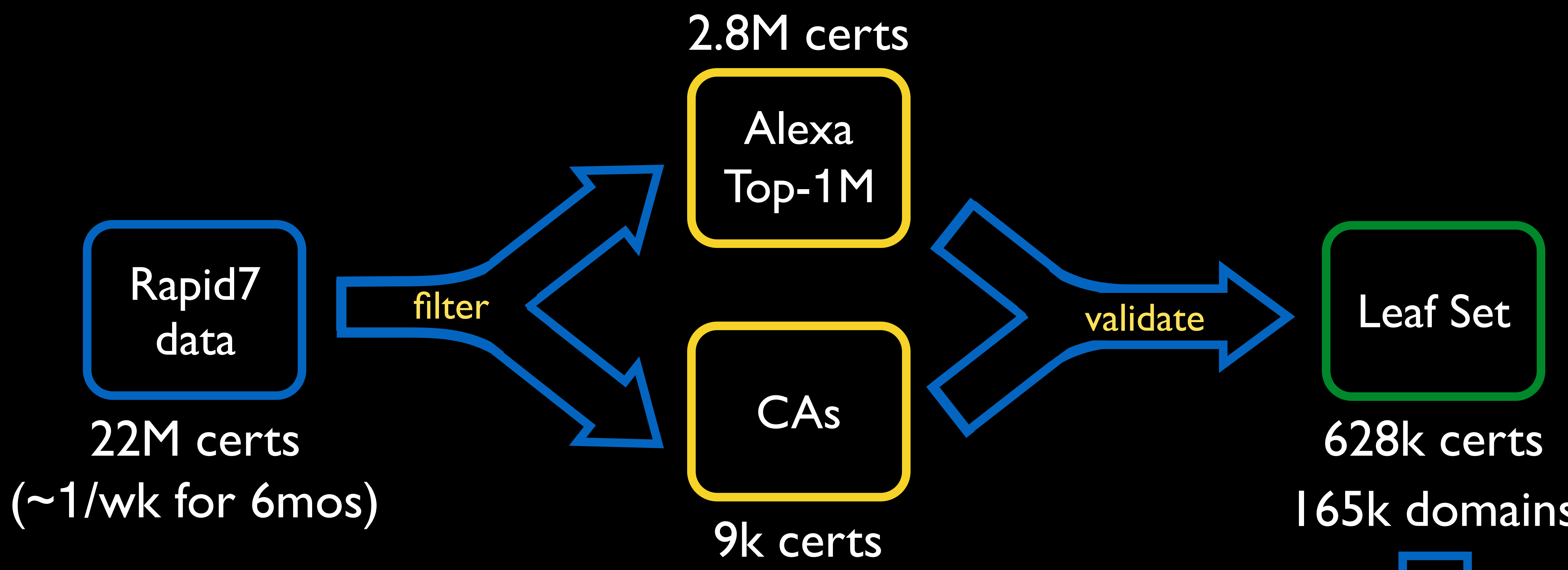


2.8M certs
Alexa Top-1M
Rapid7 data
filter
CAs
22M certs
(~1/wk for 6mos)
9k certs

Dataset

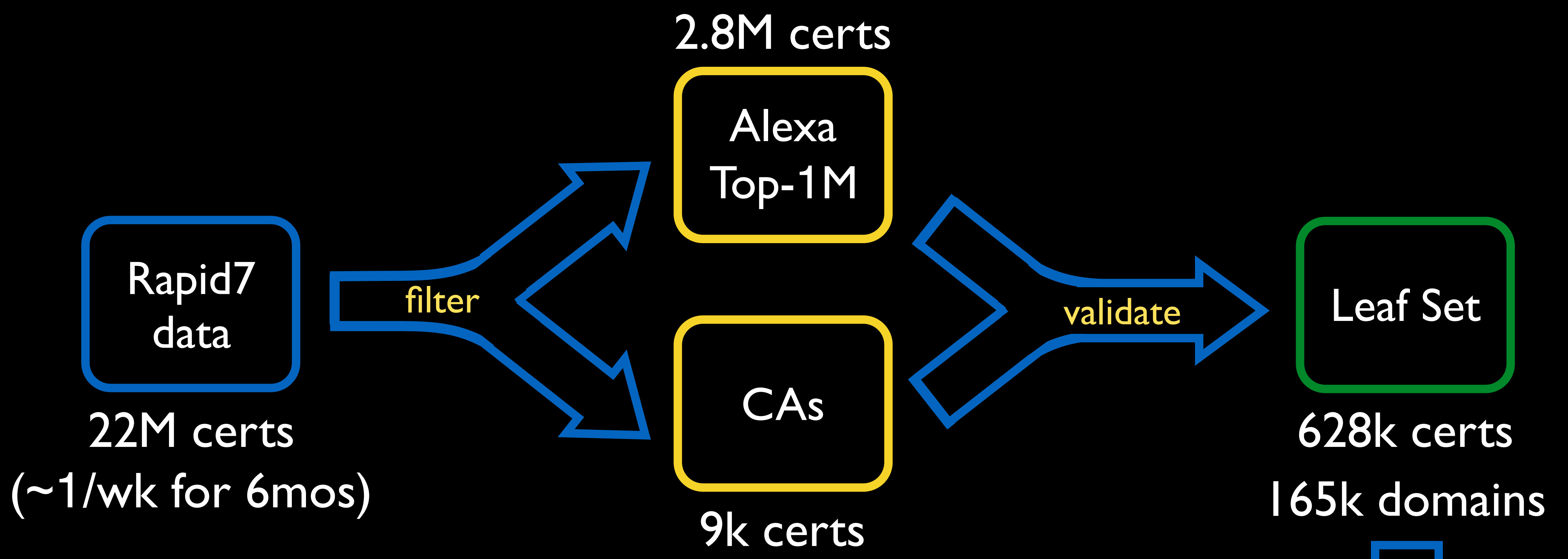

2.8M certs
Alexa Top-1M
Rapid7 data
filter
CAs
validate
Leaf Set
22M certs
(~1/wk for 6mos)
9k certs
628k certs
165k domains

Dataset



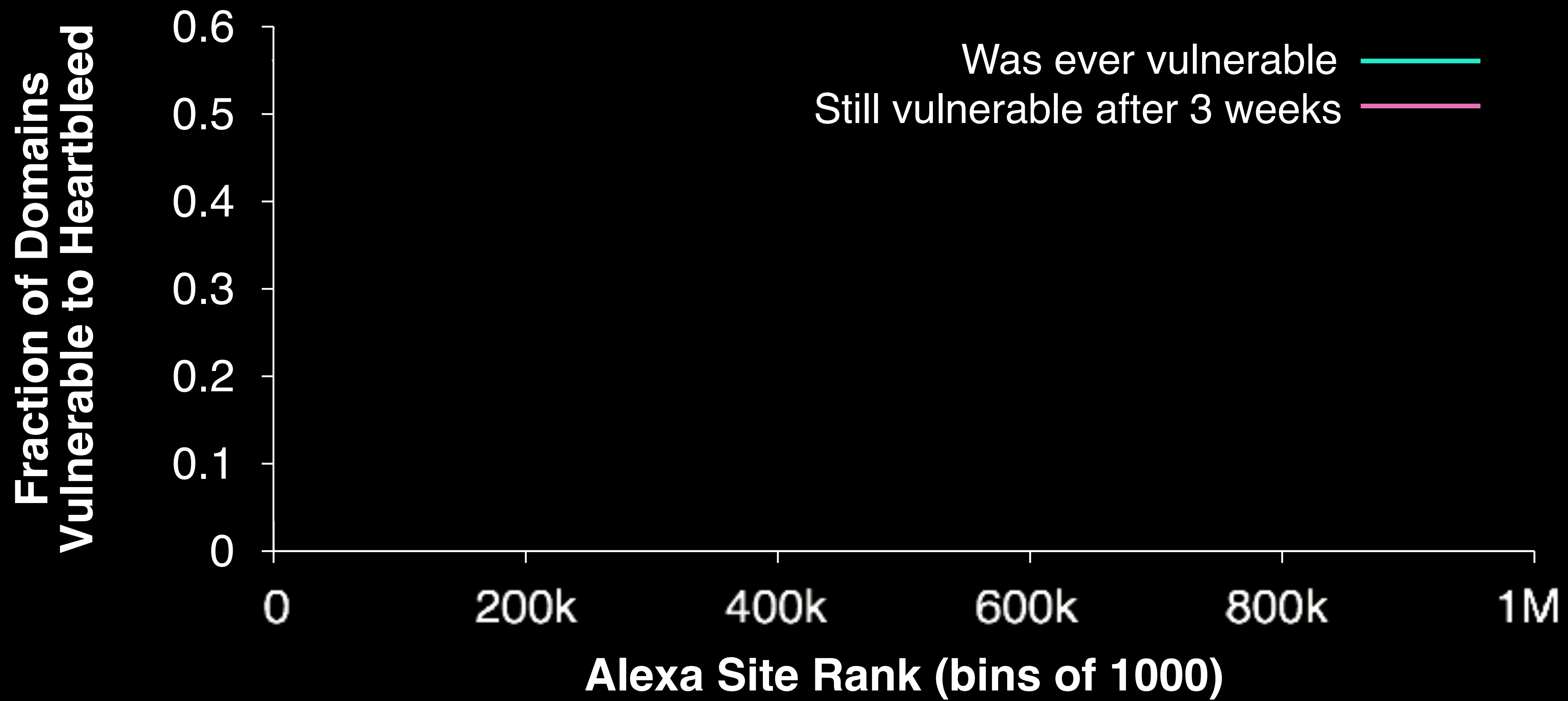
- Download CRLs
- Detect vulnerability
- Identify *Heartbleed-induced* reissues & revocations

Dataset



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

Prevalence and patch rates



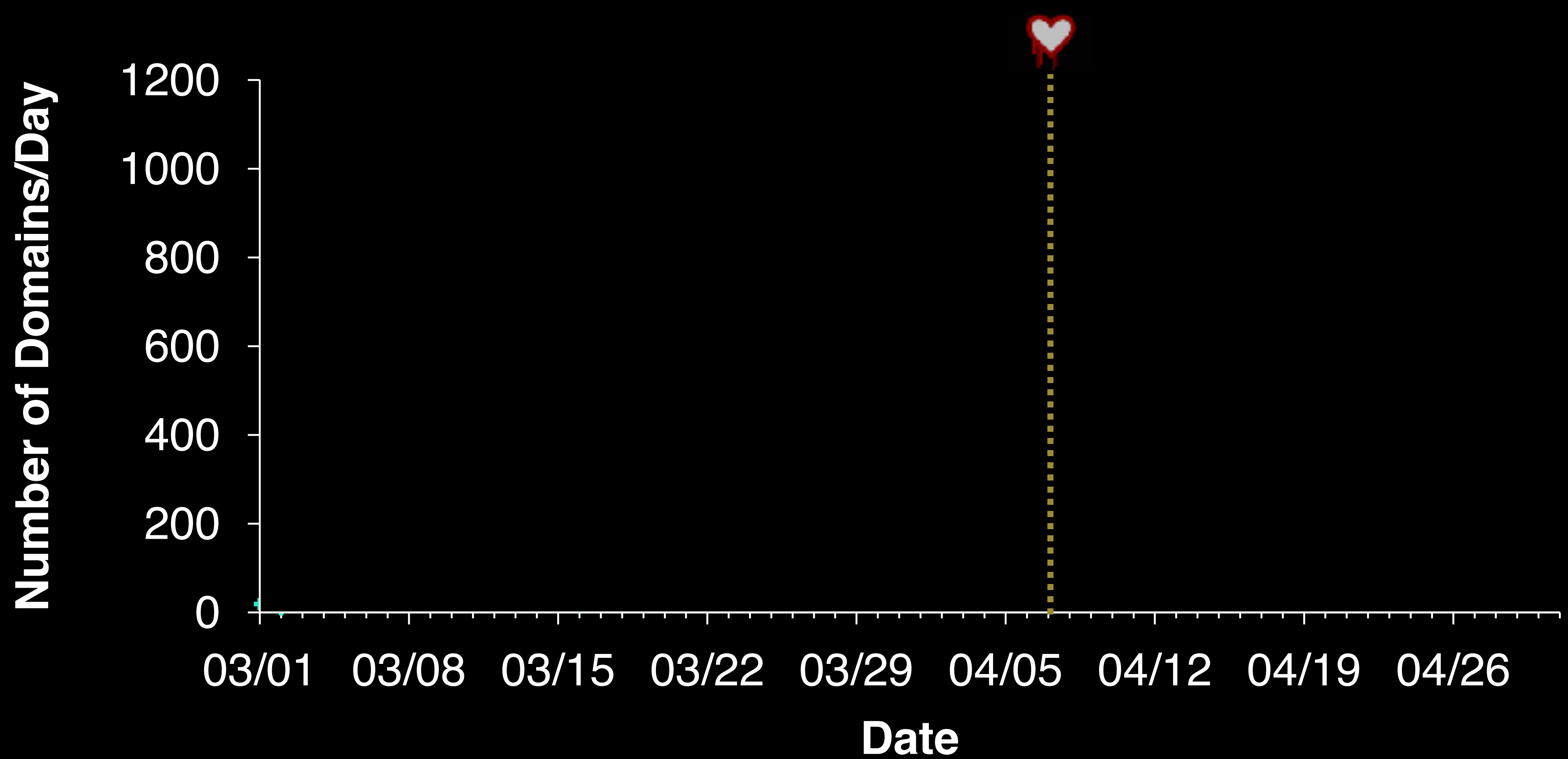
Prevalence and patch rates

Prevalence and patch rates

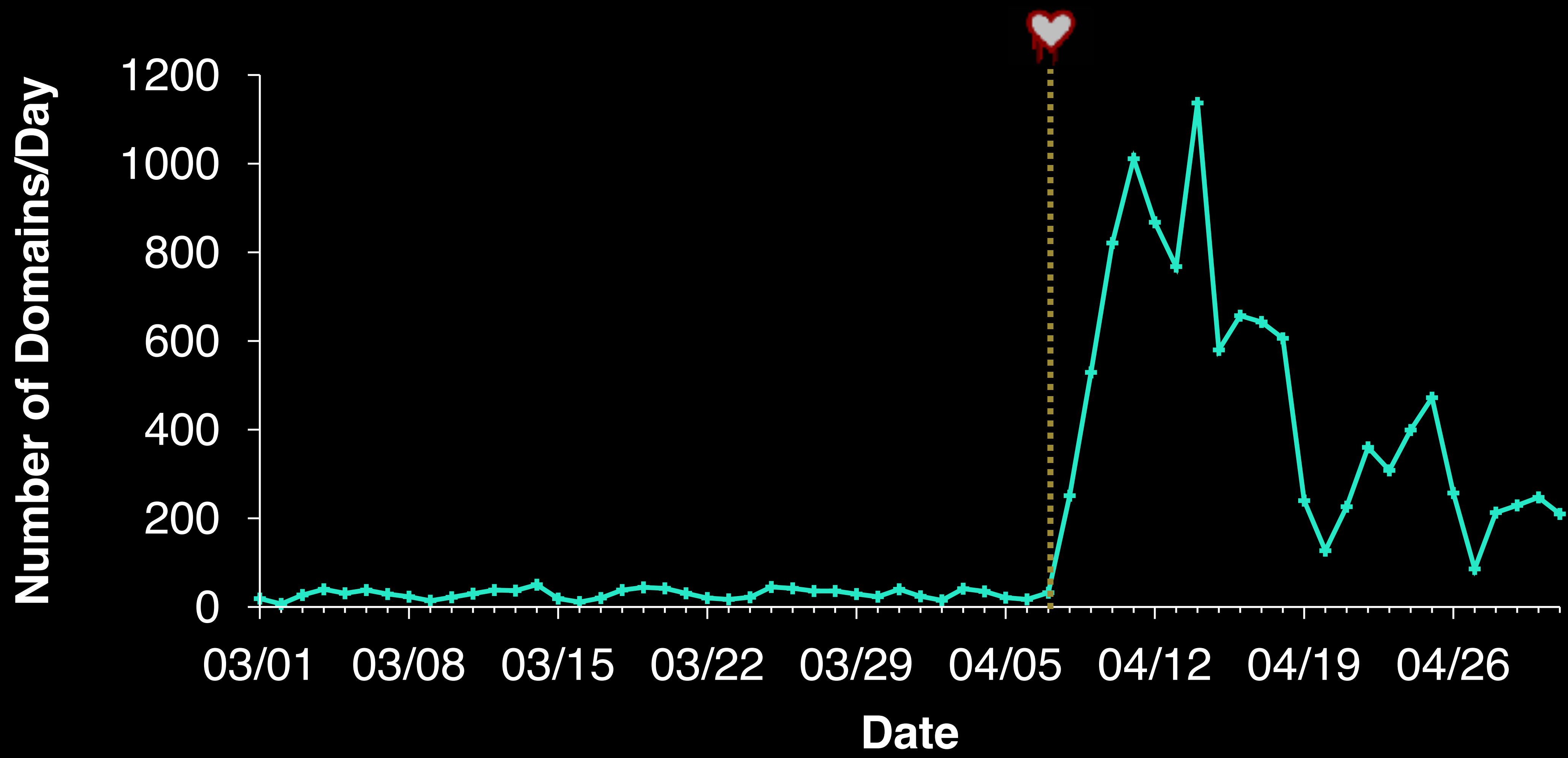
Patching rates are mostly positive

Only ~7% had not patched within 3 weeks

How quickly were certs revoked?

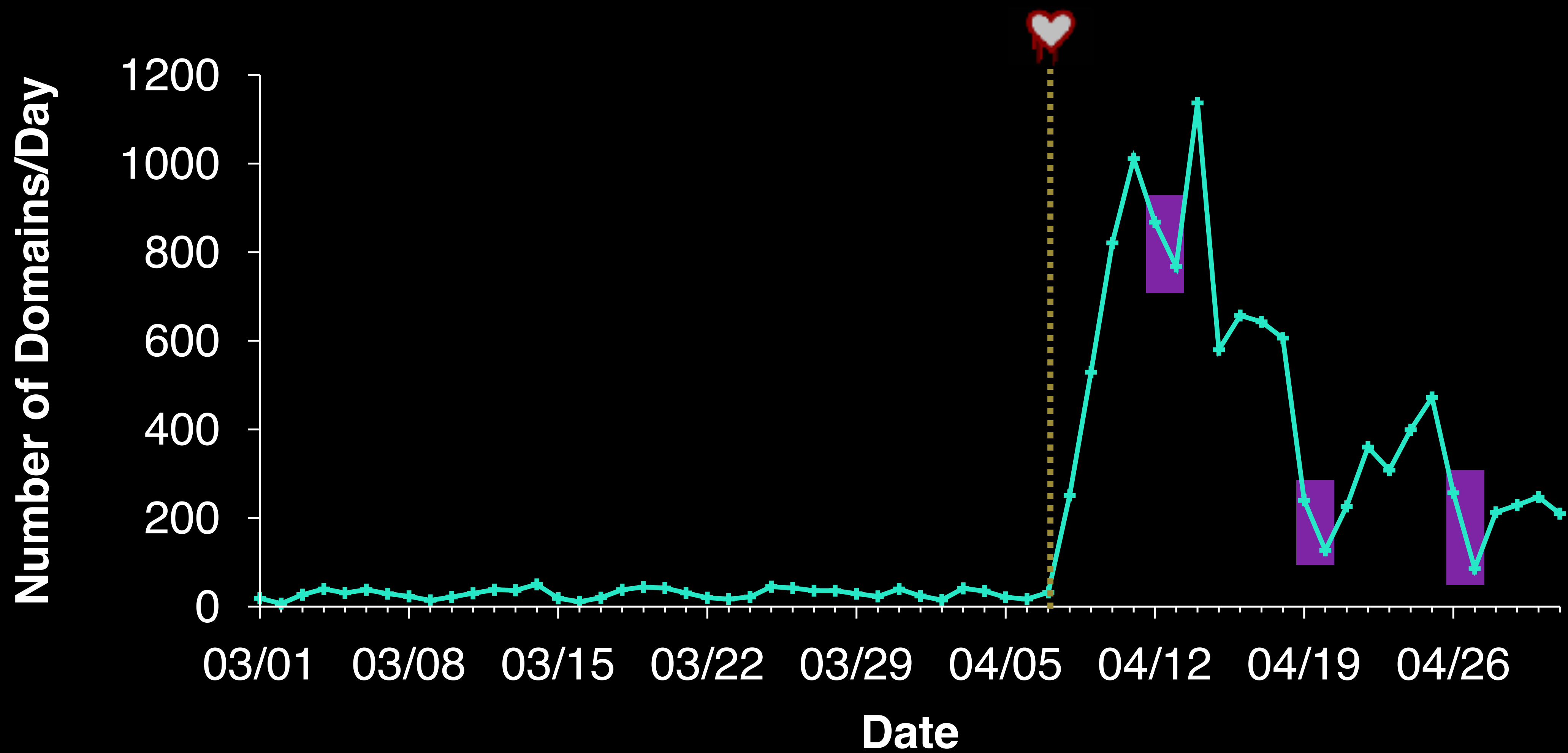


How quickly were certs revoked?



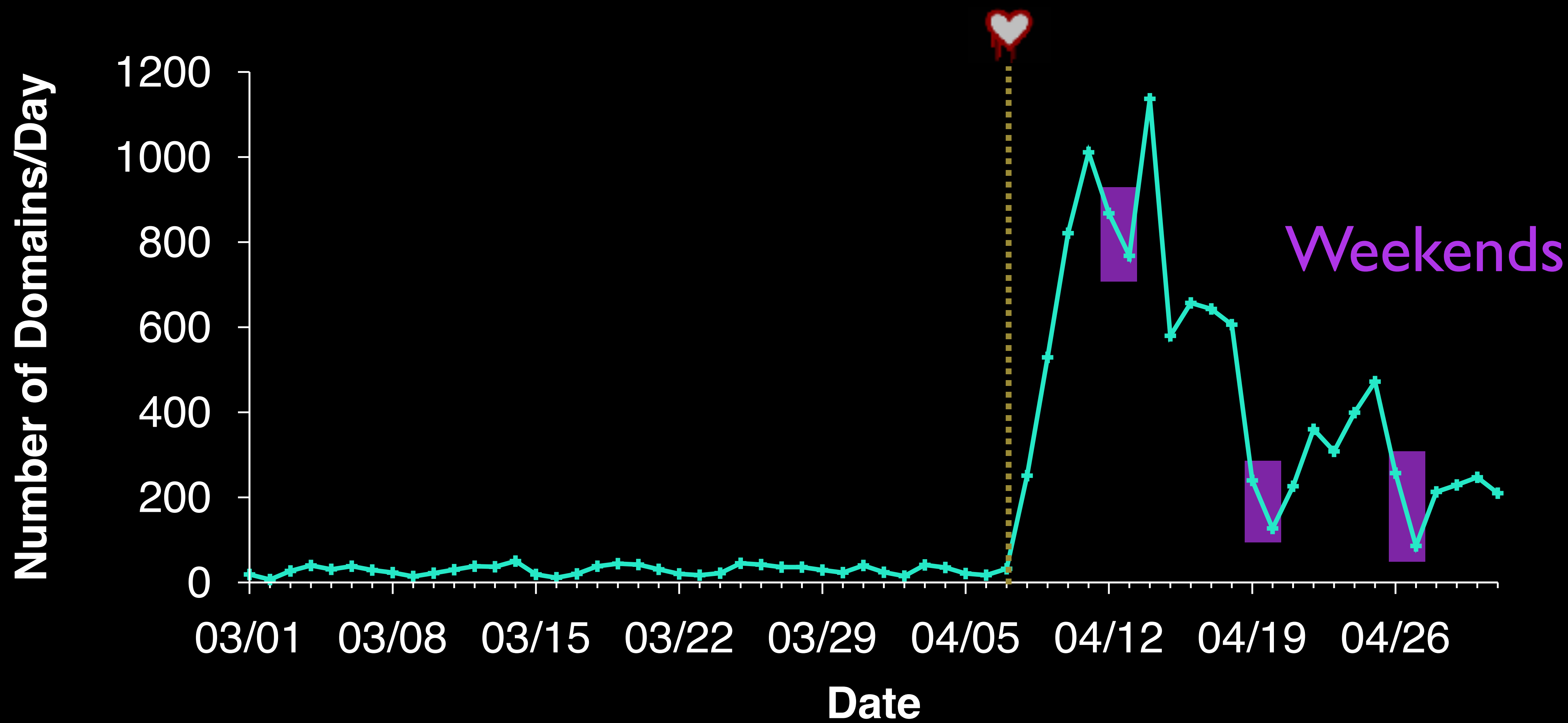
Reaction ramps up quickly

How quickly were certs revoked?



Reaction ramps up quickly

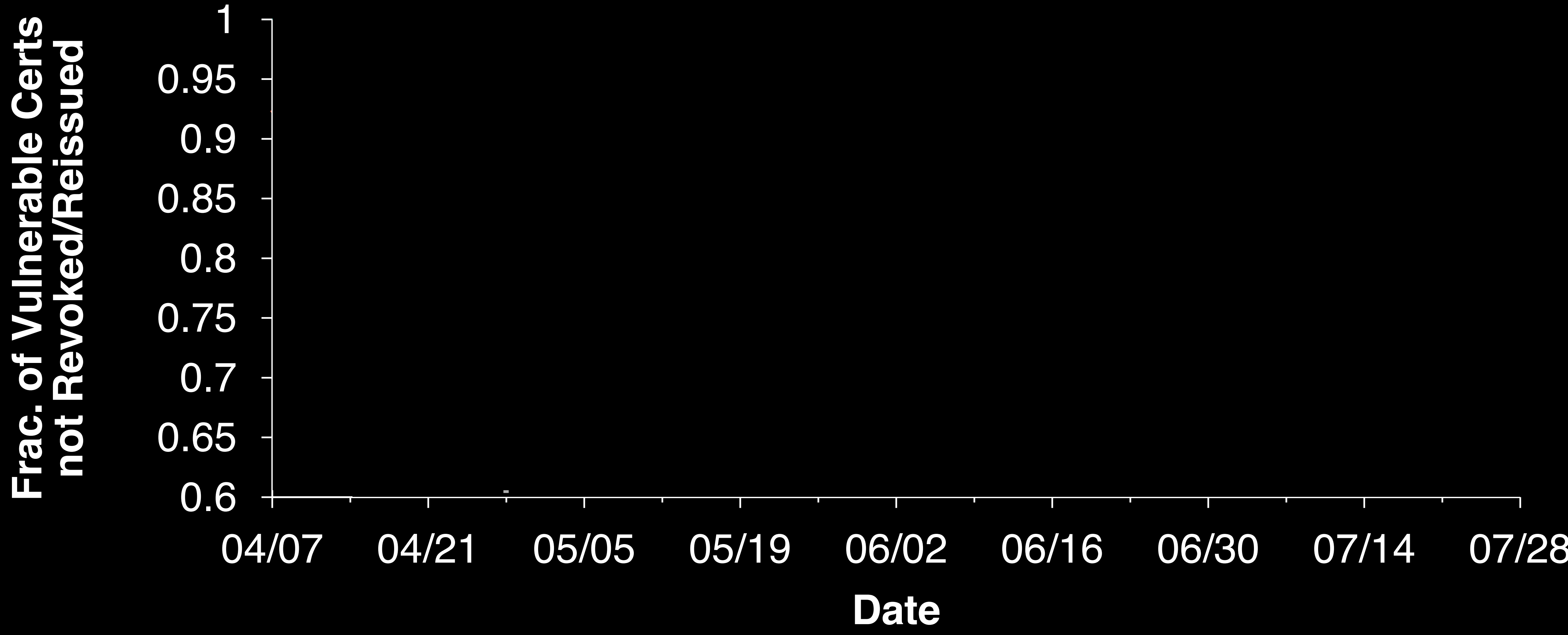
How quickly were certs revoked?



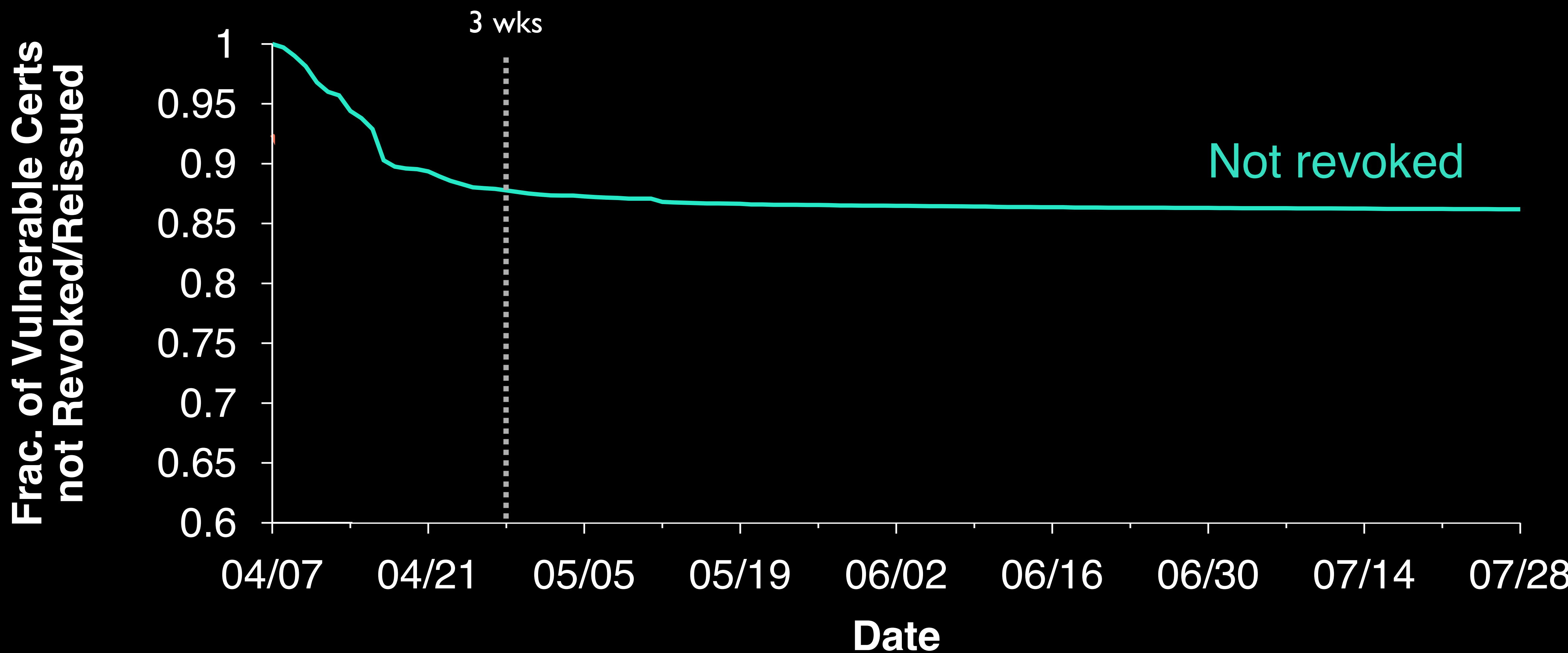
Reaction ramps up quickly

Security takes the weekends off

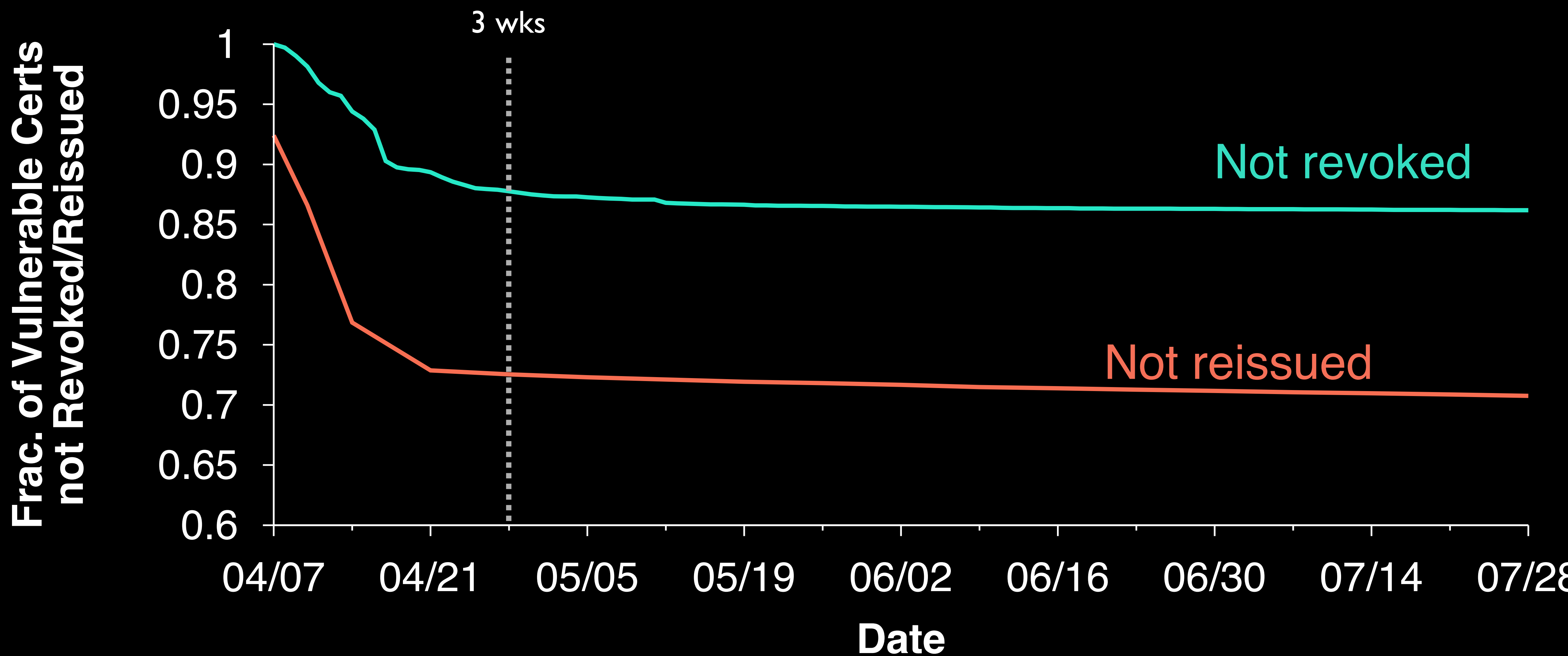
Certificate update rates



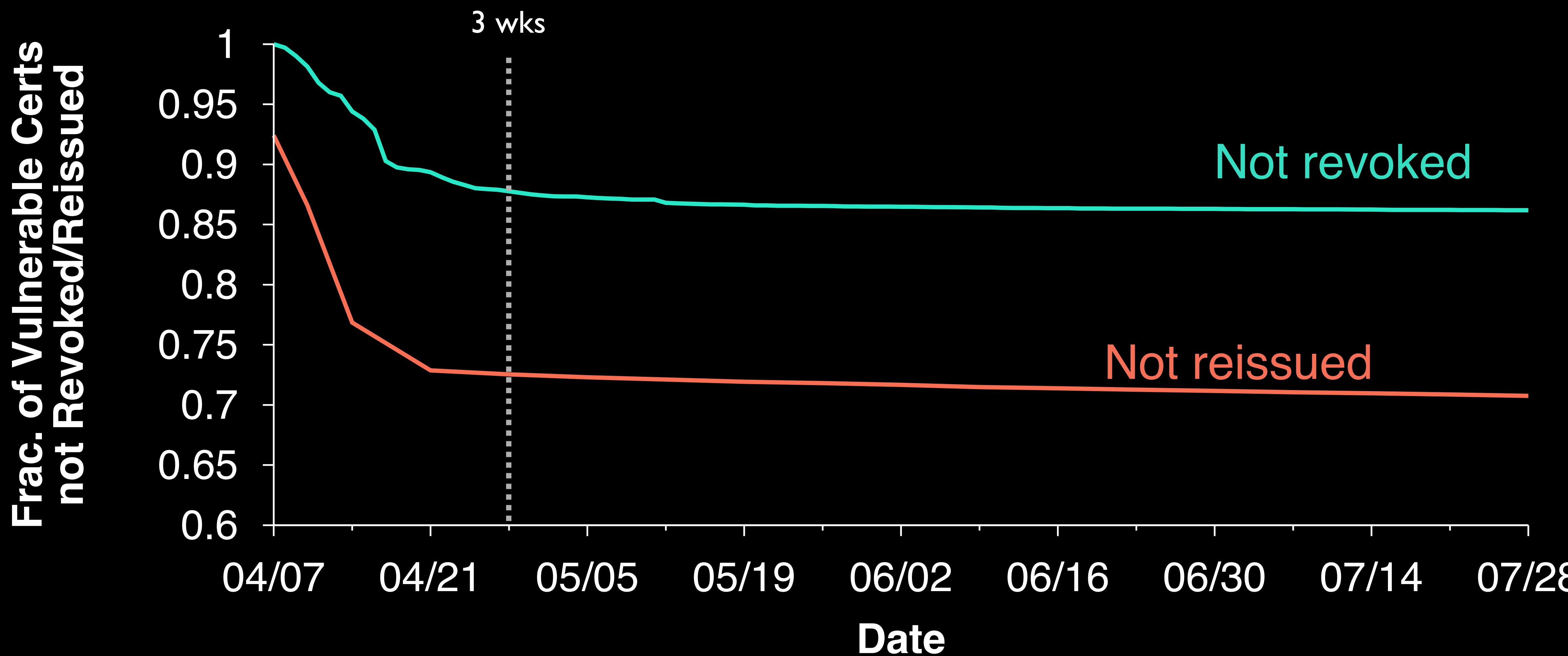
Certificate update rates



Certificate update rates



Certificate update rates



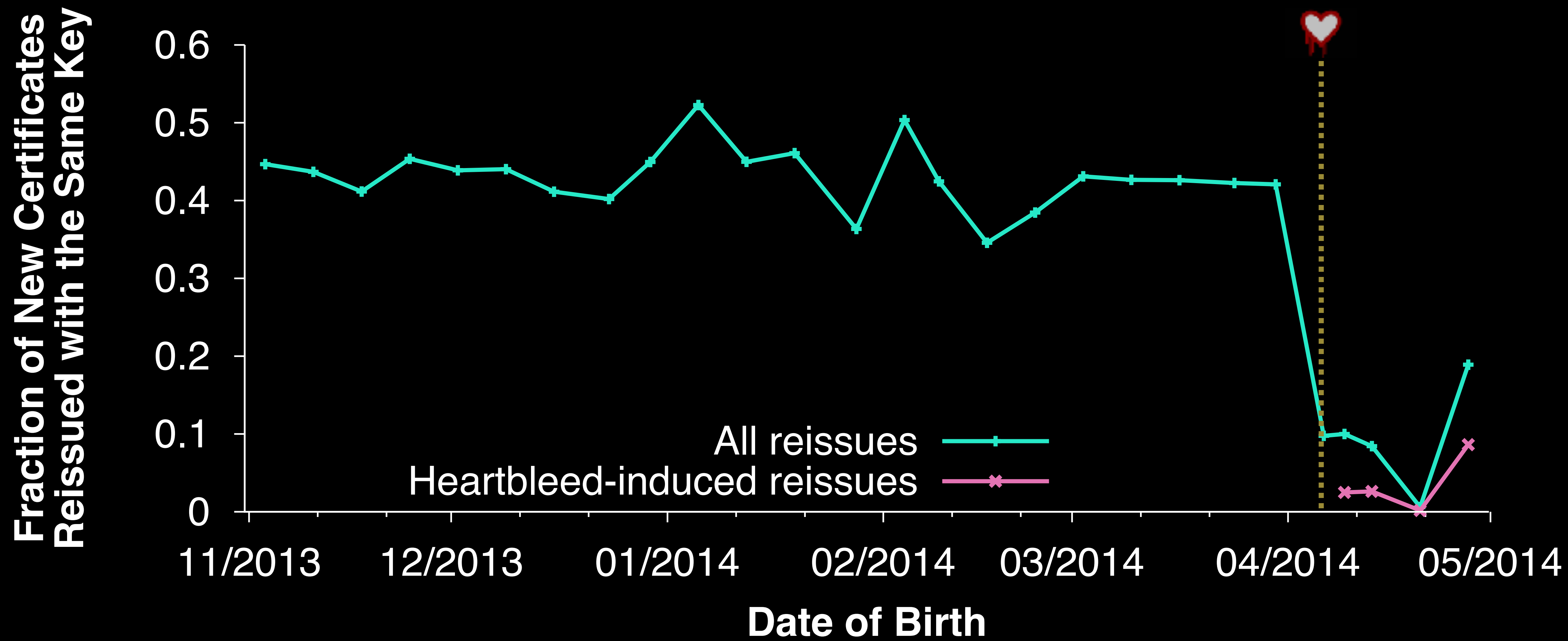
Similar pattern to patches:

Exponential drop-off, then levels out

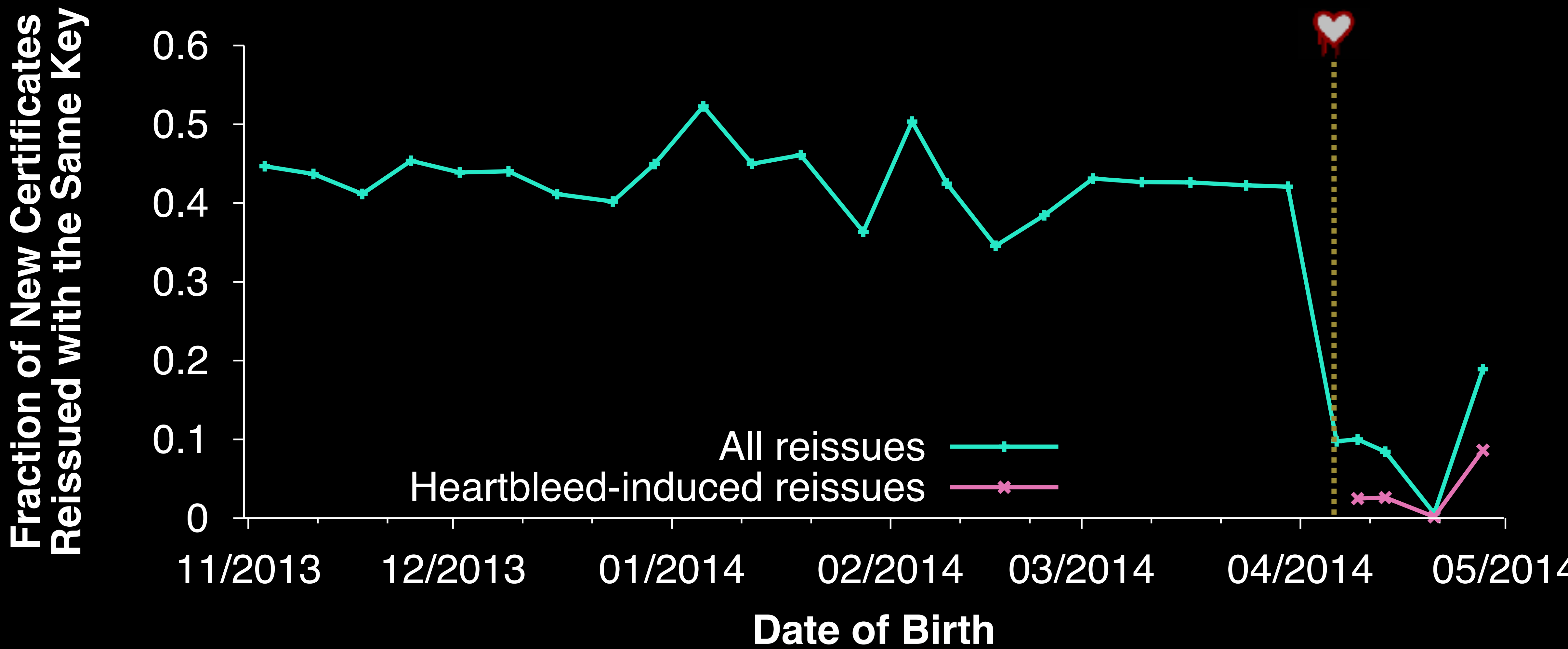
After 3 weeks: 13% Revoked 27% Reissued

Reissue ⇒ New key?

Reissue ⇒ New key?



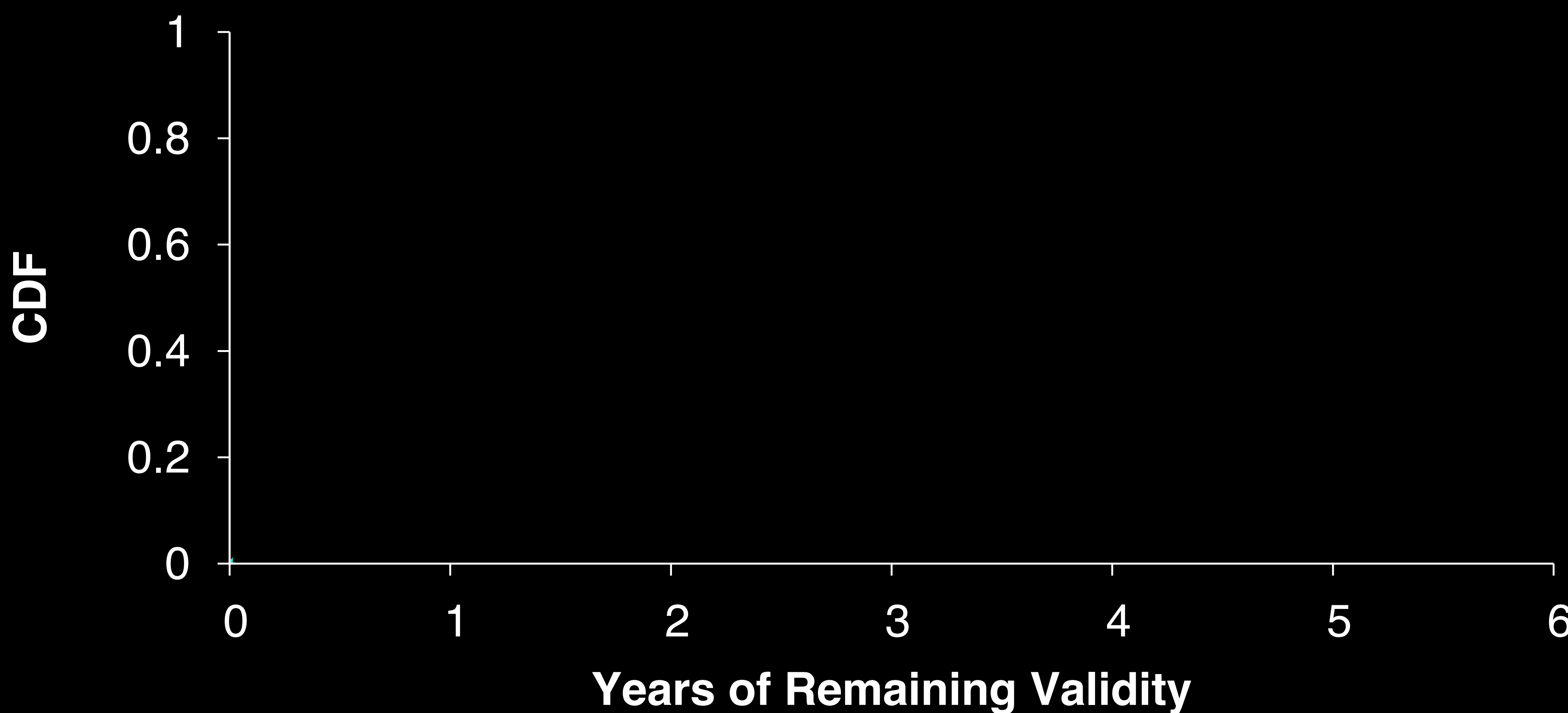
Reissue ⇒ New key?



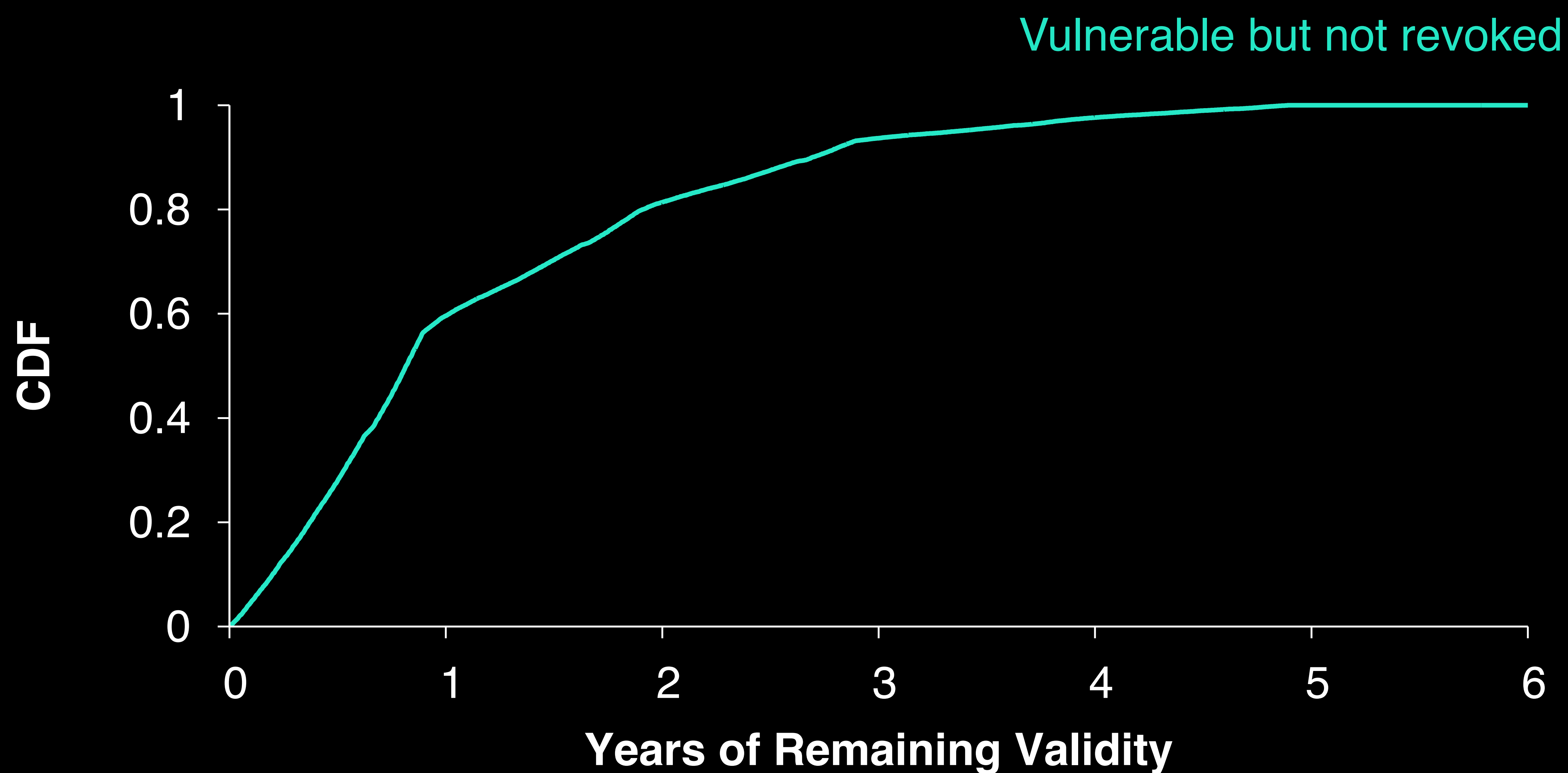
Reissuing the same key is common practice

4.1% Heartbleed-induced

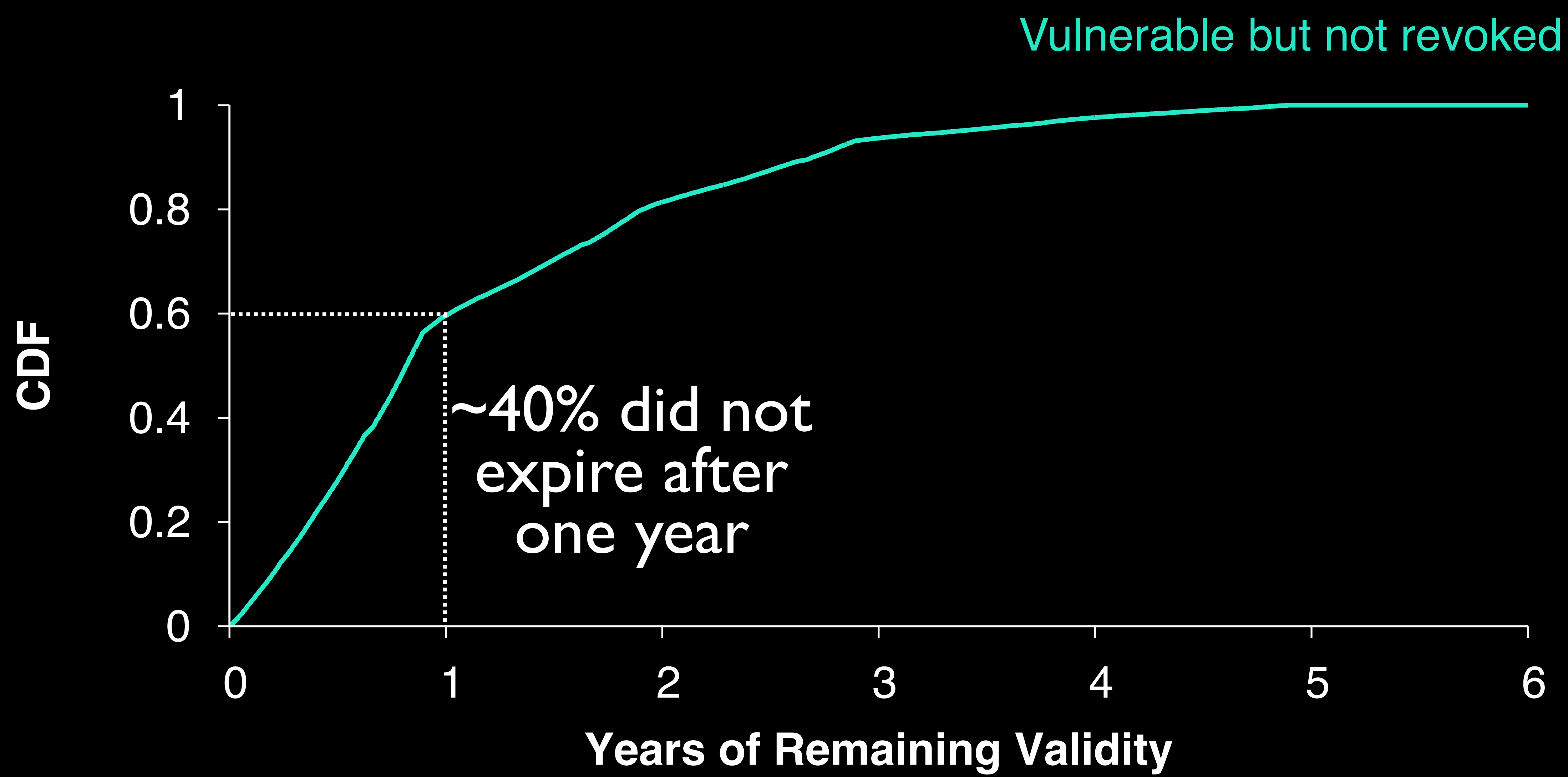
Can we wait for expiration?



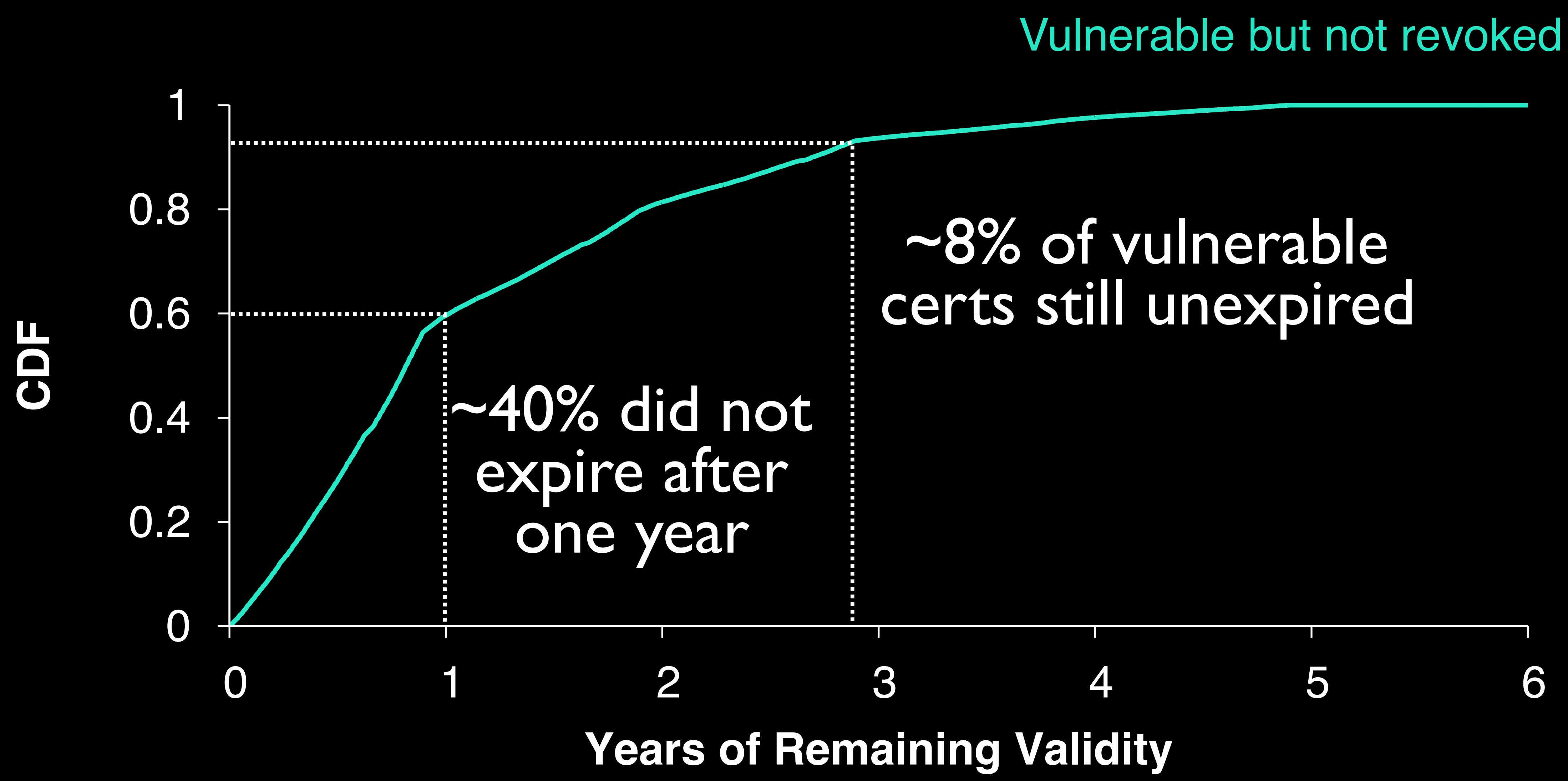
Can we wait for expiration?



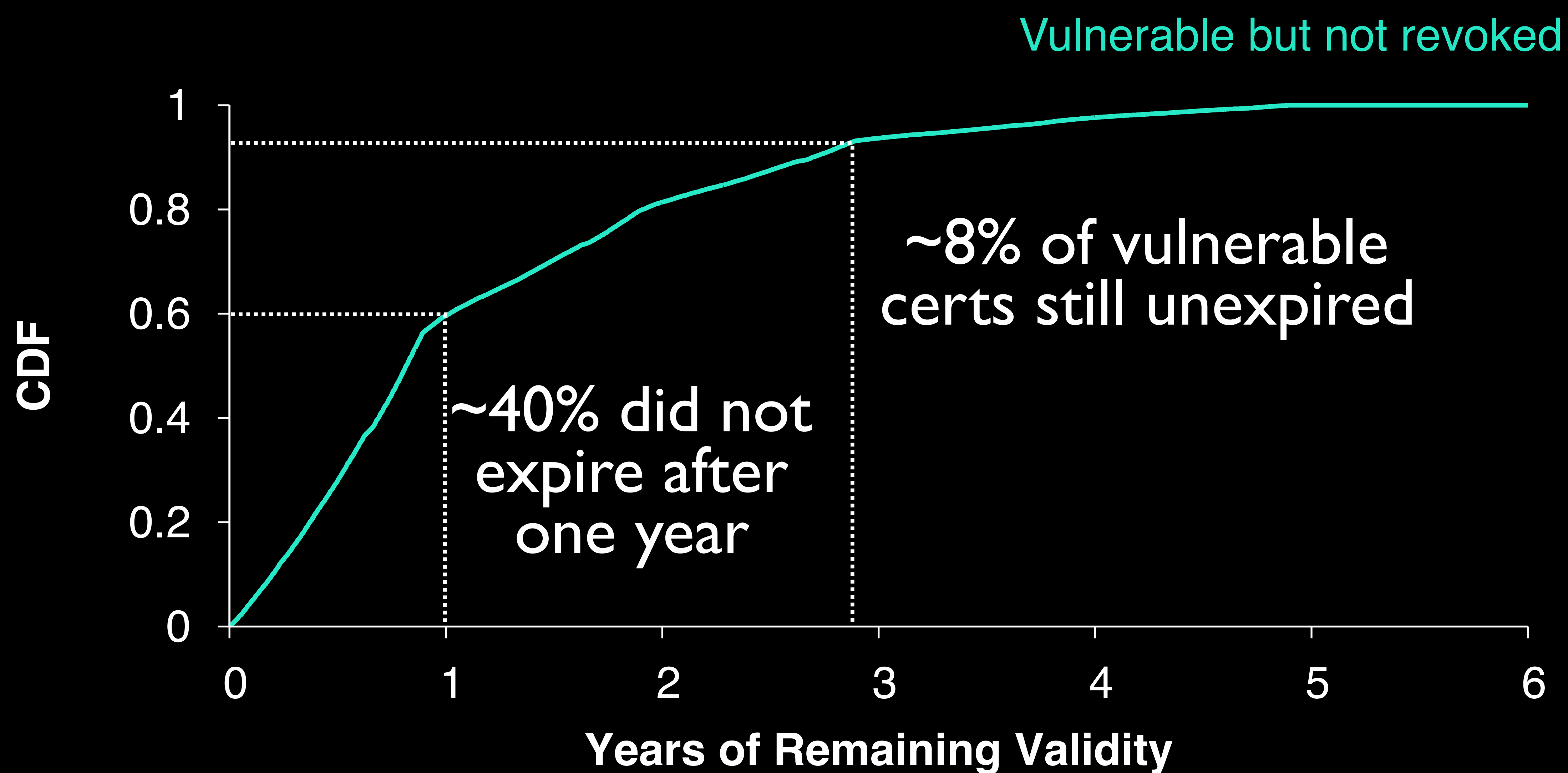
Can we wait for expiration?



Can we wait for expiration?

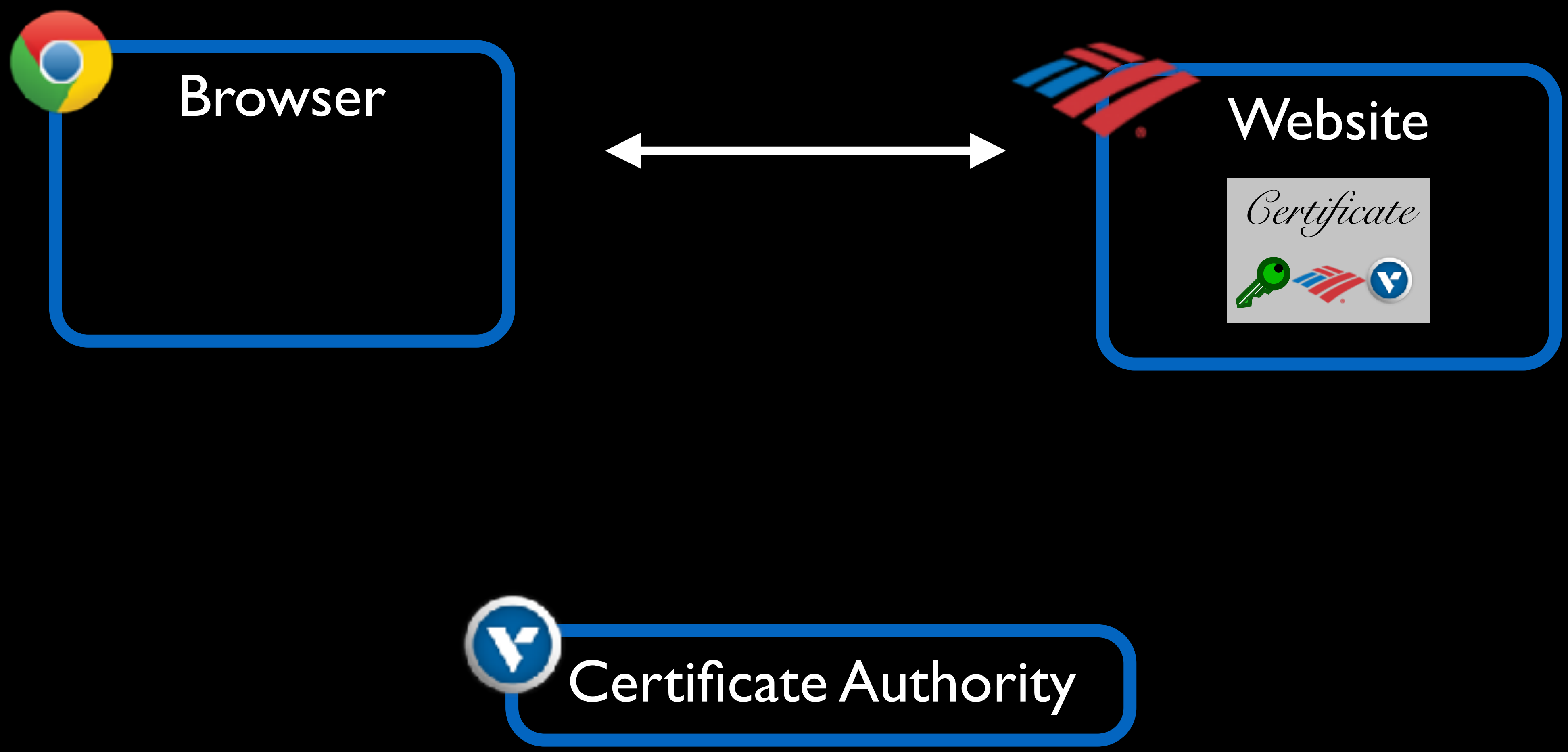


Can we wait for expiration?

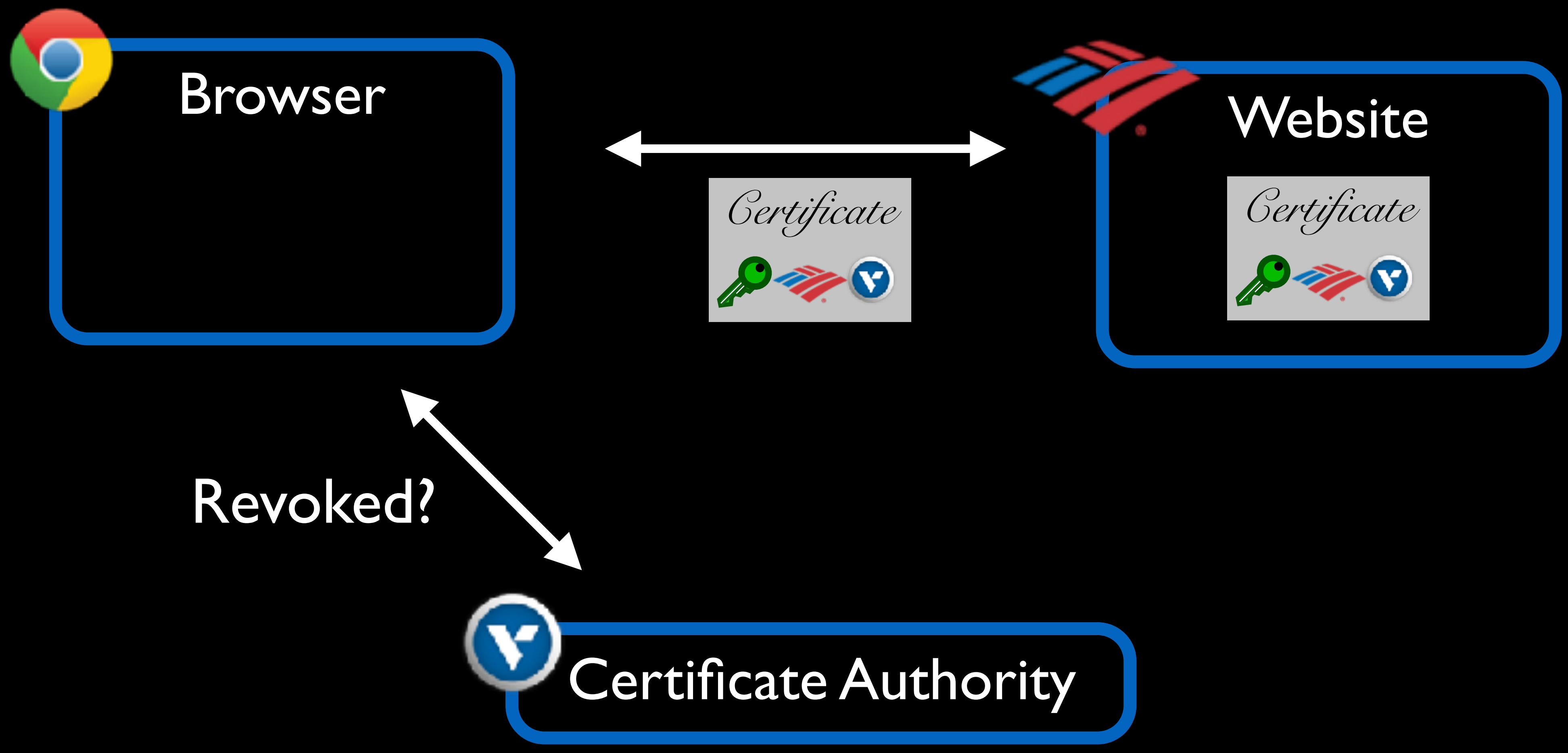


We may be dealing with Heartbleed for *years*

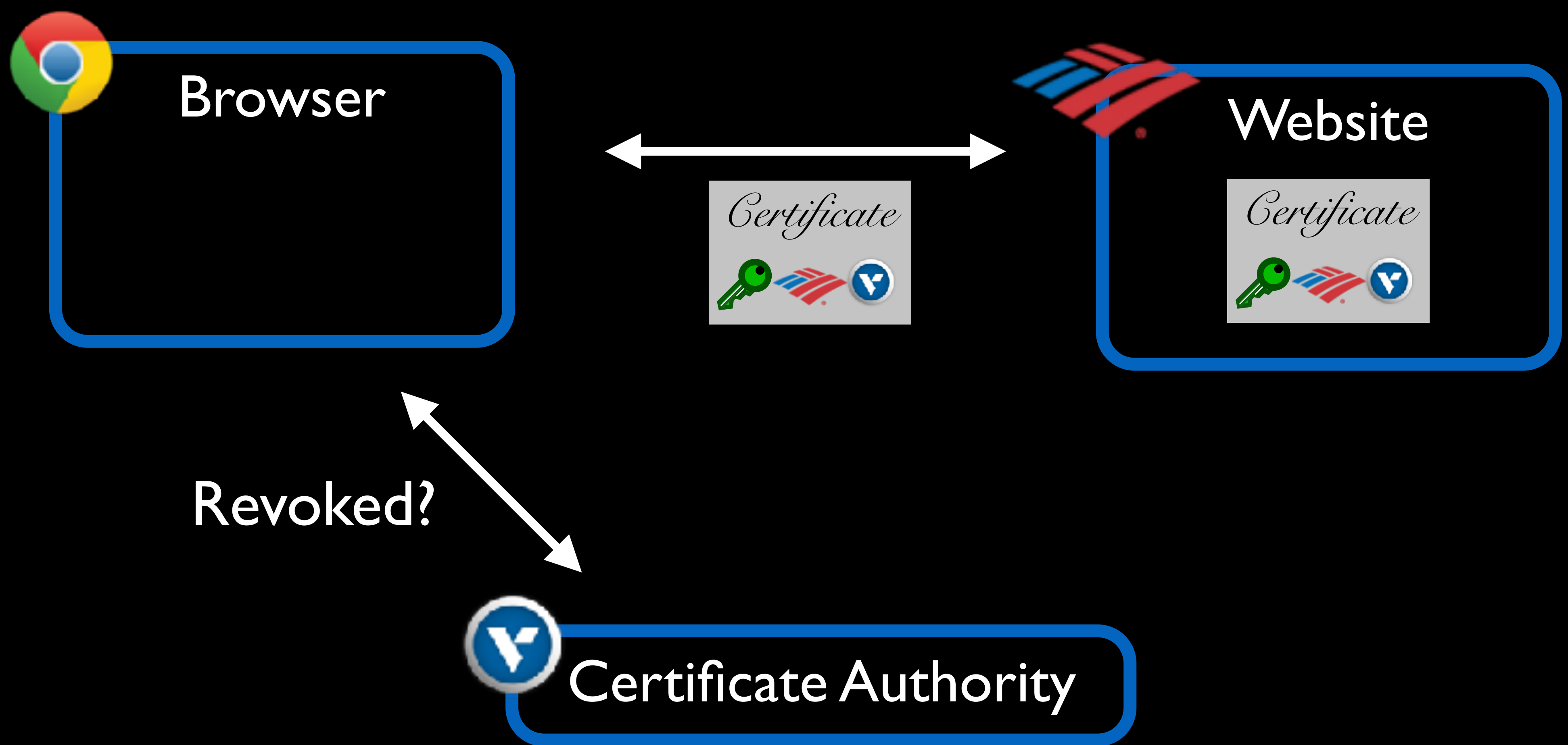
Security is an economic concern



Security is an economic concern



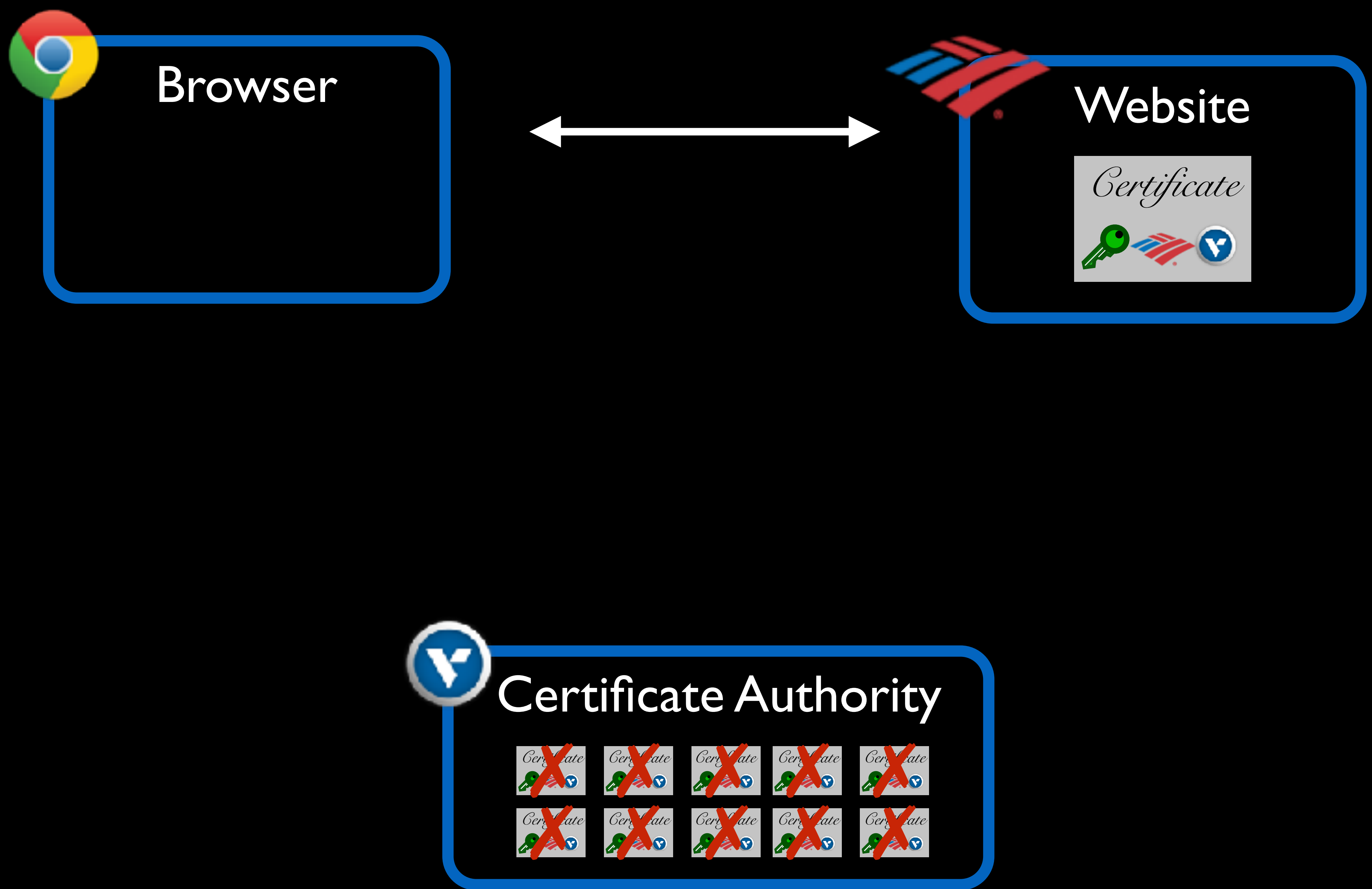
Security is an economic concern



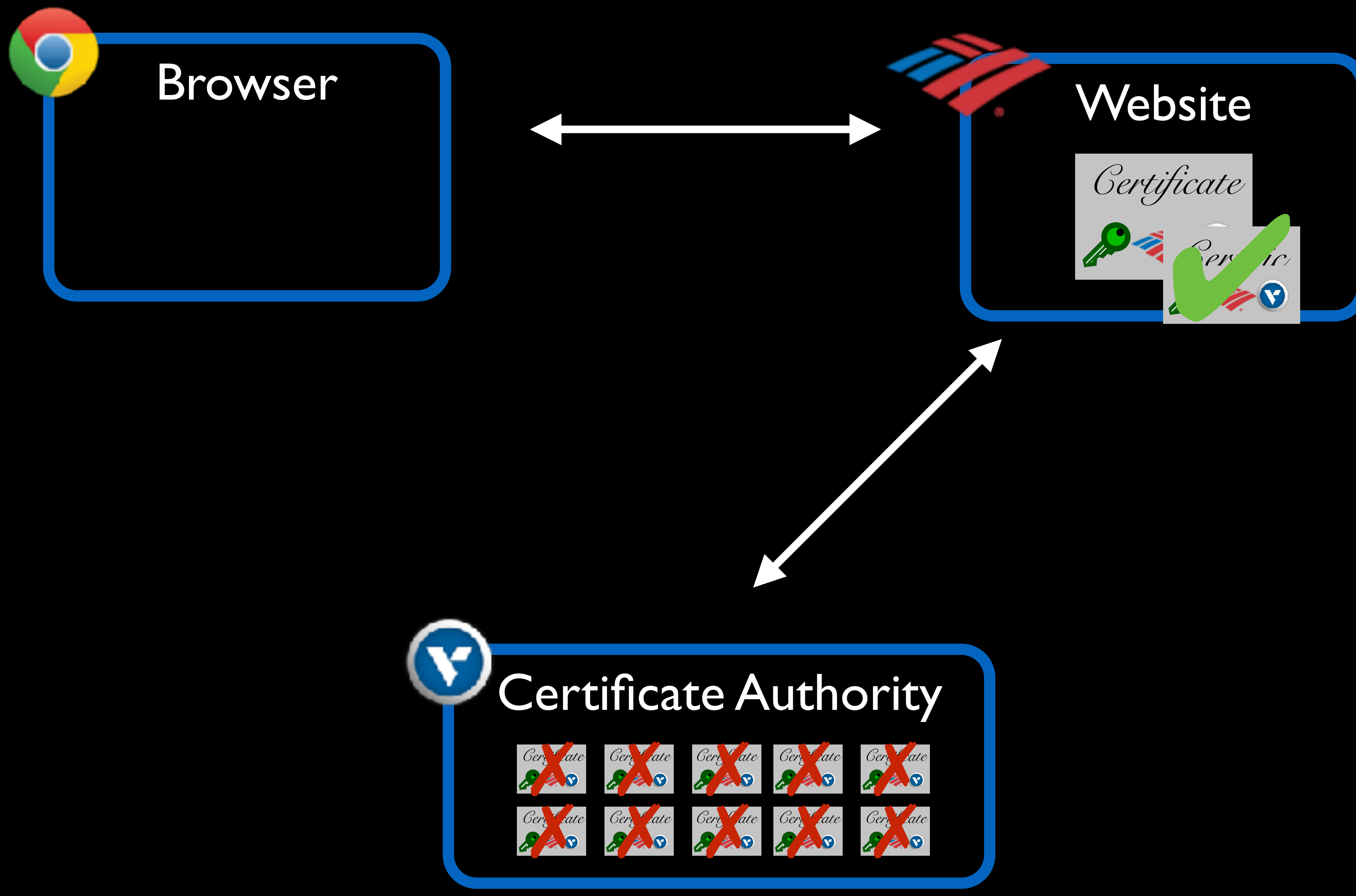
Browsers face tension between security and page load times

CAs face tension between security and bandwidth costs

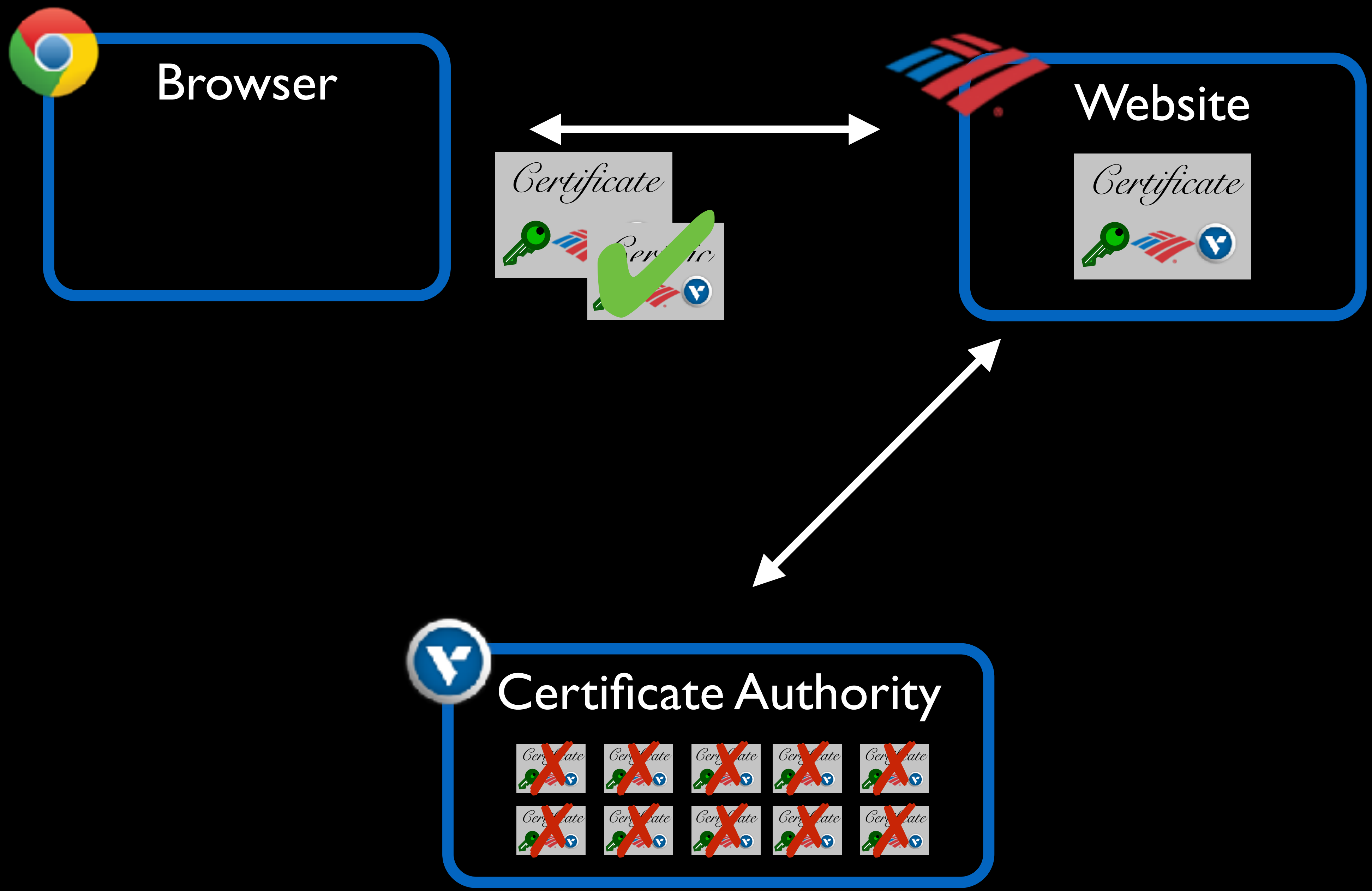
OCSP Stapling



OCSP Stapling



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

Testing browser behavior

Revocation protocols

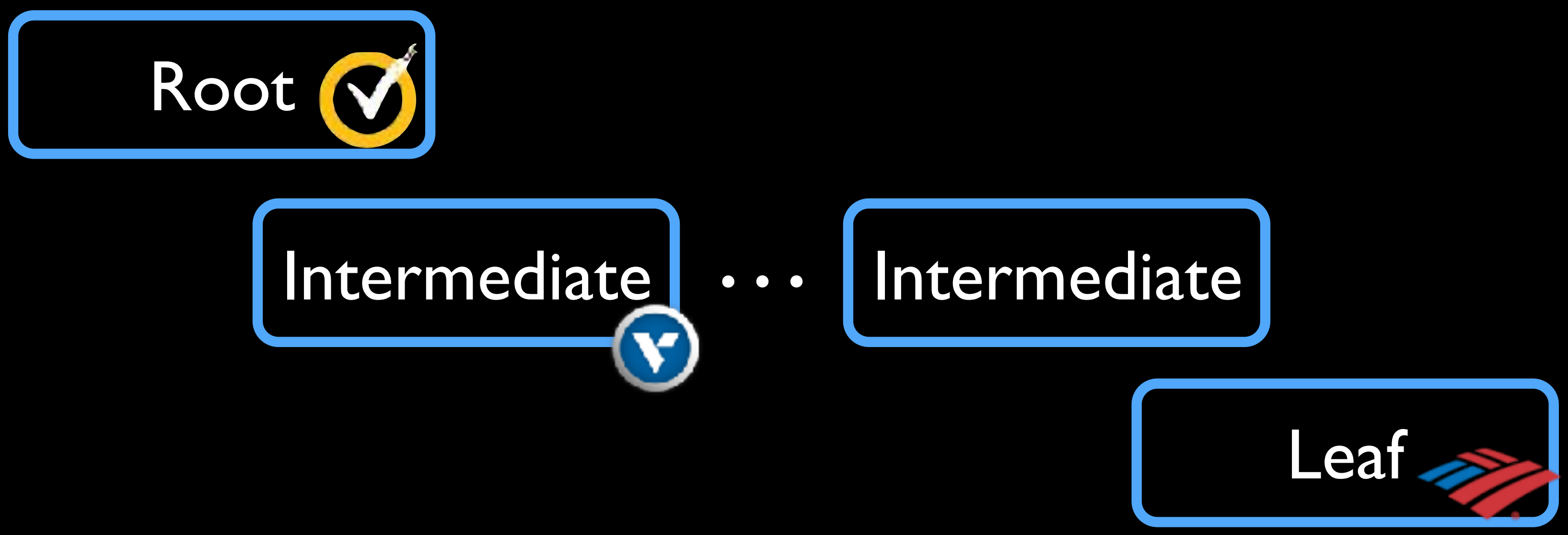
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

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



Testing browser behavior

Revocation protocols

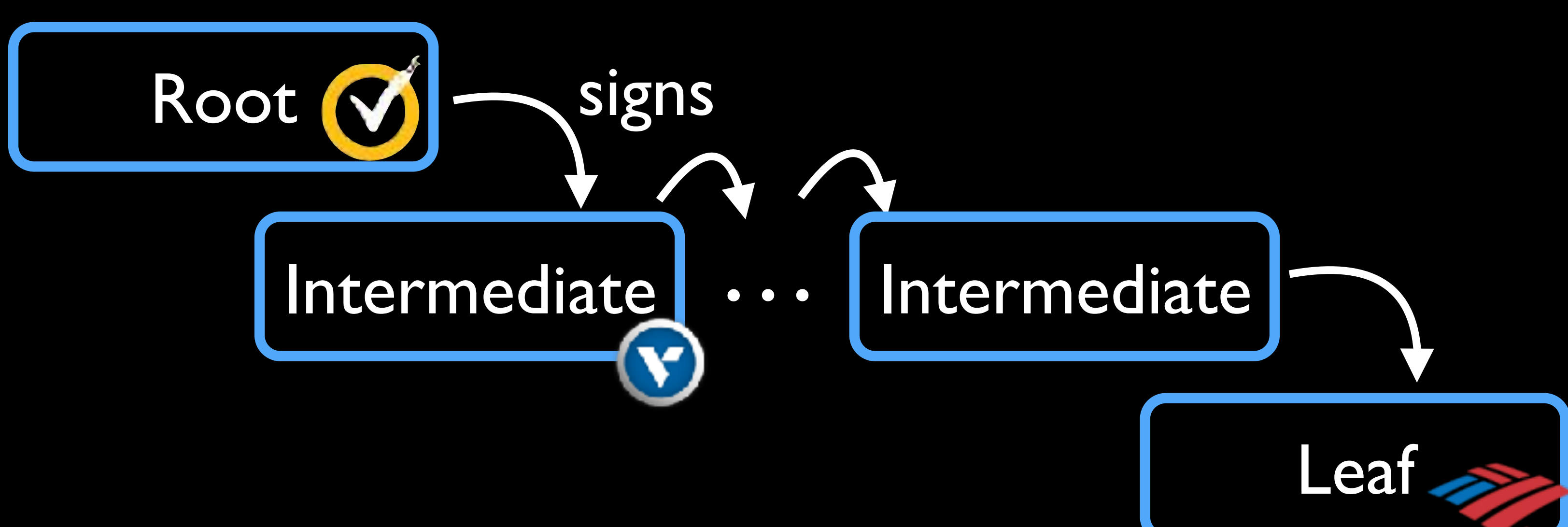
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

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



Testing browser behavior

Revocation protocols

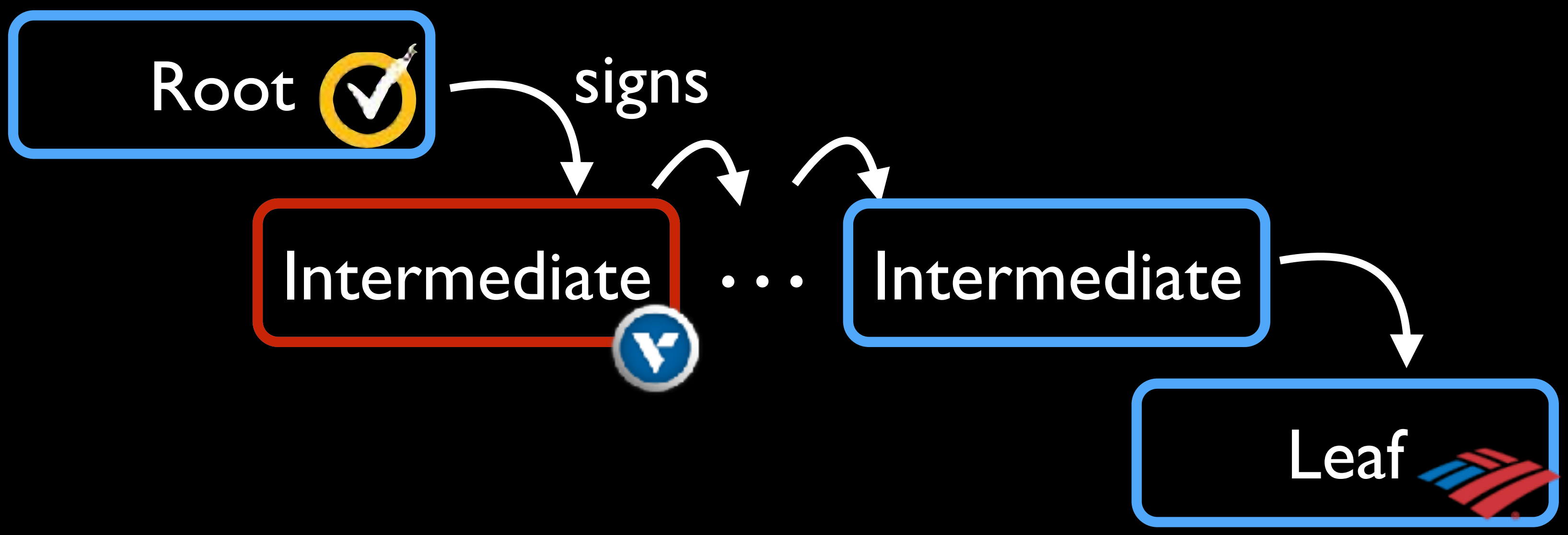
- Browsers *should* support all major protocols
    - CRLs, OCSP, OCSP stapling

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



Testing browser behavior

Revocation protocols

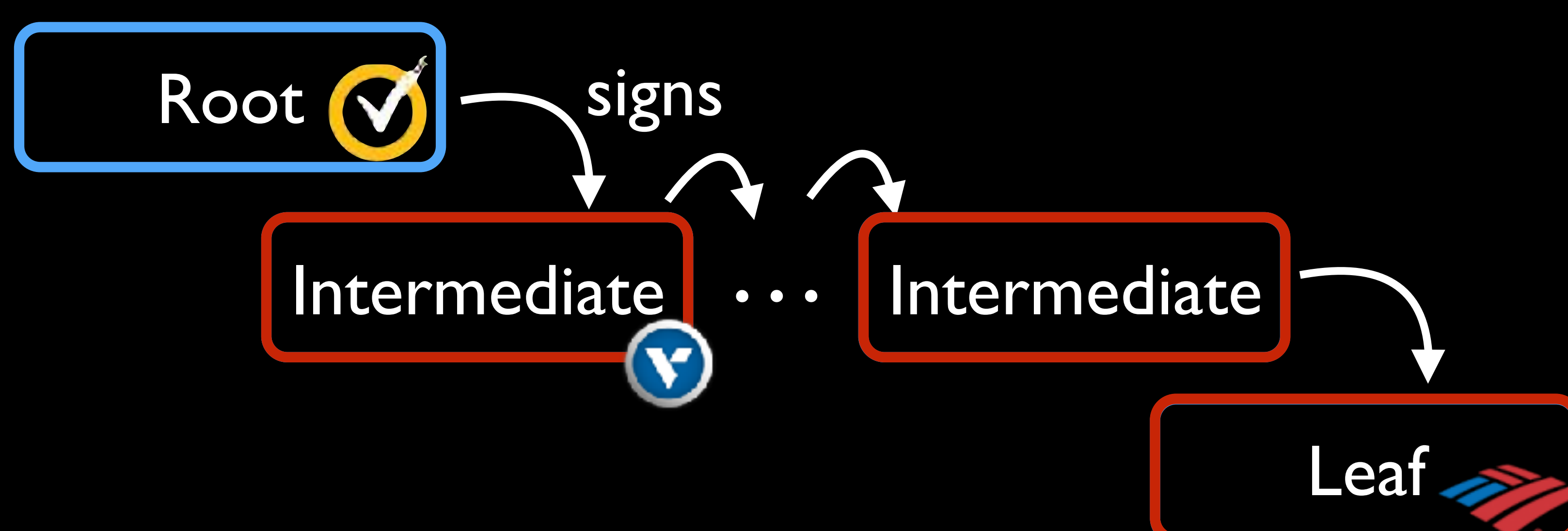
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

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

Chain lengths

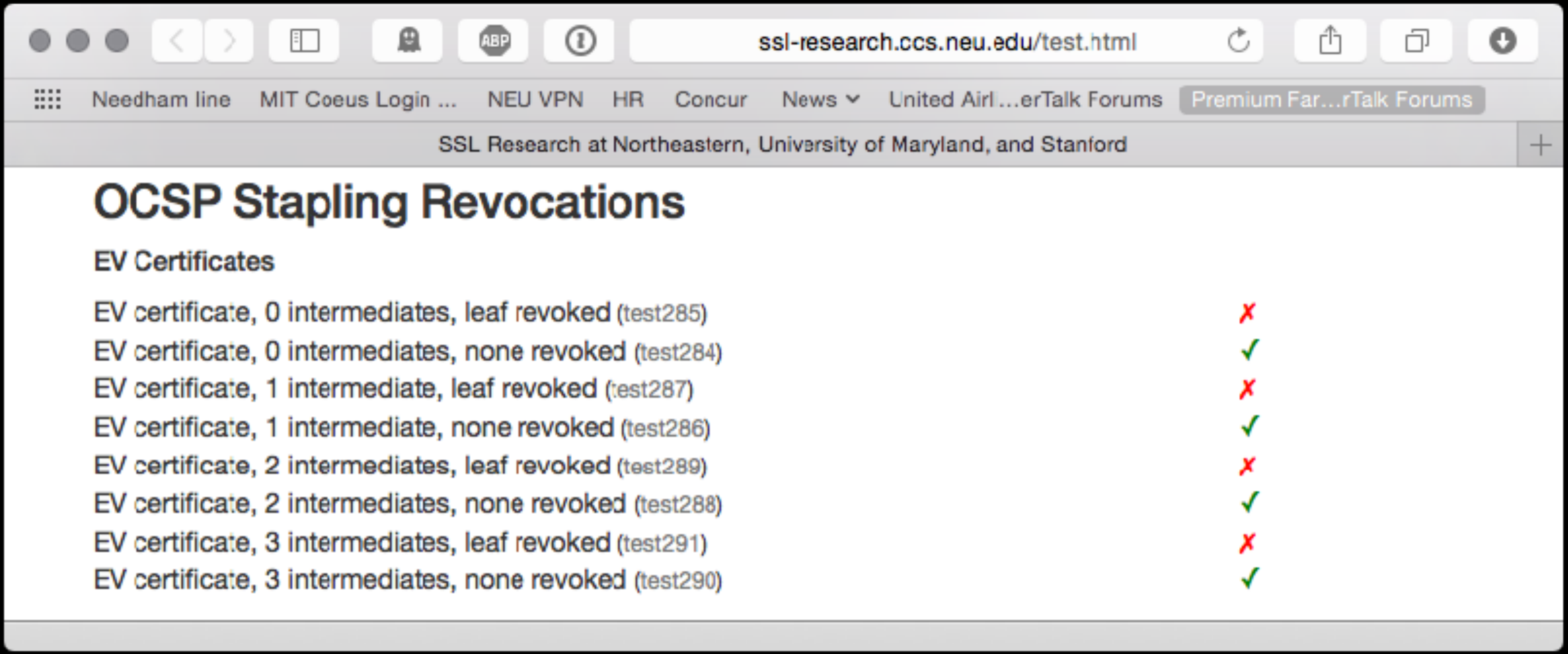
- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root



Test harness

Implemented 192 tests using fake root certificate + Javascript

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



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													
	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														

✔ 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

		Chrome 42 OS X	Chrome 42 Win.	Chrome 42 Linux
CRL				
Int. 1	Revoked	EV	✓	EV
	Unavailable	EV	✓	–
Int. 2+	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Leaf	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
OCSP				
Int. 1	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Int. 2+	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Leaf	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
OCSP Stapling				
	Request OCSP Staple	✓	✓	✓
	Respect Revoked Staple	✗	✓	–

Chrome

Generally, only checks for EV certs
~3% of all certs

Allows if revocation info unavailable

Supports OCSP stapling

✓ 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

		Deskt Firefox 35–37
	CRL	
Int. 1	Revoked	✗
	Unavailable	✗
Int. 2+	Revoked	✗
	Unavailable	✗
Leaf	Revoked	✗
	Unavailable	✗
	OCSP	
Int. 1	Revoked	EV
	Unavailable	✗
Int. 2+	Revoked	EV
	Unavailable	✗
Leaf	Revoked	✓
	Unavailable	✗
OCSP Stapling		
Request OCSP Staple		✓
Respect Revoked Staple		✓

Firefox

Never checks CRLs

Only checks intermediates for EV certs

Allows if revocation info unavailable

Supports OCSP stapling

✔ 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

		Safari 6–8
	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	–

Safari

Checks CRLs and OCSP

Allows if revocation info unavailable

Except for first intermediate, for CRLs

Does *not* support OCSP stapling

✓ 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

		IE 7–9	IE 10–11
CRL			
Int. 1	Revoked	✓	✓
	Unavailable	✓	✓
Int. 2+	Revoked	✓	✓
	Unavailable	✗	✗
Leaf	Revoked	✓	✓
	Unavailable	✗	A
OCSP			
Int. 1	Revoked	✓	✓
	Unavailable	✓	✓
Int. 2+	Revoked	✓	✓
	Unavailable	✗	✗
Leaf	Revoked	✓	✓
	Unavailable	✗	A
OCSP Stapling			
Request OCSP Staple		✓	✓
Respect Revoked Staple		✓	✓

Internet Explorer

Checks CRLs *and* OCSP

Often rejects if revocation info unavailable
- Pops up alert for leaf in IE 10+

Supports OCSP stapling

✓ 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

		Mobile Browsers			
		iOS 6–8	Andr. 4.1–5.1 Stock	Andr. 4.1–5.1 Chrome	IE 8.0
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		✗	I	I	✗
Respect Revoked Staple		–	–	–	–

Mobile Browsers

Uniformly *never* check

Android browsers request Staple
…and promptly ignore it

✔ 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

		Chrome 42 OS X	Chrome 42 Win.	Chrome 42 Linux	Firefox 35–37	Opera 12.17	Opera 28.0	Safari 6–8	IE 7–9	IE 10–11	iOS 6–8	Andr. 4.1–5.1 Stock	Andr. 4.1–5.1 Chrome	IE 8.0
		Desktop Browsers									**Mobile Browsers**			
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	–	✓	✓	–	–	–	–

✓ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

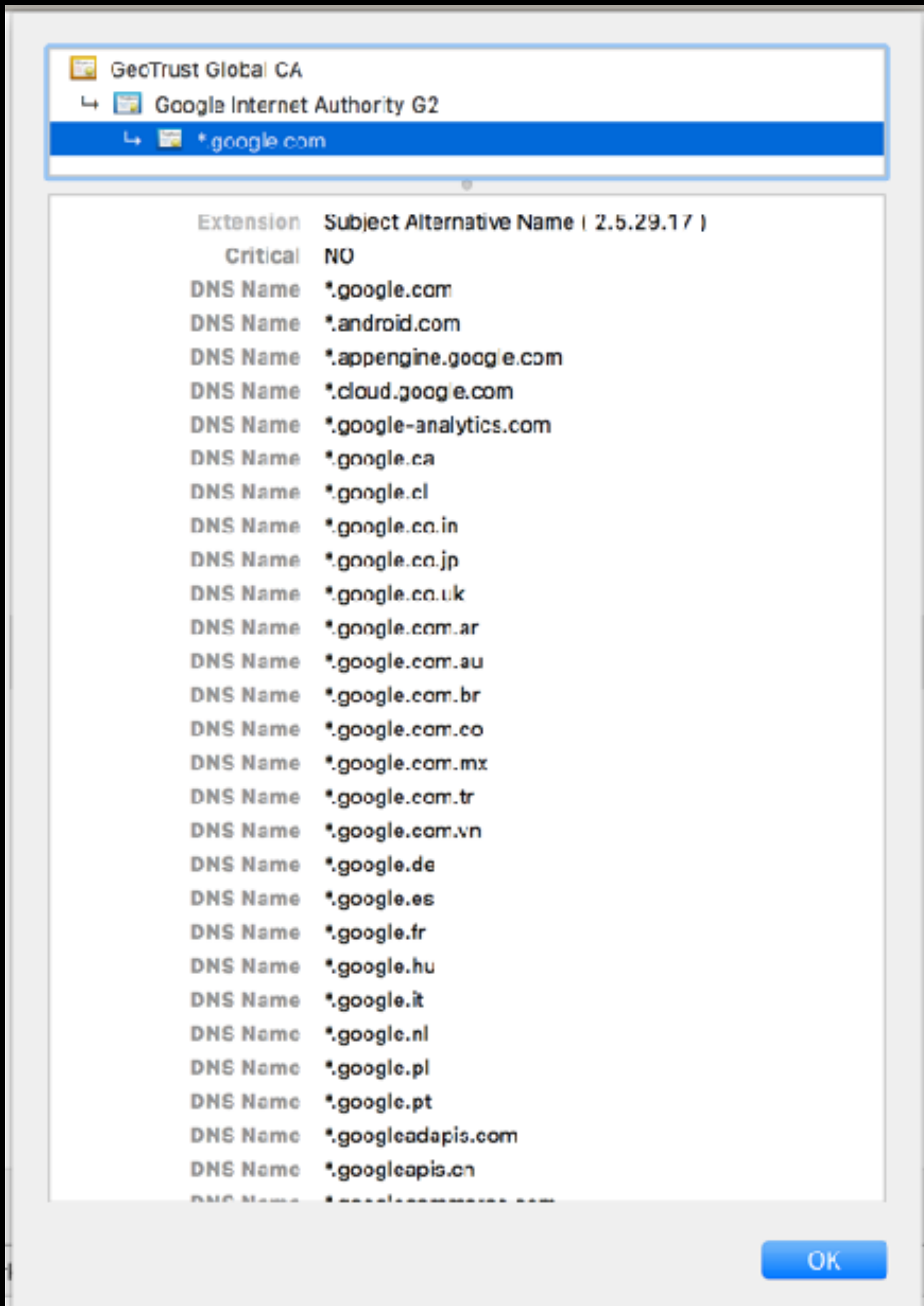
A Pops up alert to user
L/W Passes on Linux/Win.

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	–	✓	✓	–	–	–	–

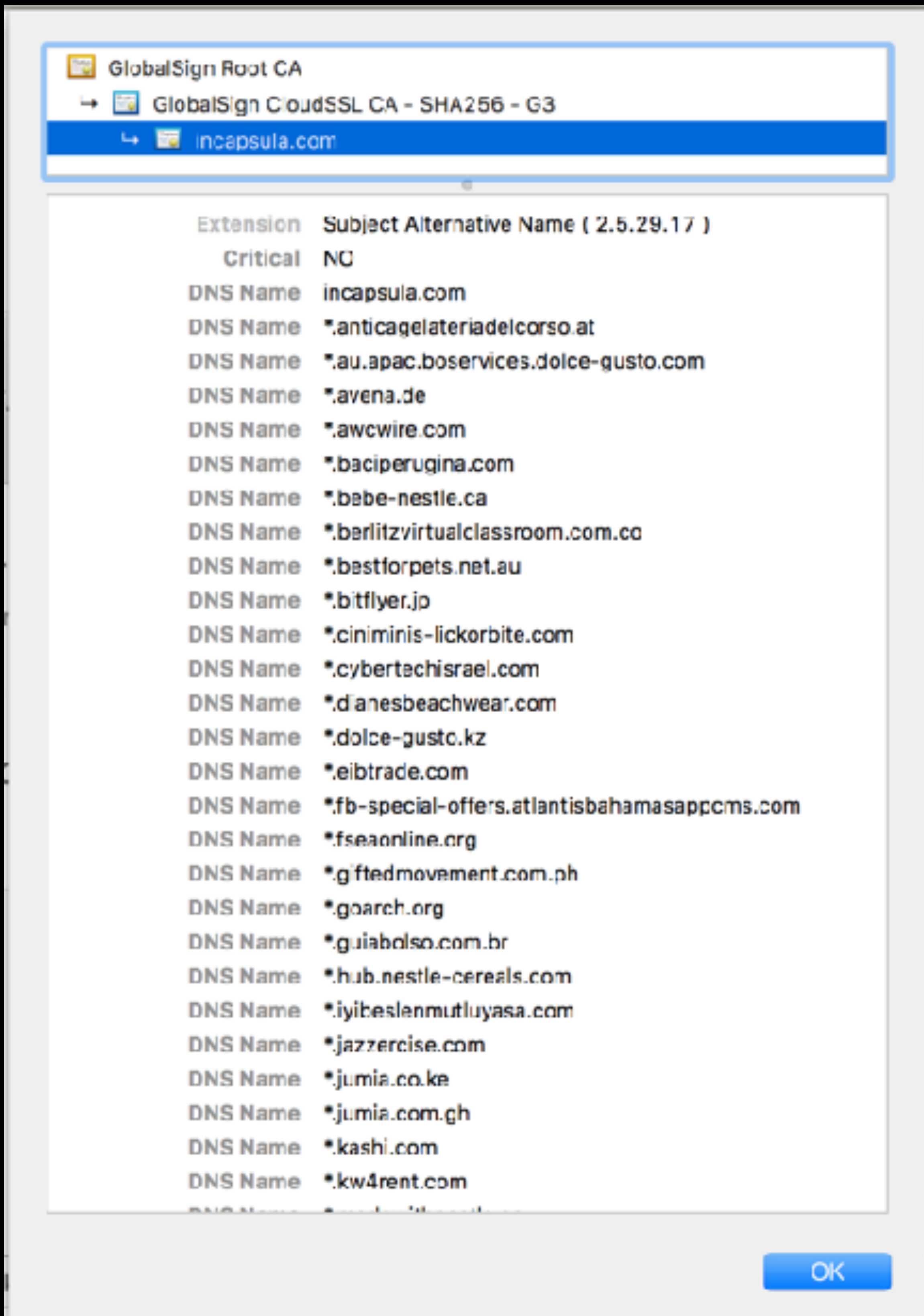
Browser developers are not
doing what the PKI needs them to do

Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

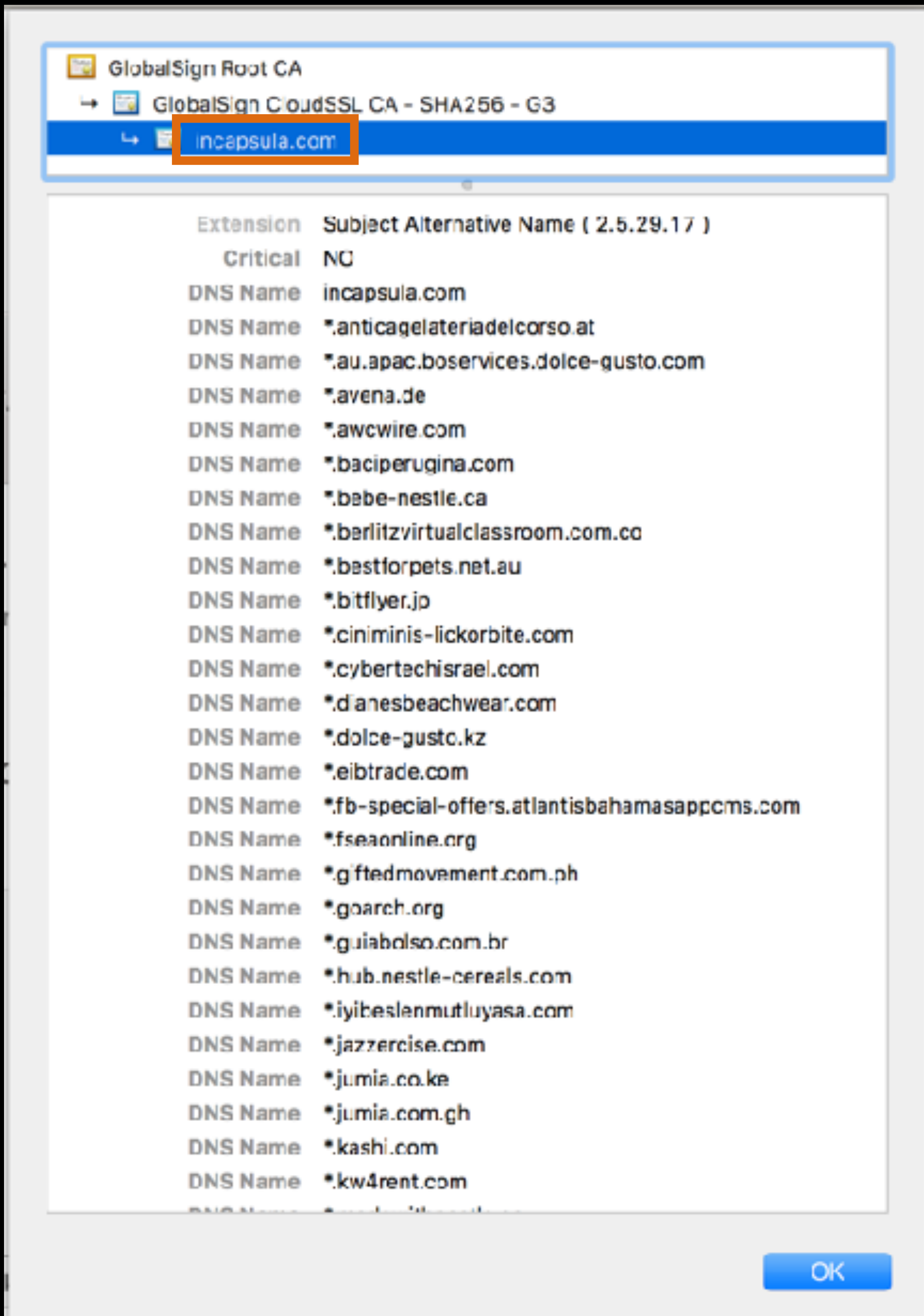
Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

Practice: *Different organizations* lumped together

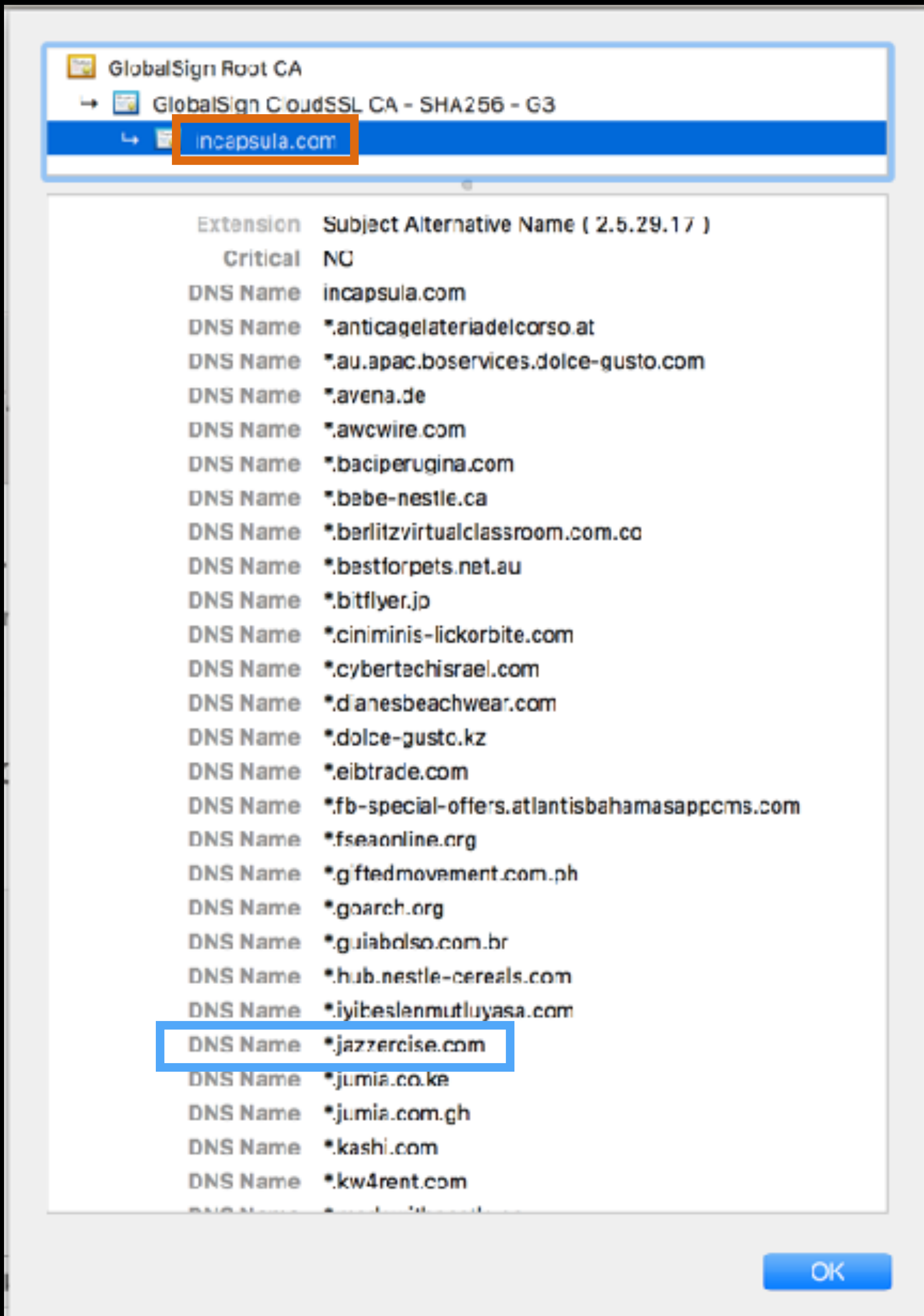
Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

Practice: *Different organizations* lumped together

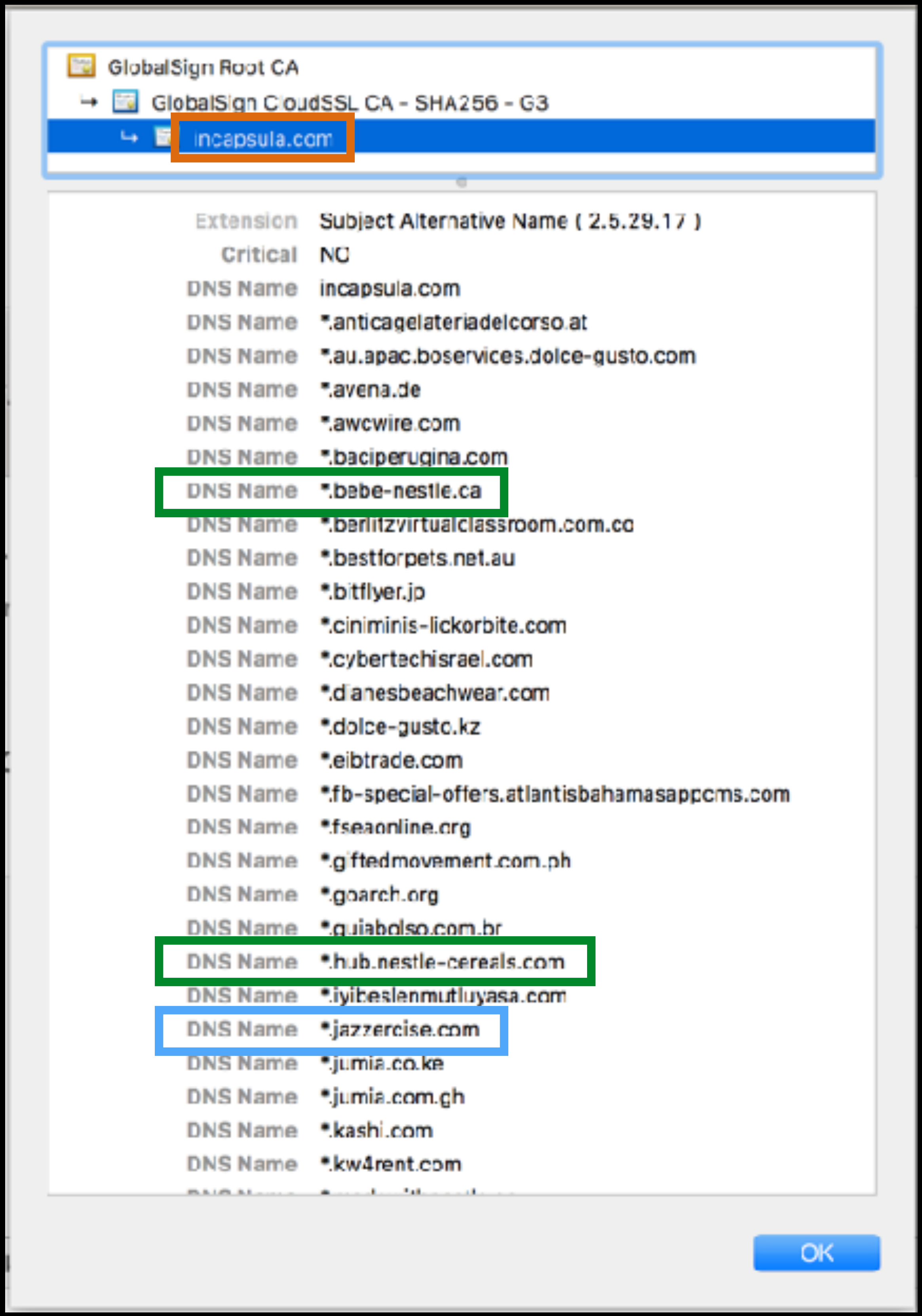
Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

Practice: *Different organizations* lumped together

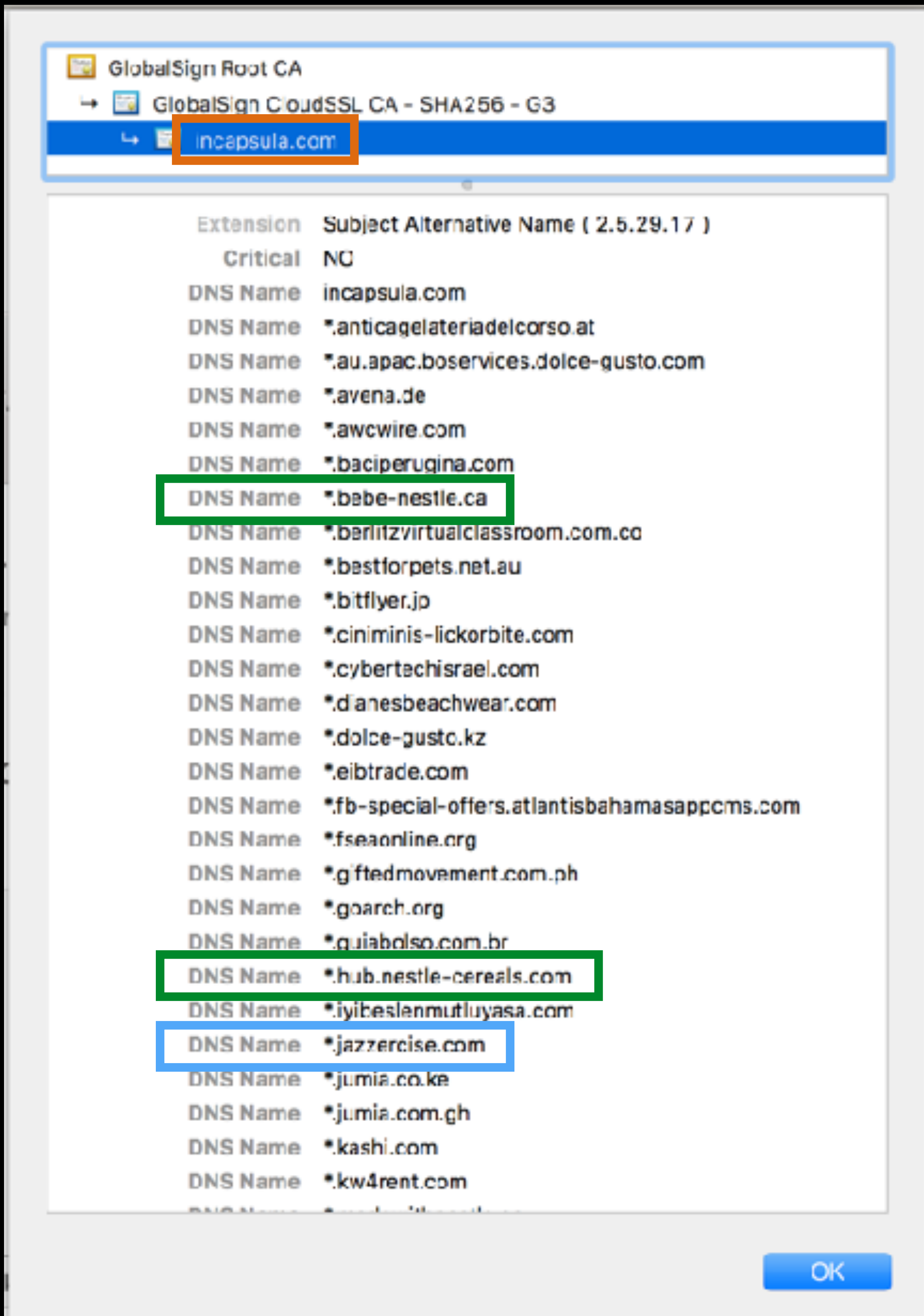
Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

Practice: *Different organizations* lumped together

Subject Alternate Name (SAN) Lists



Spirit: Multiple names for the *same organization*

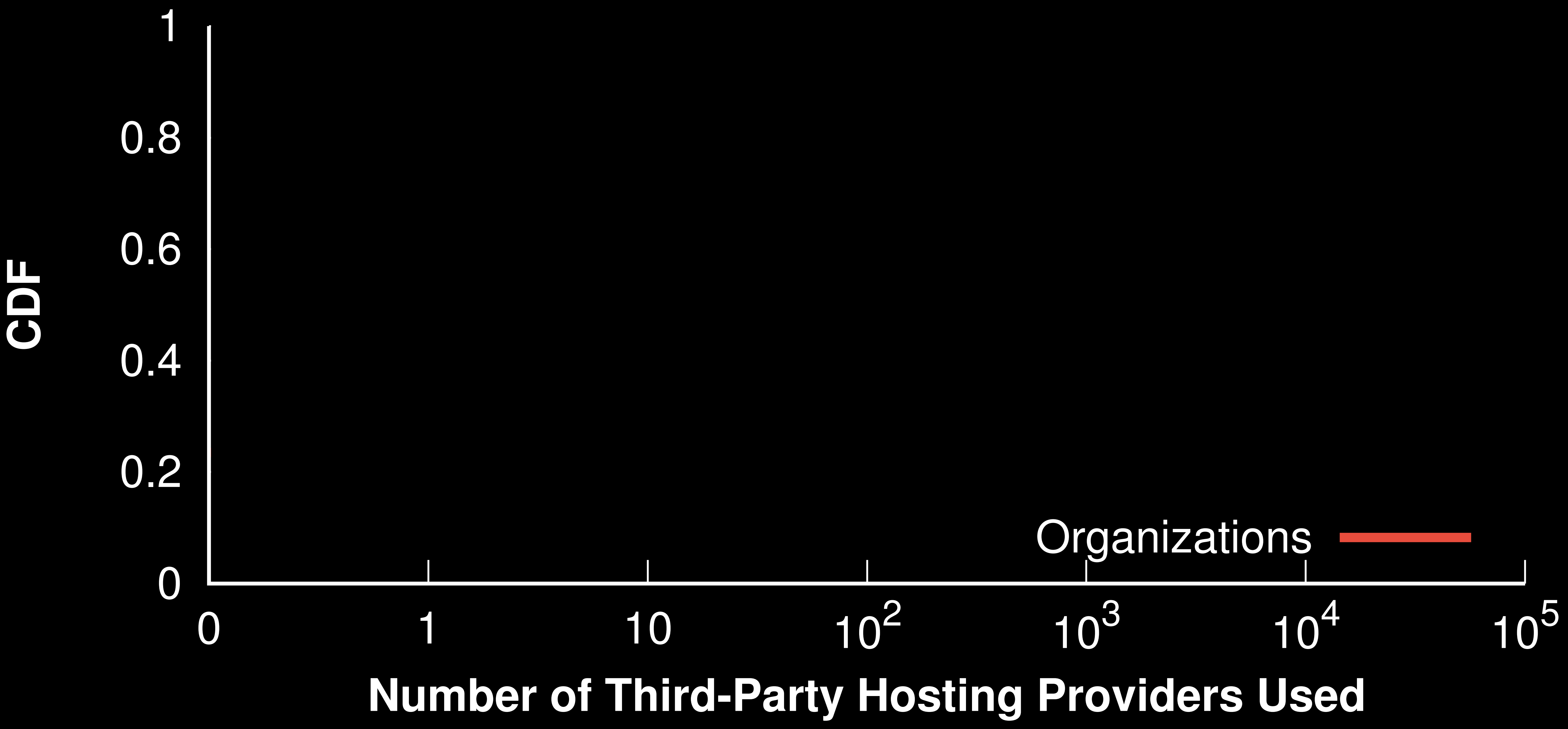
Practice: *Different organizations* lumped together

Cruise-liner Certificate

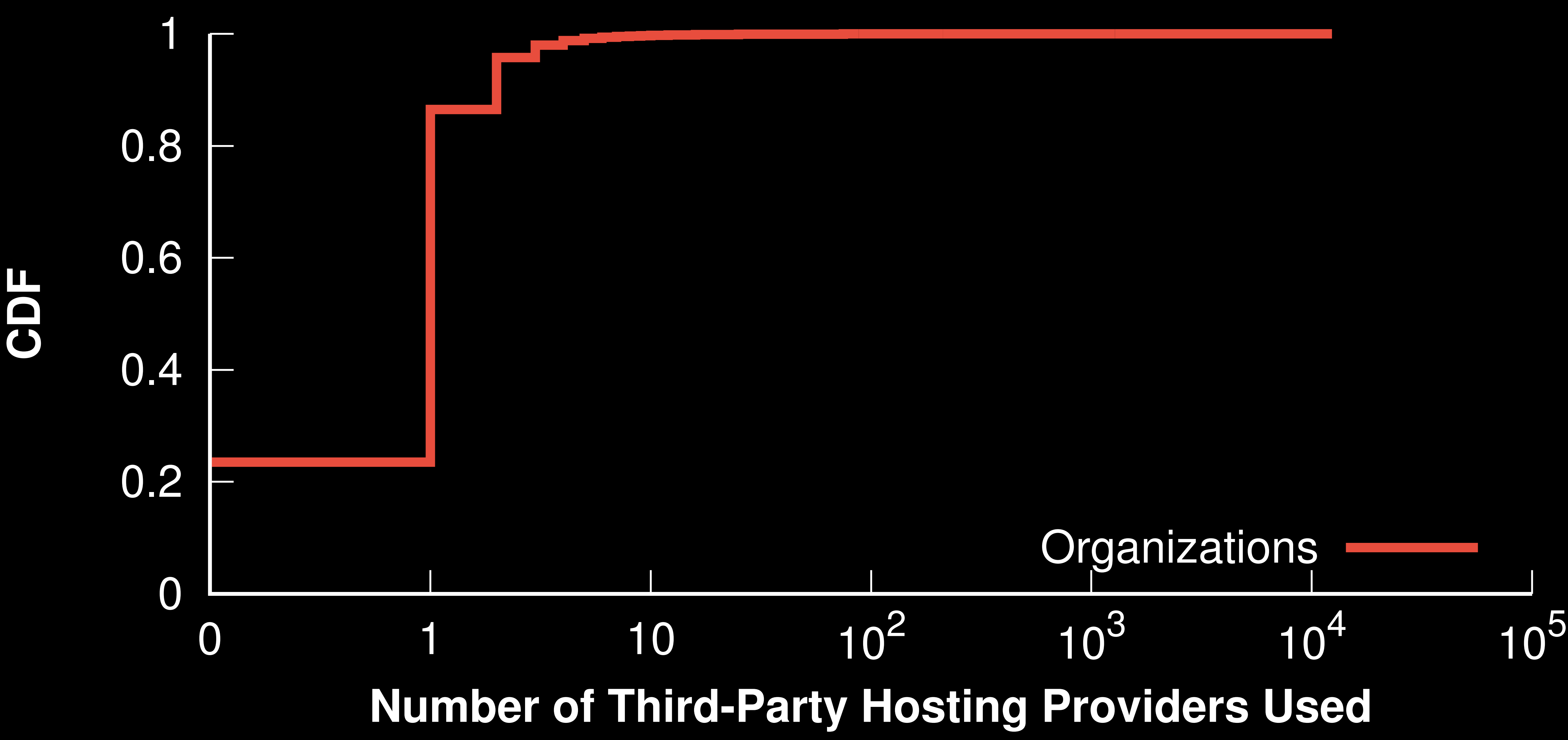
Who gets the private key?

Who manages it?

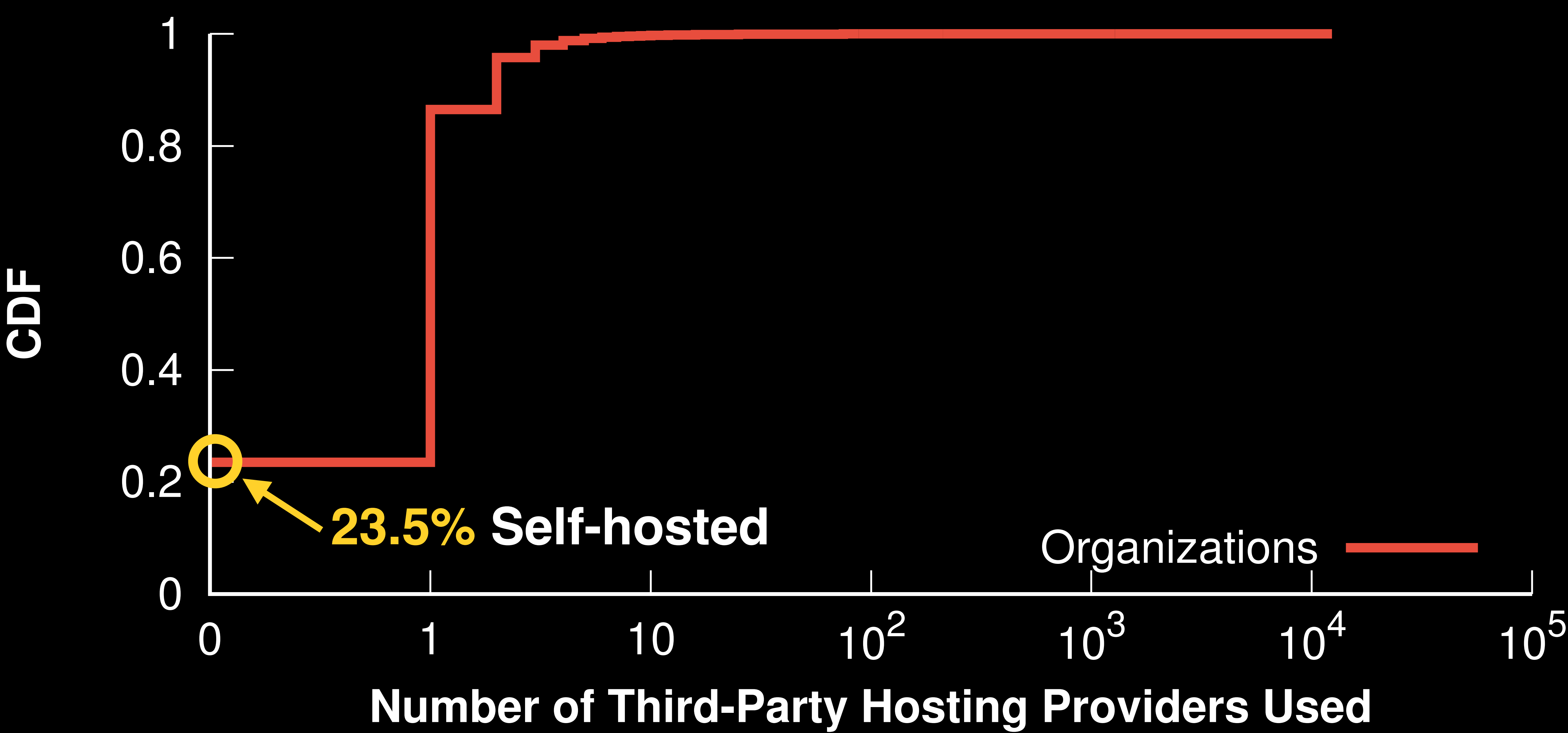
How prevalent is key sharing?



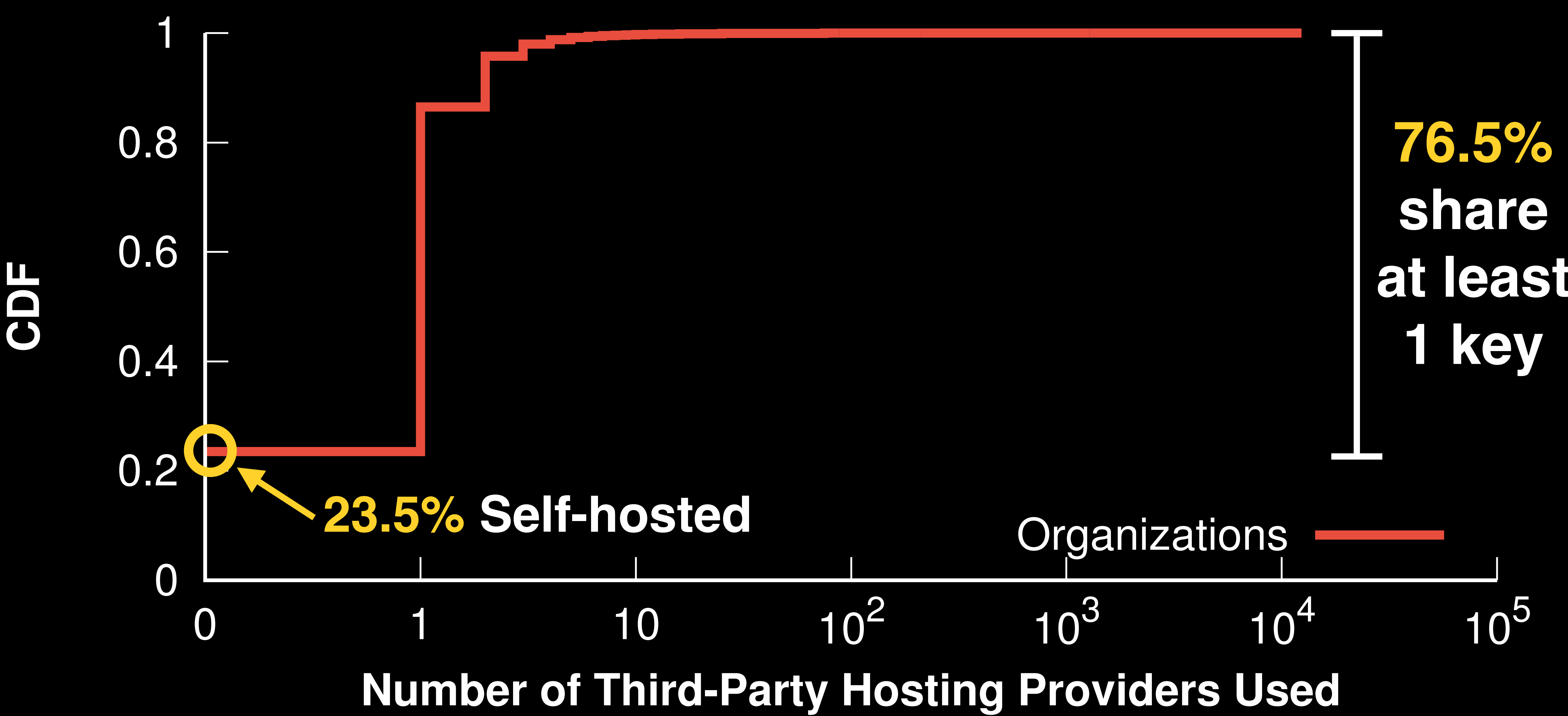
How prevalent is key sharing?



How prevalent is key sharing?

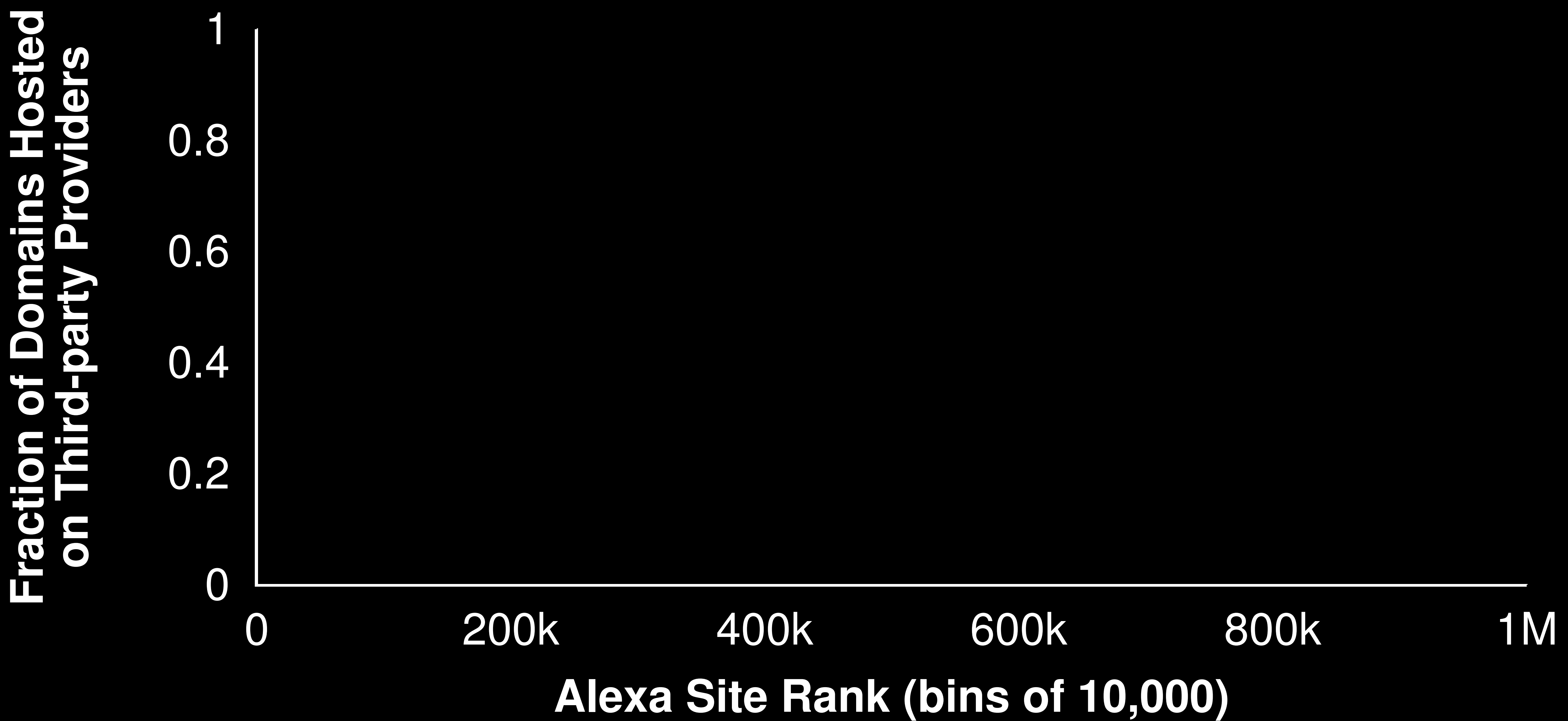


How prevalent is key sharing?

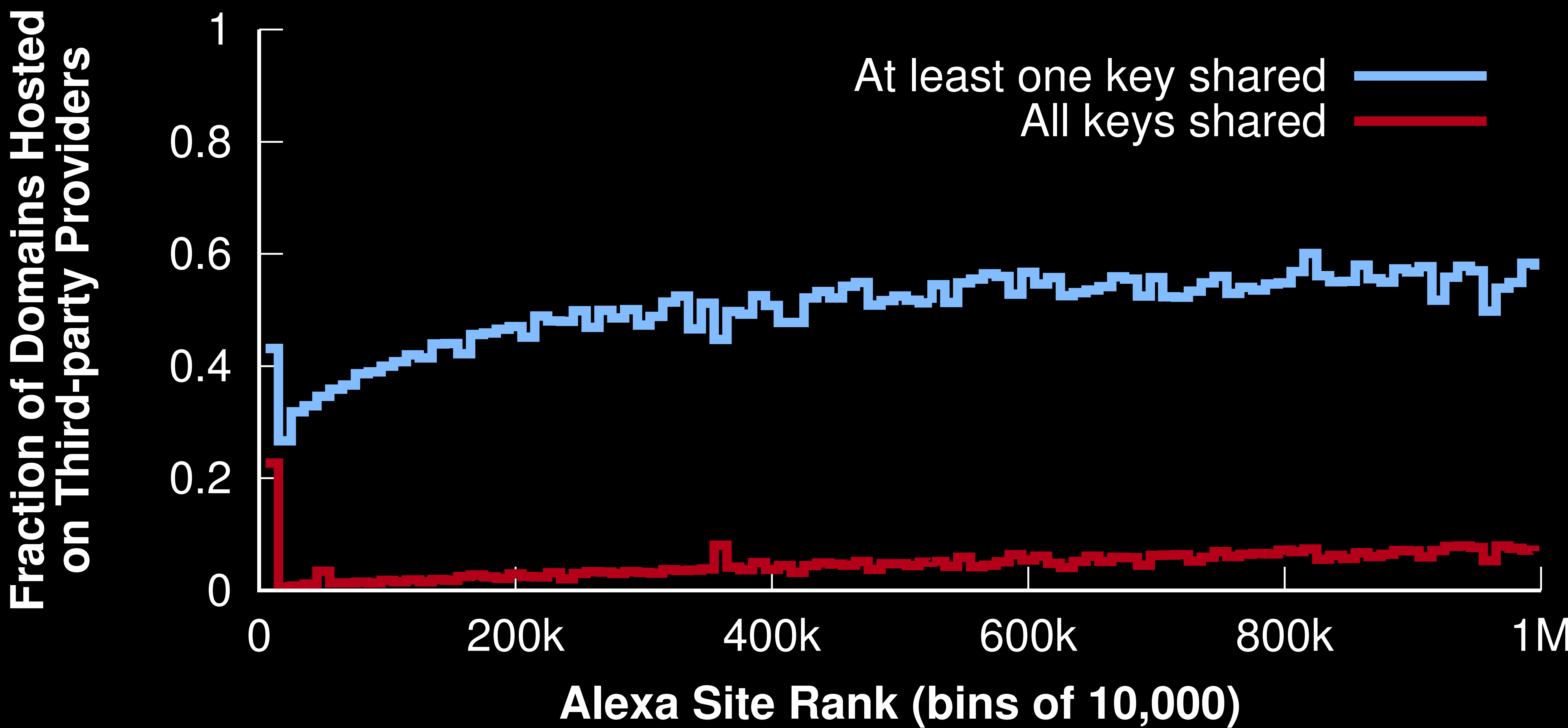


How prevalent is key sharing?

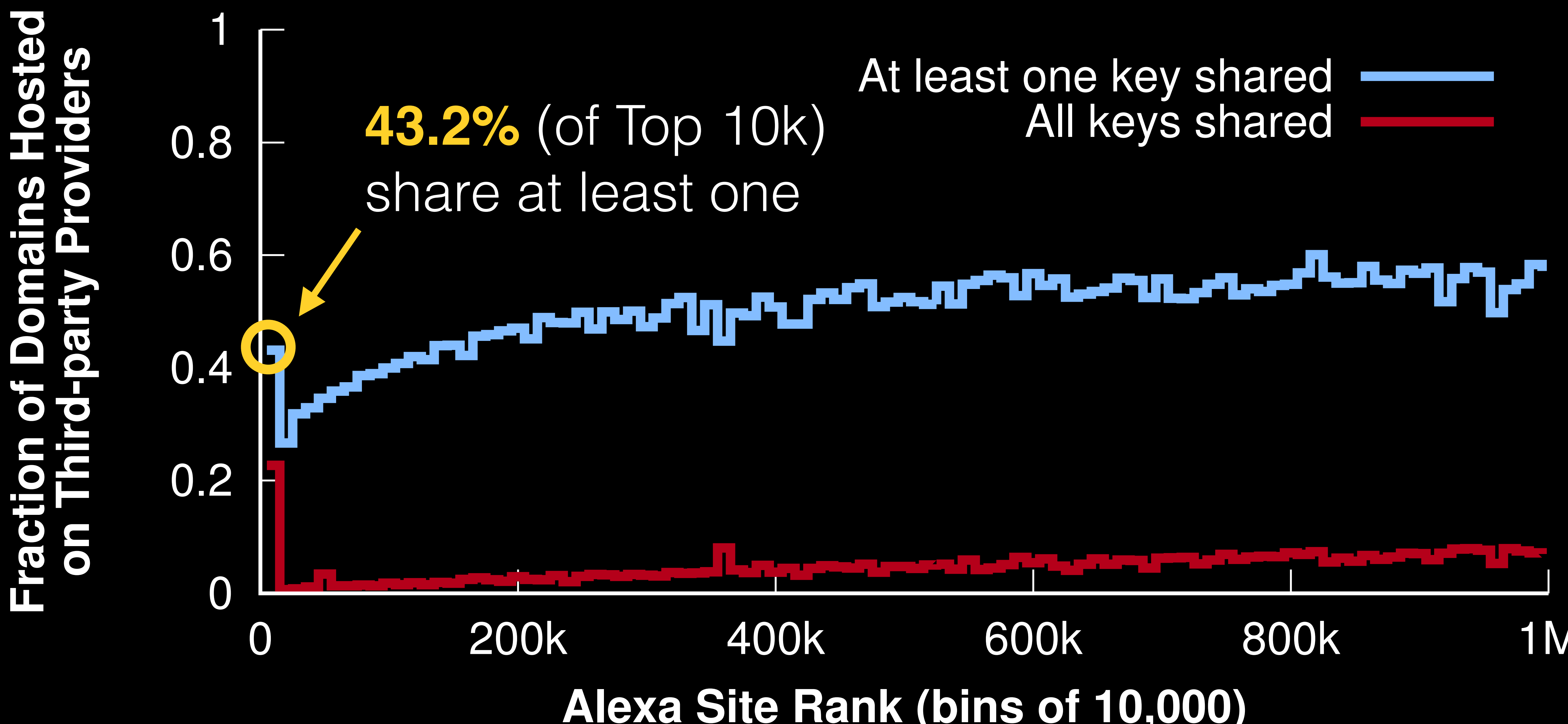
Who shares?


Fraction of Domains Hosted on Third-party Providers
1
0.8
0.6
0.4
0.2
0
0
200k
400k
600k
800k
1M
Alexa Site Rank (bins of 10,000)

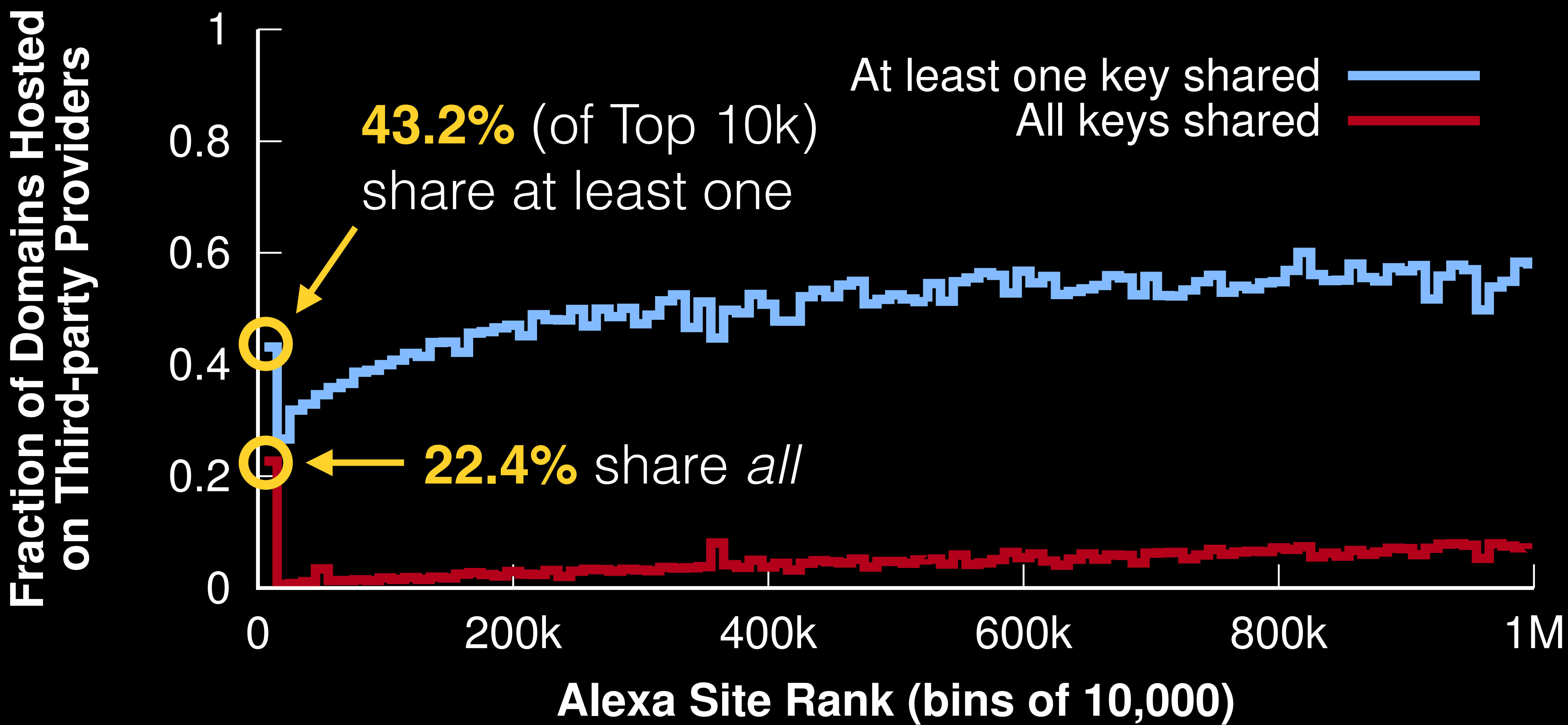
Who shares?


Fraction of Domains Hosted on Third-party Providers
1
0.8
0.6
0.4
0.2
0
At least one key shared
All keys shared
0
200k
400k
600k
800k
1M
Alexa Site Rank (bins of 10,000)

Who shares?

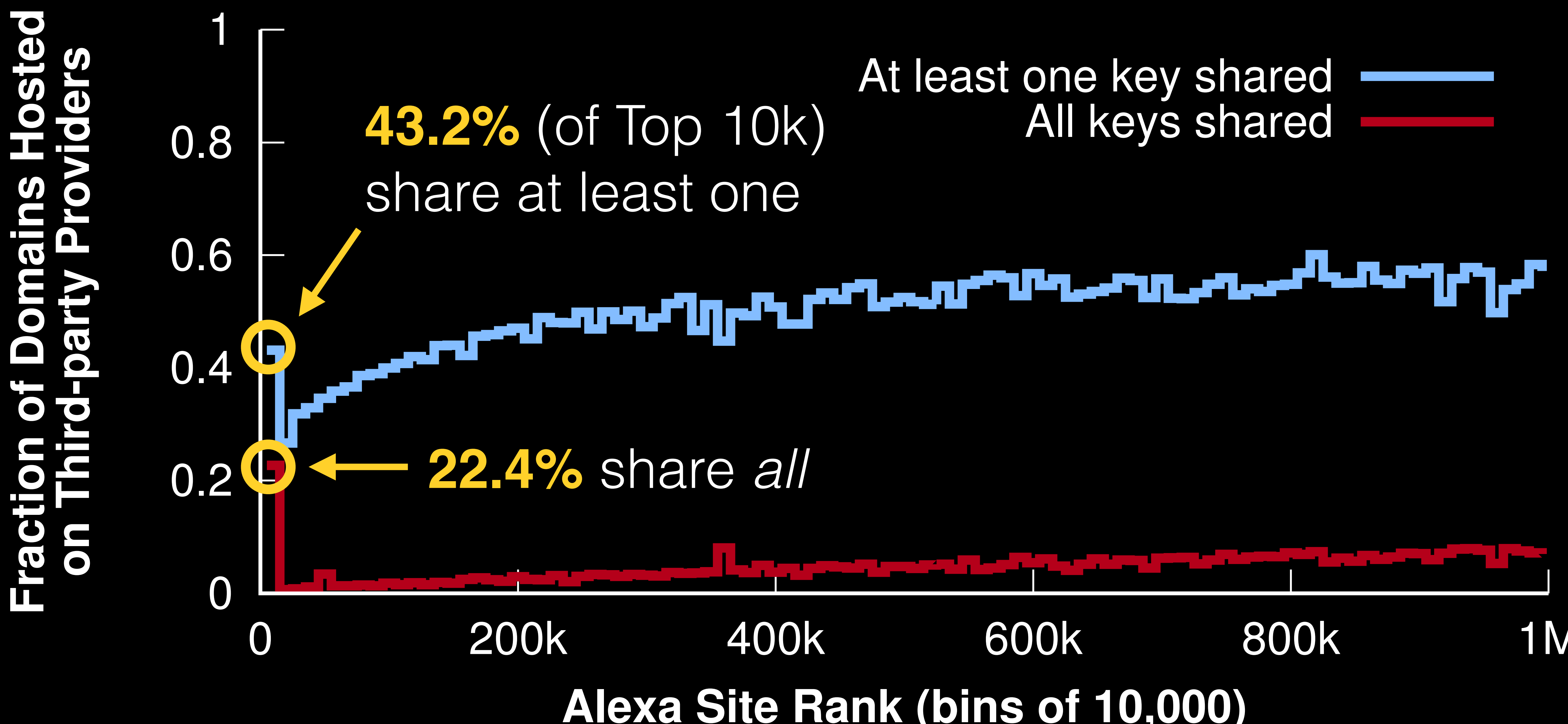

Fraction of Domains Hosted on Third-party Providers
1
0.8
0.6
0.4
0.2
0
0
200k
400k
600k
800k
1M
Alexa Site Rank (bins of 10,000)
At least one key shared
All keys shared
43.2% (of Top 10k) share at least one

Who shares?



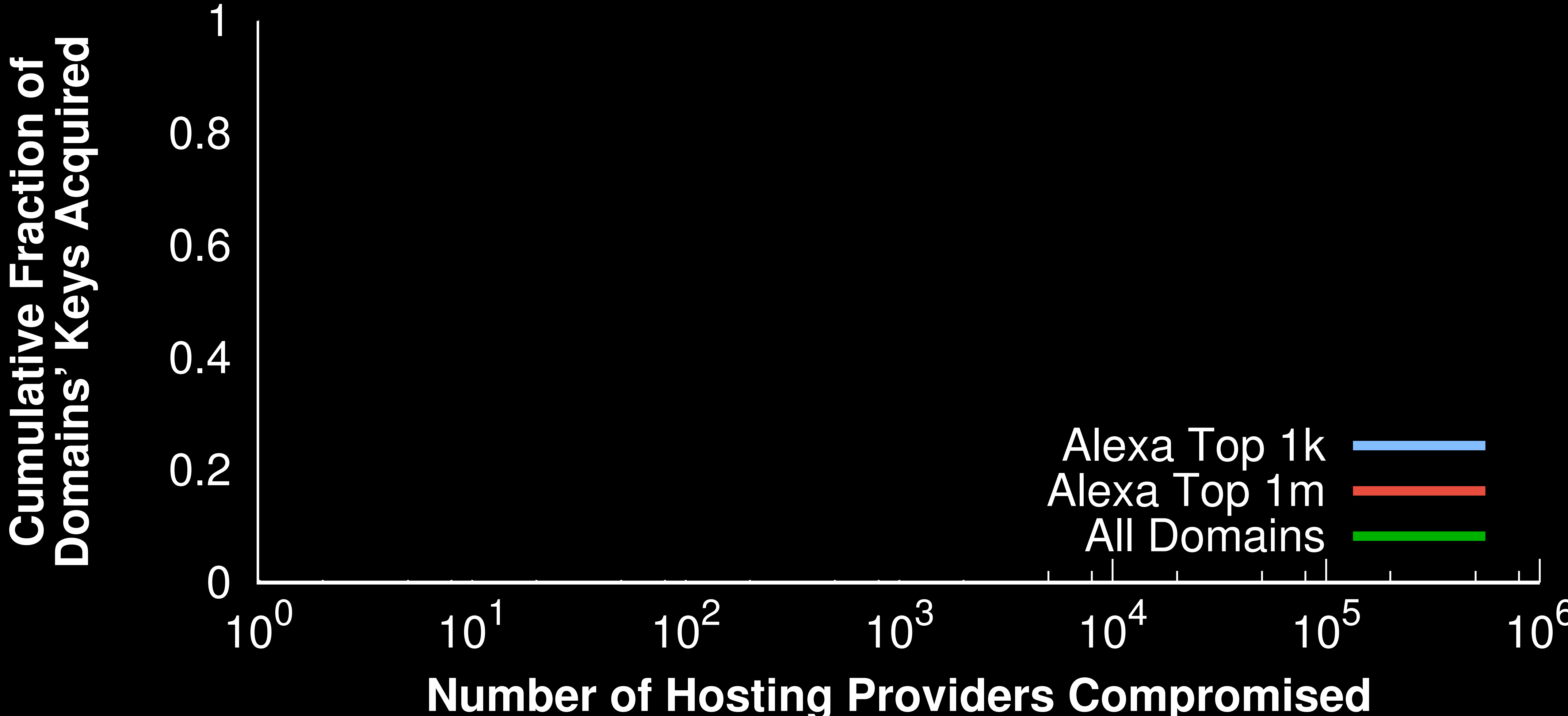
At least one key shared
All keys shared
43.2% (of Top 10k) share at least one
22.4% share all
Fraction of Domains Hosted on Third-party Providers
Alexa Site Rank (bins of 10,000)
0
0.2
0.4
0.6
0.8
1
0
200k
400k
600k
800k
1M

Who shares?

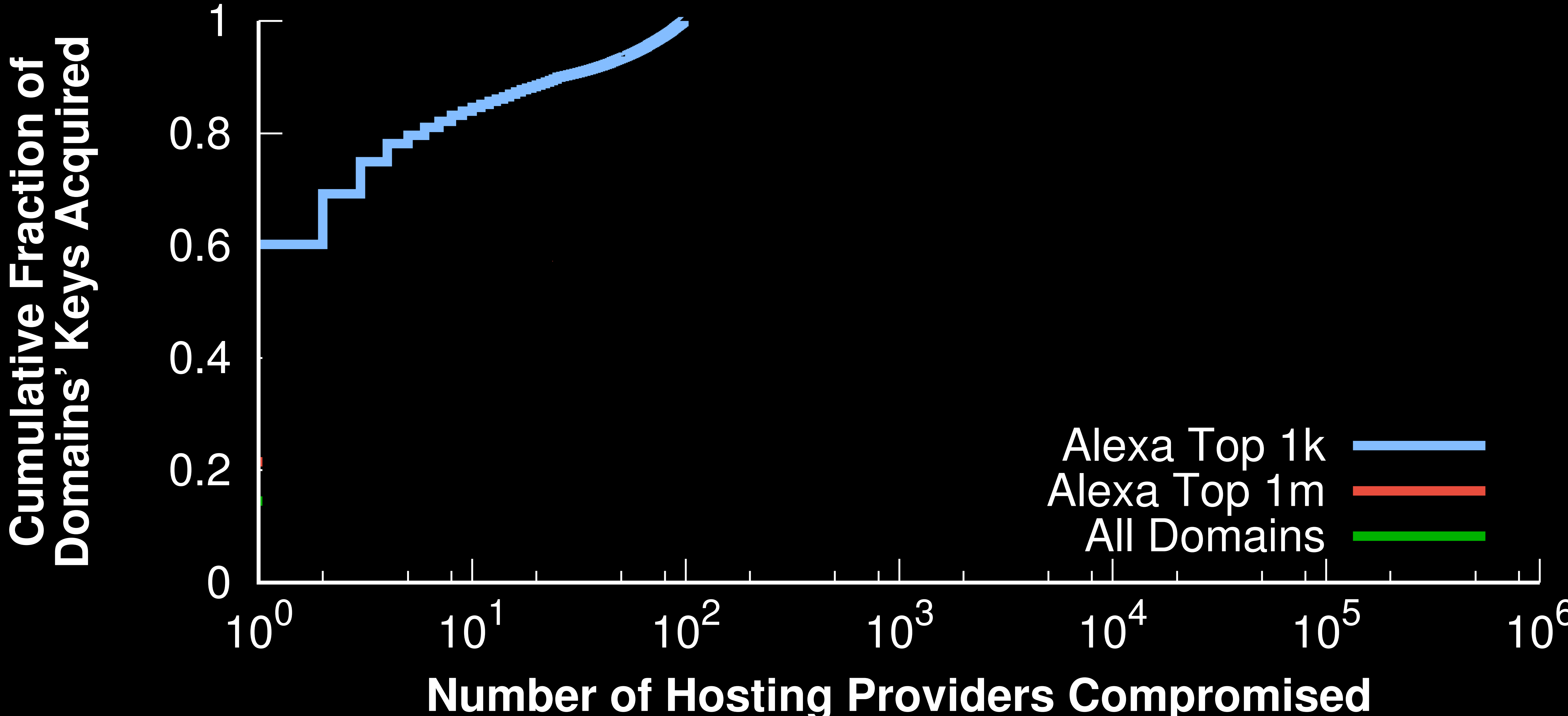


Key sharing is common across the Internet

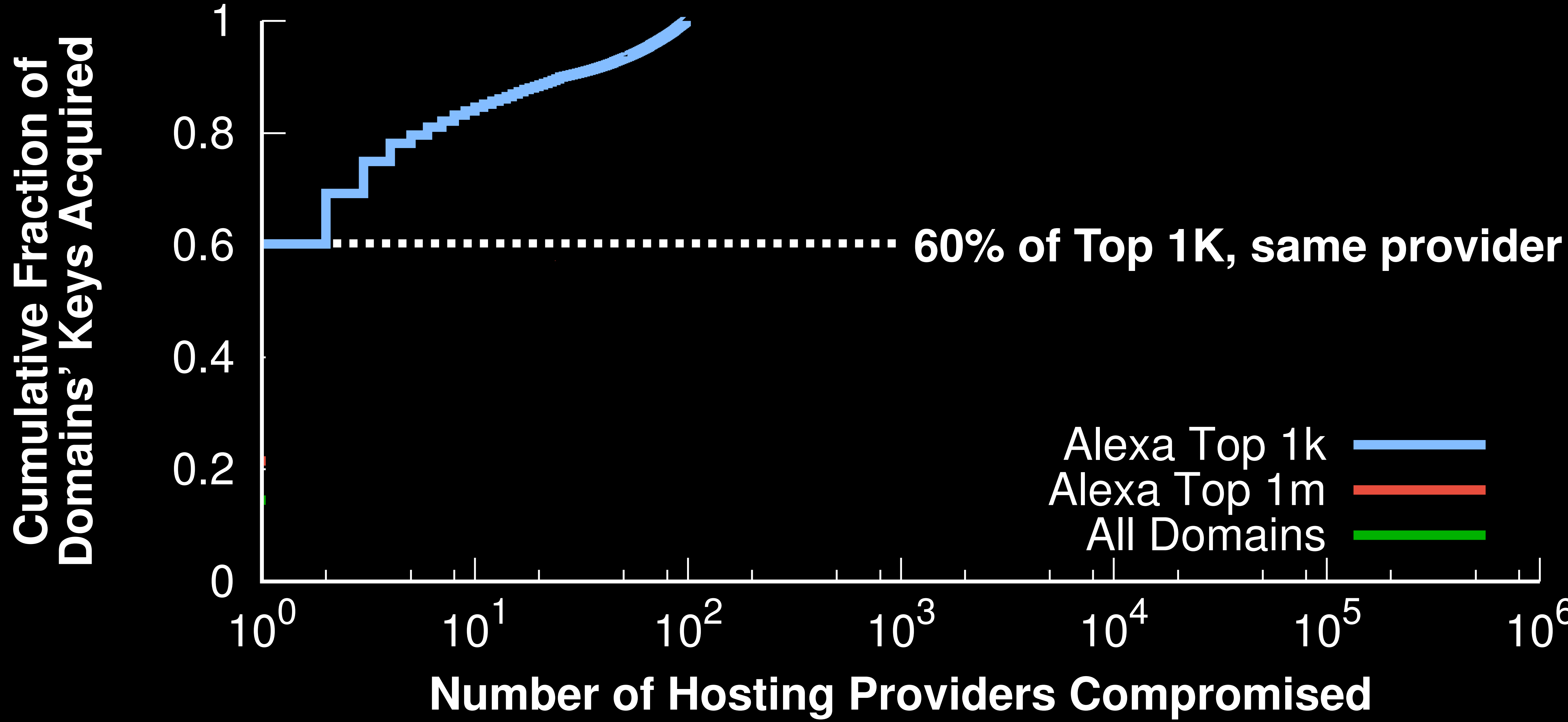
Does key sharing make enticing attack targets?



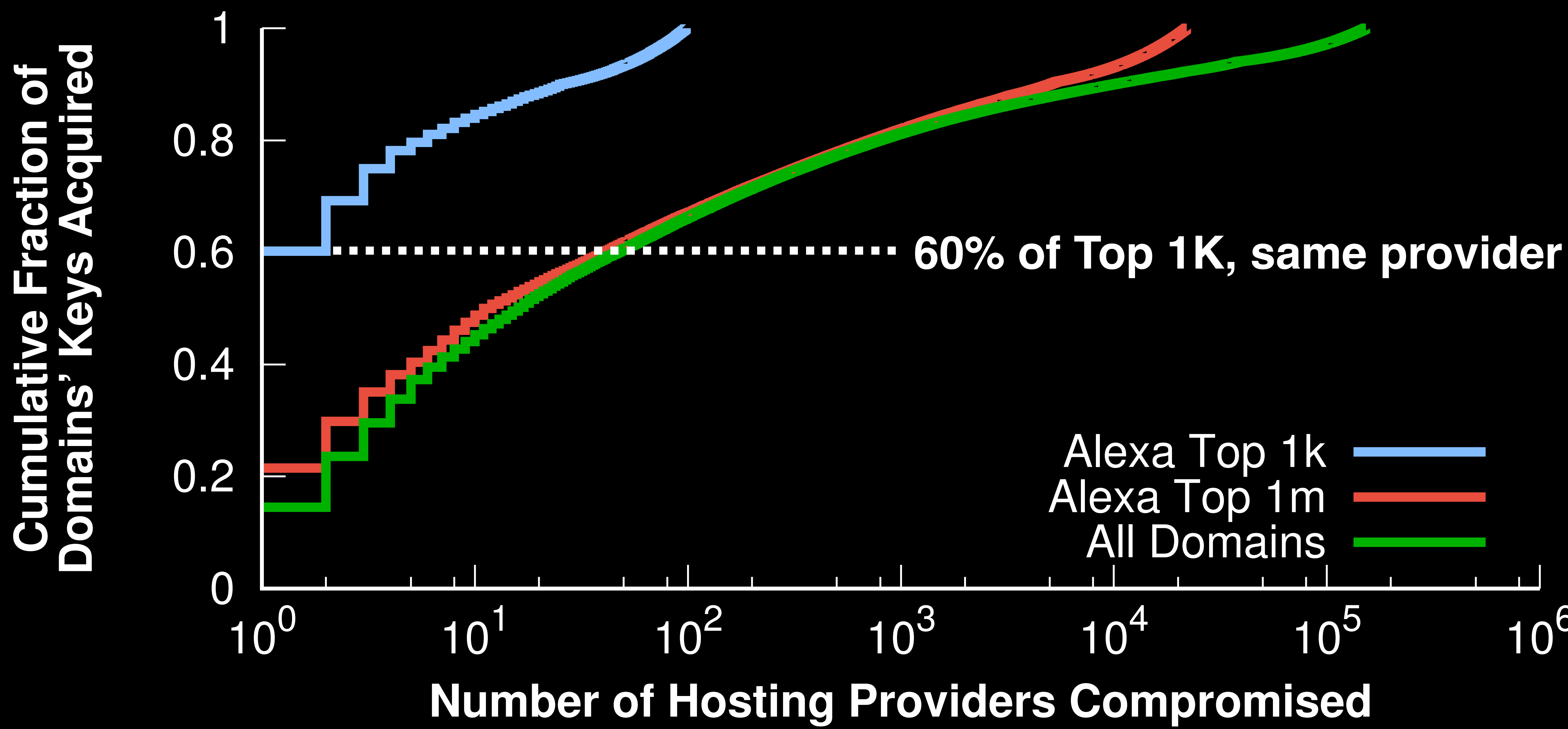
Does key sharing make enticing attack targets?



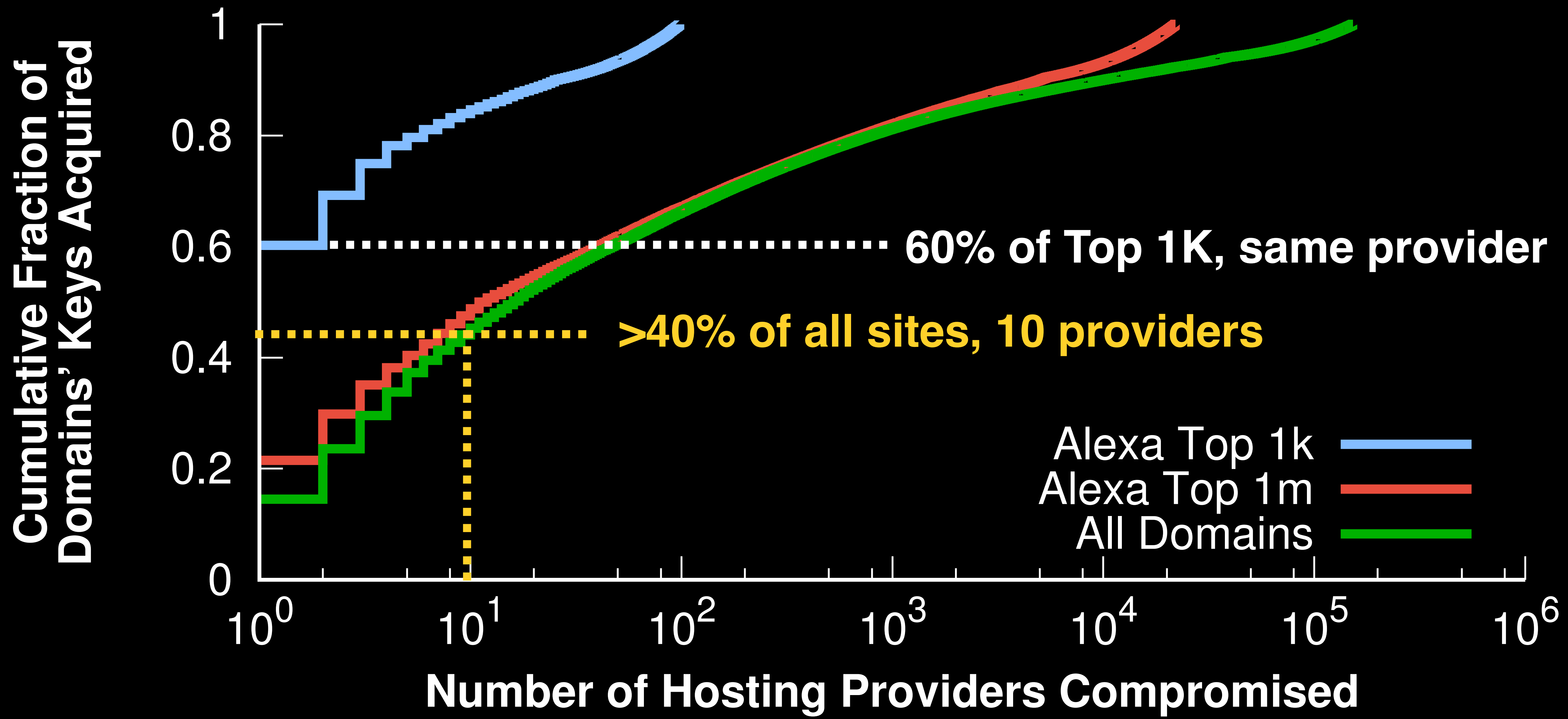
Does key sharing make enticing attack targets?



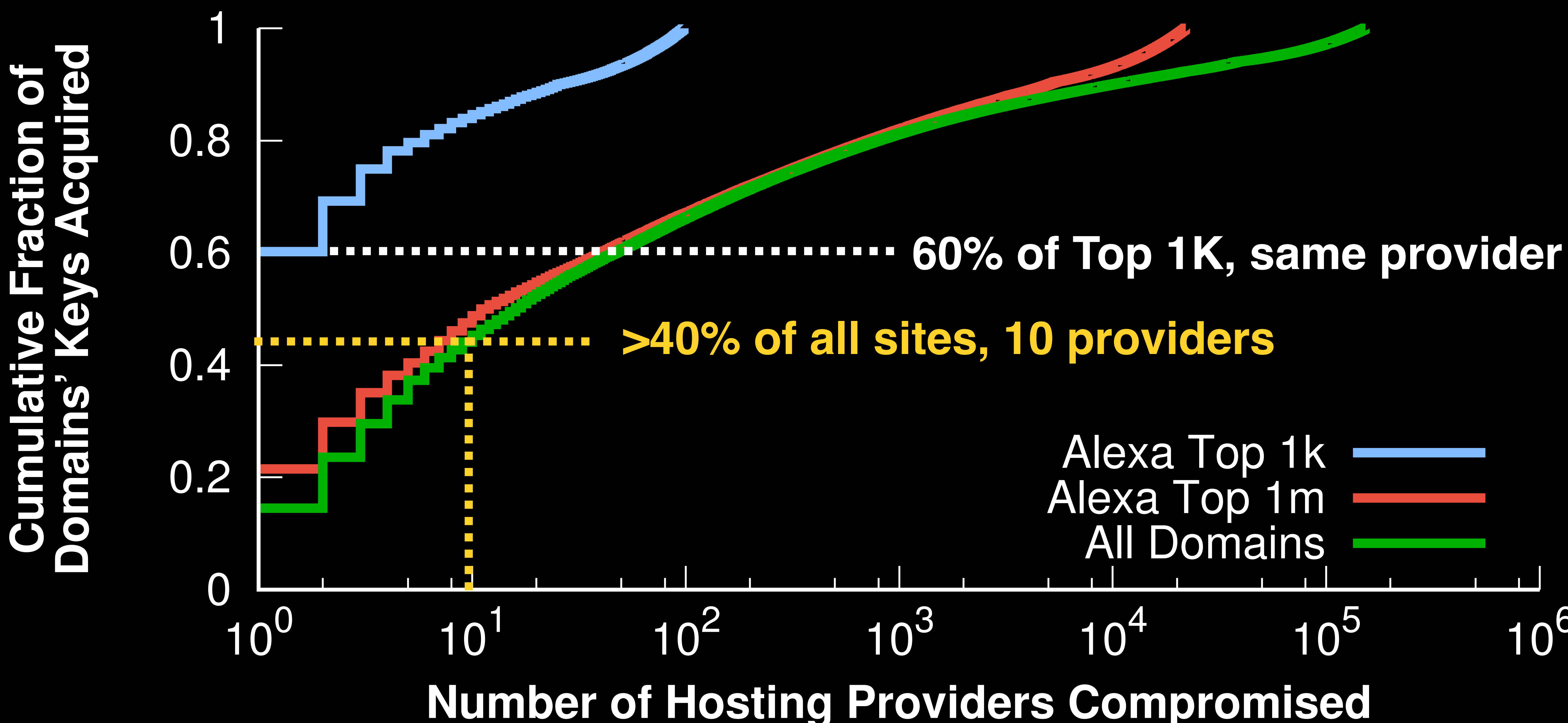
Does key sharing make enticing attack targets?



Does key sharing make enticing attack targets?



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

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)