CSE227 – Graduate Computer Security

TLS I

UC San Diego

Housekeeping

General course things to know

- Midpoint check-in document is due **2/14 at 11:59pm PT**
    - Introduction (frame the problem)
    - Related work section (should include ~5 – 10 relevant papers)
    - Research plan, current status, what's left to do
- Midpoint check-in meetings will happen the week of the 17th, more details to come

Today's lecture

Learning Objectives

- Talk about RSA, TLS, efficient factoring of numbers with shared primes, etc.
- Discuss the Ps and Qs paper, and why the paper is so significant

Preliminaries

Public-key Cryptosystems

- What is a public-key cryptosystem?

Public-key Cryptosystems

- What is a public-key cryptosystem?
  - A public-key cryptosystem is one where cryptography (e.g., encryption, integrity) is achieved through *pairs* of keys, a public key and a private key.

Public-key Cryptosystems

- What is a public-key cryptosystem?
    - A public-key cryptosystem is one where cryptography (e.g., encryption, integrity) is achieved through *pairs* of keys, a public key and a private key.
- What is a public key?

Public-key Cryptosystems

- What is a public-key cryptosystem?
  - A public-key cryptosystem is one where cryptography (e.g., encryption, integrity) is achieved through *pairs* of keys, a public key and a private key.
- What is a public key?
- What is a private key?

Public-key Cryptosystems

- What is a public-key cryptosystem?
  - A public-key cryptosystem is one where cryptography (e.g., encryption, integrity) is achieved through *pairs* of keys, a public key and a private key.
- What is a public key?
- What is a private key?
- What is the fundamental assumption of public keys and private keys?

Public-key Cryptosystems

- What is a public-key cryptosystem?
  - A public-key cryptosystem is one where cryptography (e.g., encryption, integrity) is achieved through *pairs* of keys, a public key and a private key.
- What is a public key?
- What is a private key?
- What is the fundamental assumption of public keys and private keys?
  - Two keys are easy to generate, but discovering private key (d) from the public key (e) is computationally *infeasible*

What is RSA?

What is RSA?

RSA (Rivest-Shamir-Adleman) is a public-key cryptosystem used for secure data transmission.

How RSA works

Alice

Bob

How RSA works

Alice

Alice's keypair generation scheme:

1. Generate two numbers, p and q. What is true about these numbers?

Bob

How RSA works

Alice

Alice's keypair generation scheme:

1. Generate two numbers, *p* and *q*. What is true about these numbers?
2. Computer N = p * q, this is known as the modulus. Is N public or private?

Bob

How RSA works

Alice

Alice's keypair generation scheme:

1. Generate two numbers, *p* and *q*. What is true about these numbers?
2. Computer N = p * q, this is known as the modulus. Is N public or private?
3. Compute λ(N), LCM(p-1,q-1), called the totient function

Bob

How RSA works

Alice

Alice's keypair generation scheme:

1. Generate two numbers, *p* and *q*. What is true about these numbers?
2. Computer N = p * q, this is known as the modulus. Is N public or private?
3. Compute λ(N), LCM(p-1,q-1), called the totient function
4. Choose *e*, such that e and λ(N) are co-prime. What it mean for two numbers to be co-prime?

Bob

How RSA works

Alice

Alice’s keypair generation scheme:

1. Generate two numbers, *p* and *q*. What is true about these numbers?
2. Computer N = p * q, this is known as the modulus. Is N public or private?
3. Compute λ(N), LCM(p-1,q-1), called the totient function
4. Choose *e*, such that e and λ(N) are co-prime. What it mean for two numbers to be co-prime?
5. Compute $d \equiv (e^{-1}) (\text{mod } \lambda(N))$, d is the modular multiplicative inverse of e mod $\lambda(N)$

Bob

How RSA works

Alice

In the end, Alice has two keys:

Public Key: (e, N)

Private Key: (d, N)

Bob

How RSA works

Alice

In the end, Alice has two keys:
Public Key: (e_alice, N_alice)
Private Key: (d_alice, N_alice)

Bob also has two keys:
Public Key: (e_bob, N_bob)
Private Key: (d_bob, N_bob)

Bob

How RSA works

Alice wants to send an encrypted message to Bob. How would she do that?

Pub:(e_alice,N_alice)
Priv: (d_alice,N_alice)

Alice

Pub:(e_bob,N_bob)
Priv: (d_bob,N_bob)

Bob

How RSA works

Alice wants to send an encrypted message to Bob. How would she do that?

Pub:(e_{alice}, N_{alice})
Priv: (d_{alice}, N_{alice})

Alice

$x = m^{e_{bob}} \bmod N$ →

Pub:(e_{bob}, N_{bob})
Priv: (d_{bob}, N_{bob})

Bob

How RSA works

Bob wants to sign a message m to Alice. How would he do that?

Pub:(e_alice,N_alice)
Priv: (d_alice,N_alice)

Alice

Pub:(e_bob,N_bob)
Priv: (d_bob,N_bob)

Bob

How RSA works

Bob wants to sign a message m to Alice. How would he do that?

Pub:(e_alice,N_alice)
Priv: (d_alice,N_alice)

Alice

m, m^d_bob mod N

Pub:(e_bob,N_bob)
Priv: (d_bob,N_bob)

Bob

Breaking RSA is hard

- If you wanted to break RSA (e.g., discover the private key) given just public key material: (e, N), how would you do it?

Breaking RSA is hard

- If you wanted to break RSA (e.g., discover the private key) given just public key material: (e, N), how would you do it?
    - You would need to decompose N into its prime factors, *p* and *q*. And forward compute. **This problem, called the Integer factorization problem, is computationally very hard to do.**

Breaking RSA is hard

- If you wanted to break RSA (e.g., discover the private key) given just public key material: (e, N), how would you do it?
  - You would need to decompose N into its prime factors, *p* and *q*. And forward compute. **This problem, called the Integer factorization problem, is computationally very hard to do.**
- Is breaking RSA in this way always impossible, then?

Breaking RSA is hard

- If you wanted to break RSA (e.g., discover the private key) given just public key material: (e, N), <u>how would you do it?</u>
  - You would need to decompose N into its prime factors, *p* and *q*. And forward compute. **This problem, called the Integer factorization problem, is computationally very hard to do.**
- Is breaking RSA in this way always impossible, then?
  - No, if your bit-size is small, then you need less computational power
  - And a million other side channels…. see padding oracles, poor implementations, etc.

What is Transport Layer Security (TLS)?

What is Transport Layer Security (TLS)?

"TLS: Widely adopted security protocol designed to facilitate privacy and data security for communication over the Internet." – Cloudflare

How TLS (1.3) works (vaguely)

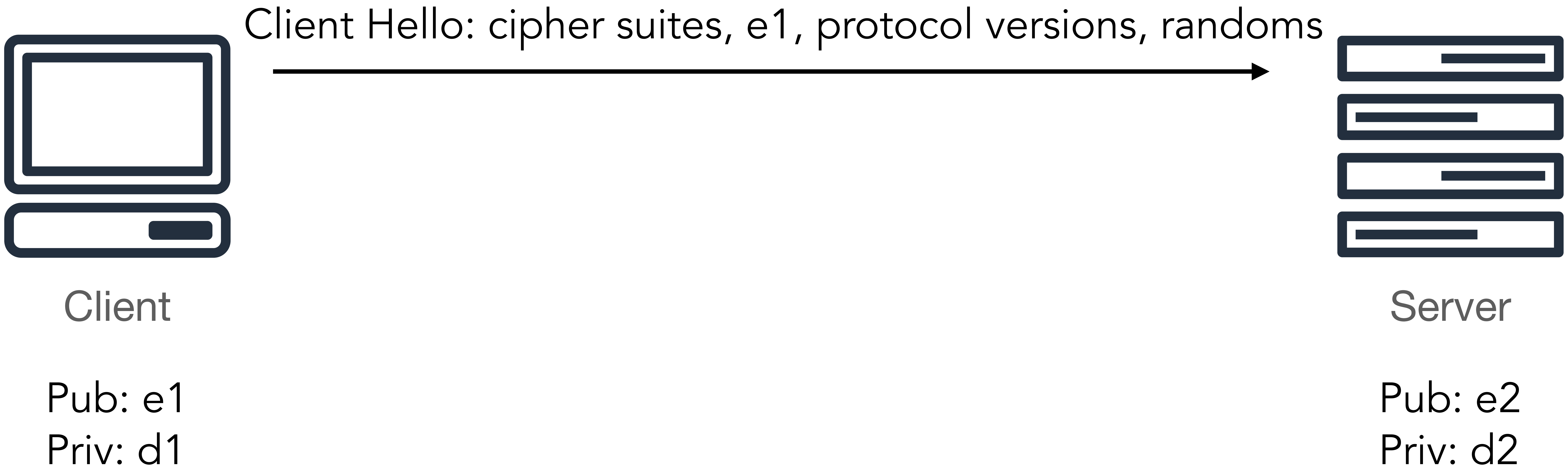
Client

Pub: e1
Priv: d1

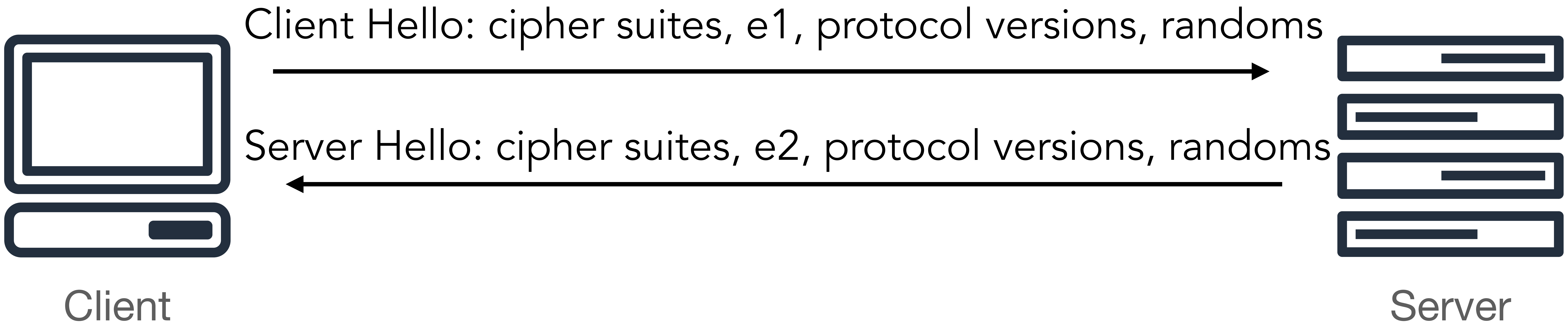
Server

Pub: e2
Priv: d2

How TLS (1.3) works (vaguely)



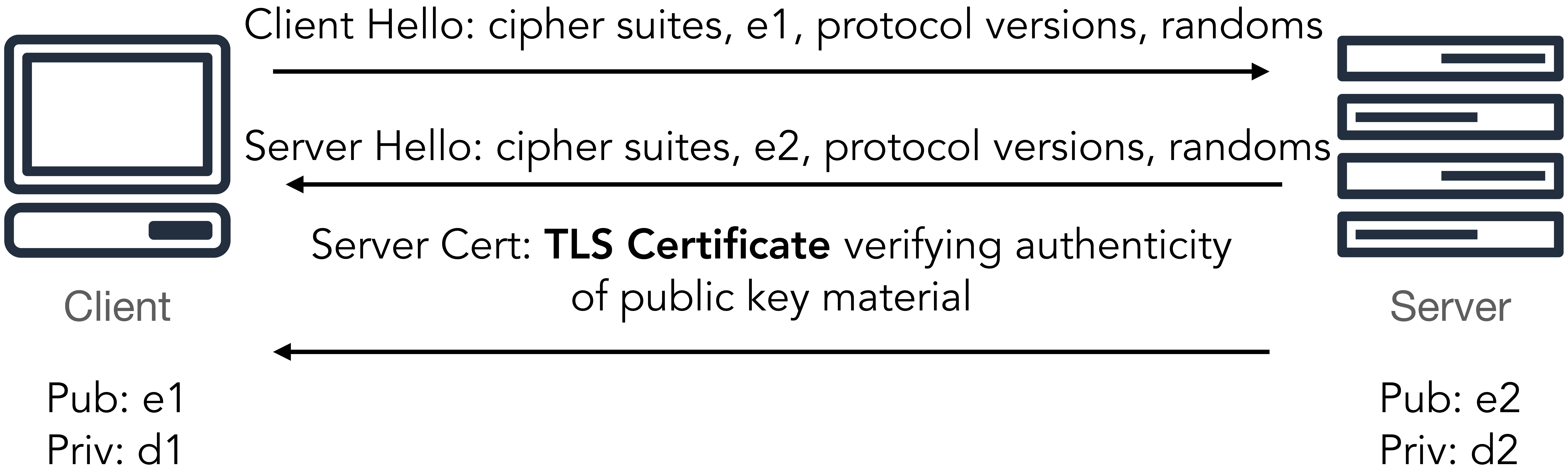
How TLS (1.3) works (vaguely)



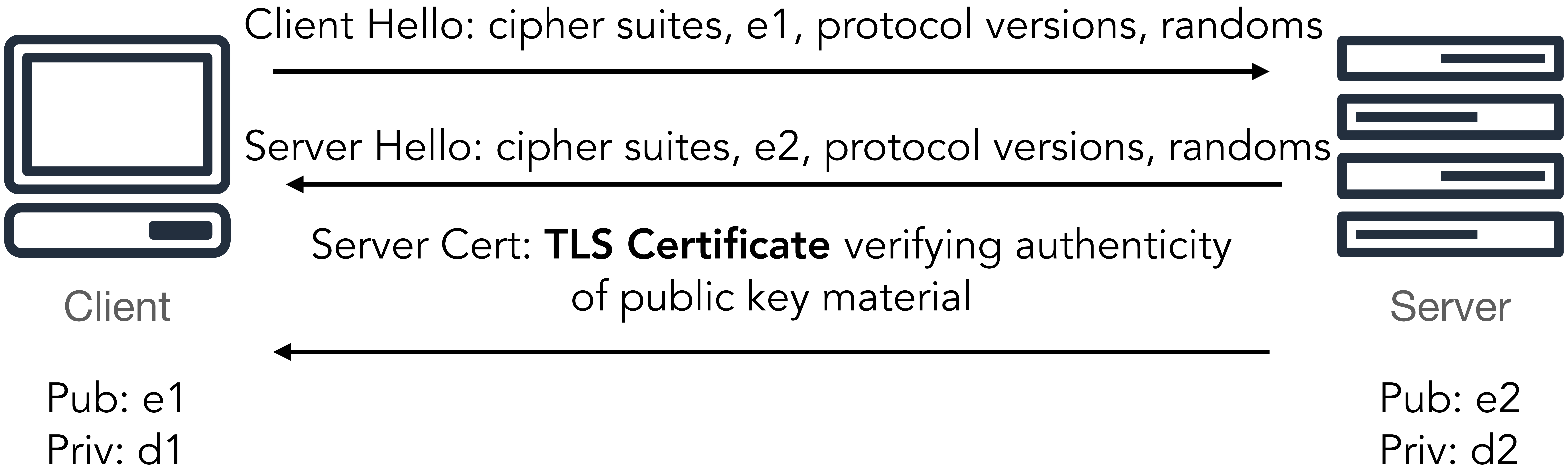
Pub: e1
Priv: d1

Pub: e2
Priv: d2

How TLS (1.3) works (vaguely)



How TLS (1.3) works (vaguely)



https://tls13.xargs.org/

Break Time + Attendance

Codeword:
MindYourBusiness

https://tinyurl.com/cse227-attend

Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices

A few words on this paper…

- This is a UCSD paper!
  - Nadia Heninger is one of the lead authors, done when she was a postdoc here at UCSD
  - The other lead was *my* postdoc advisor, done when he was a first-year graduate student (much like you!)
- This paper won best paper at USENIX Security 2012
- This paper won the USENIX Security Test-of-Time award in 2022
- Needless to say… it's a very important computer security paper. Why?

A few more words on this paper…

- IMO, one of the greatest paper titles of all time
- What does “minding your Ps and Qs” mean?

A few more words on this paper…

- IMO, one of the greatest paper titles of all time
- What does “minding your Ps and Qs” mean?
- So what does “mining your Ps and Qs” mean?
    - Factoring weak RSA public key information!

An RSA “vulnerability”

- This paper exploits a “vulnerability” in the RSA cryptosystem’s underlying assumptions. What assumption does it violate?

An RSA “vulnerability”

- This paper exploits a “vulnerability” in the RSA cryptosystem’s underlying assumptions. What assumption does it violate?
    - That public modulus N will **never share any factors with any other N.**

An RSA “vulnerability”

- This paper exploits a “vulnerability” in the RSA cryptosystem’s underlying assumptions. What assumption does it violate?
    - That public modulus N will **never share any factors with any other N.**
- Why is sharing factors so bad?

An RSA “vulnerability”

- This paper exploits a “vulnerability” in the RSA cryptosystem’s underlying assumptions. What assumption does it violate?
    - That public modulus N will **never share any factors with any other N.**
- Why is sharing factors so bad?
    - GCDs are efficiently computable (see Euclid’s algorithm)
    - If N1 and N2 share a factor *p*, the GCD between N1 and N2 would be *p*, and the key can be trivially broken

Who cares if you break someone’s private key?

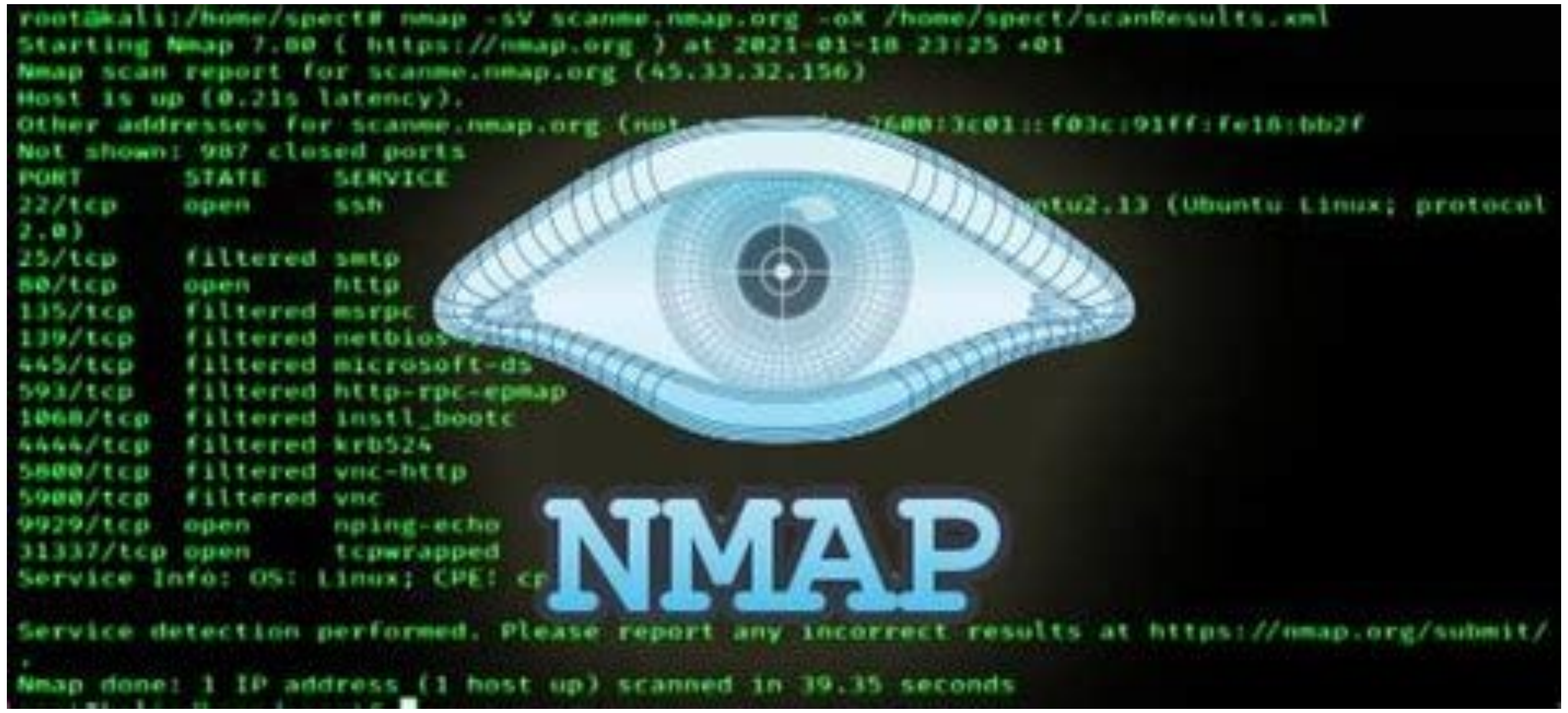
- What scenarios do the authors describe?

Who cares if you break someone's private key?

- What scenarios do the authors describe?
  - Passive attacker: If key exchange is RSA, then a passive attacker can decrypt **entire encrypted session** after the fact
  - Active attacker: If key exchange is Diffie-Hellman, passive adversary won't work, but active attacker (e.g., MiTM) can modify / decrypt traffic

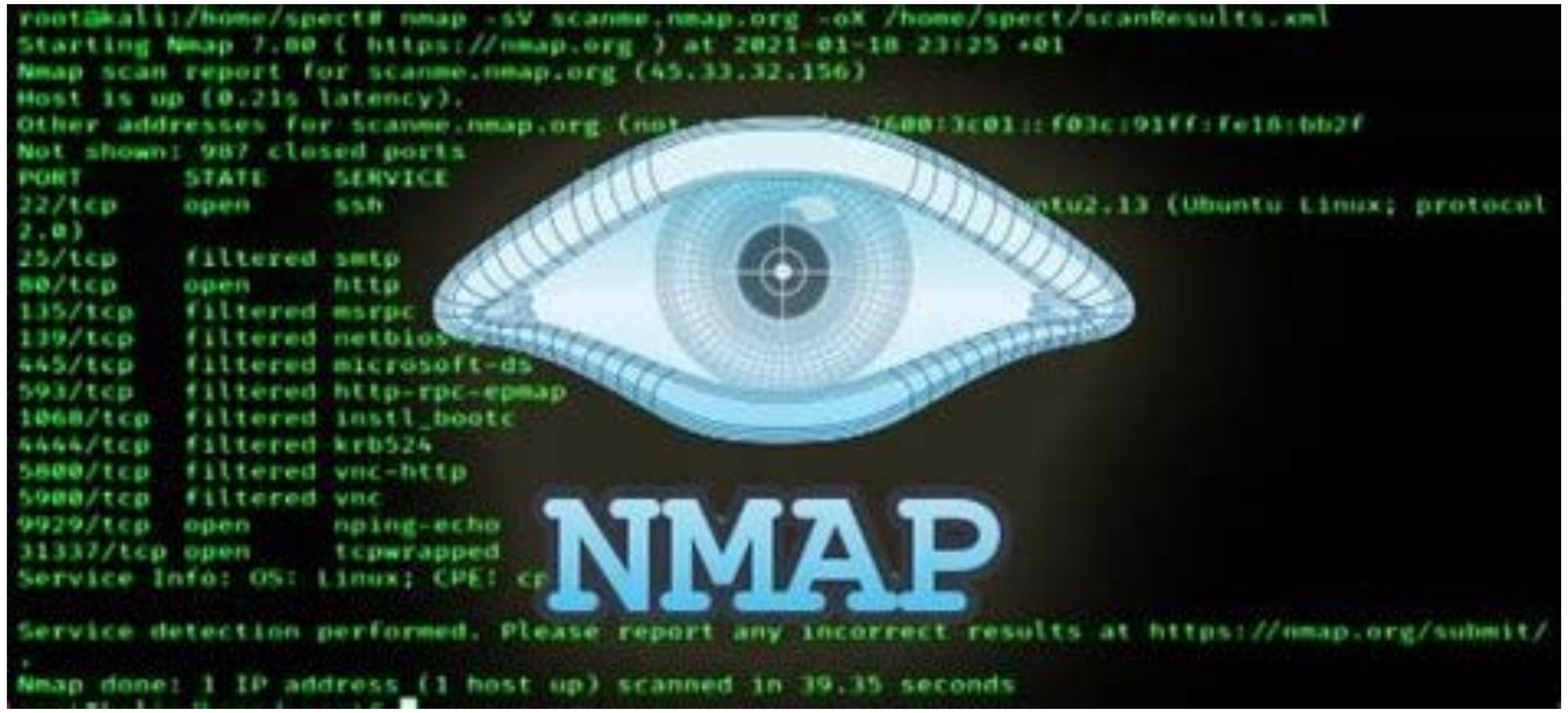
How do we find these vulnerable keys?

- What is network scanning?



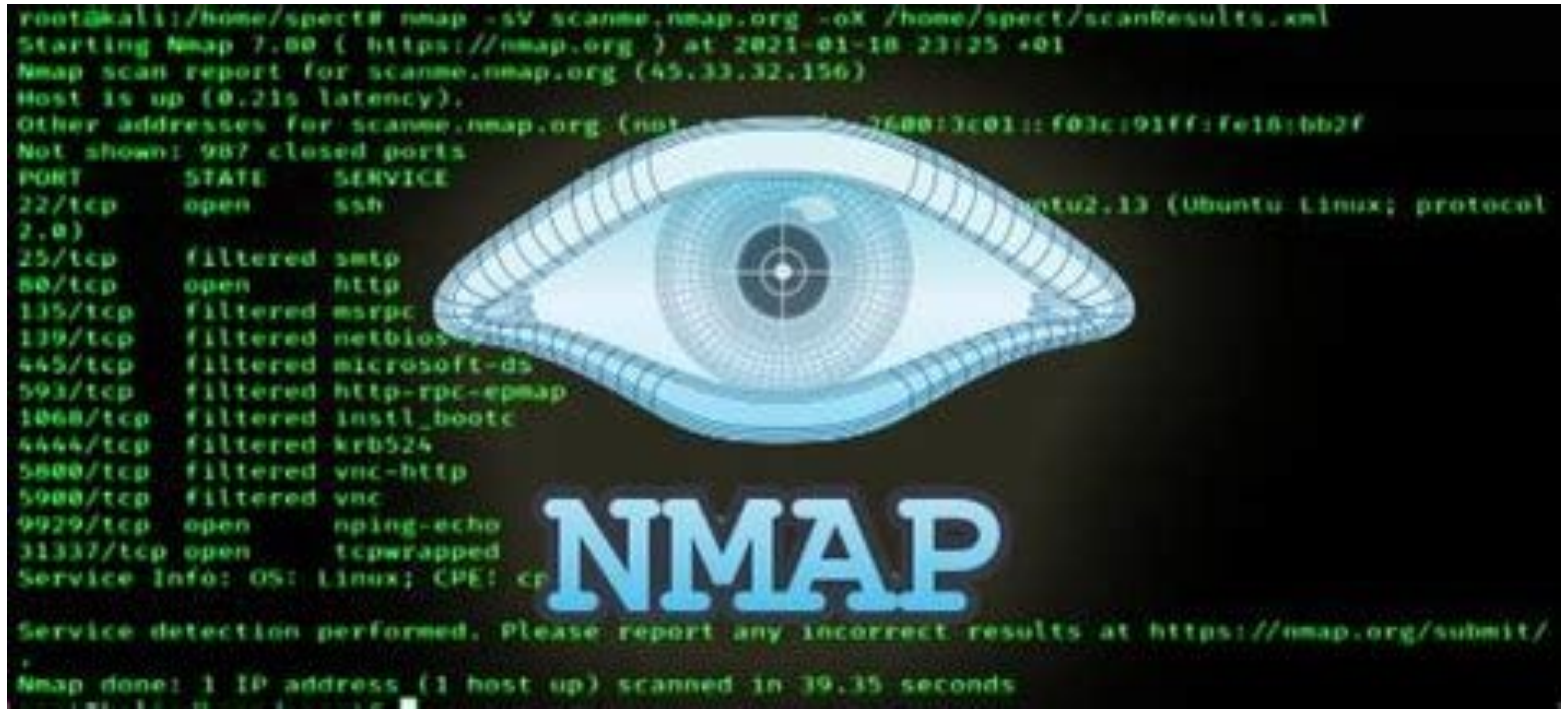
How do we find these vulnerable keys?

- What is network scanning?
- How does network scanning work to identify TCP hosts?



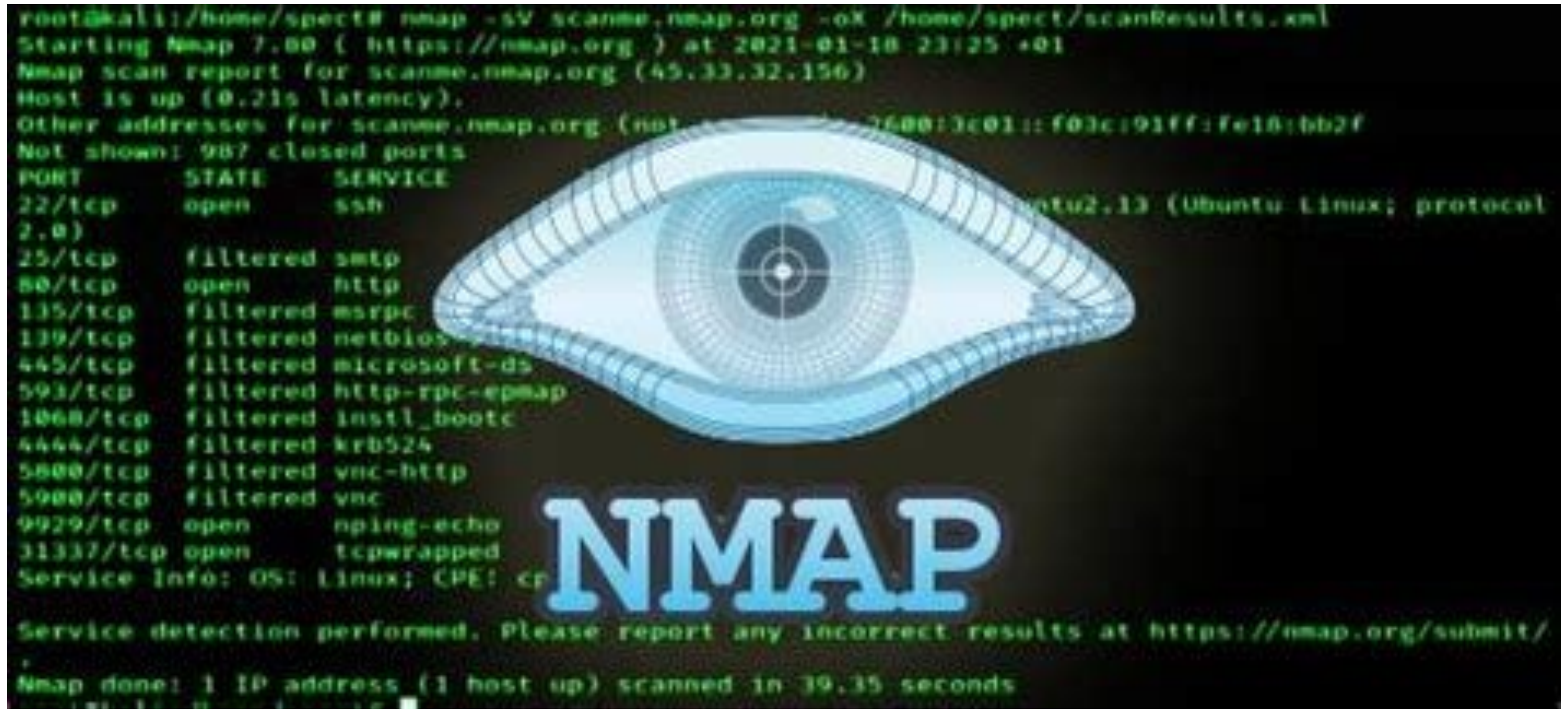
How do we find these vulnerable keys?

- What is network scanning?
- How does network scanning work to identify TCP hosts?
- What was the search space the authors searched in this paper?



How do we find these vulnerable keys?

- What is network scanning?
- How does network scanning work to identify TCP hosts?
- What was the search space the authors searched in this paper?
    - How long did it take the authors to scan the IPv4 Internet?



Scan Results

	SSL Observatory (12/2010)	Our TLS scan (10/2011)	Our SSH scans (2-4/2012)
Hosts with open port 443 or 22	≈16,200,000	28,923,800	23,237,081
Completed protocol handshakes	7,704,837	12,828,613	10,216,363
Distinct RSA public keys	3,933,366	5,656,519	3,821,639
Distinct DSA public keys	1,906	6,241	2,789,662
Distinct TLS certificates	4,021,766	5,847,957	—
Trusted by major browsers	1,455,391	1,956,267	—

Table 1: **Internet-wide scan results**—We exhaustively scanned the public IPv4 address space for TLS and SSH servers listening on ports 443 and 22, respectively. Our results constitute the largest such network survey reported to date. For comparison, we also show statistics for the EFF SSL Observatory's most recent public dataset [18].

Side note…

censys

Efficient Factorization

- Paper uses a combination of product trees and remainder trees to efficiently compute GCDs for their moduli
    - Computed GCDs for 11.1M moduli in about 5h, this would be much faster today

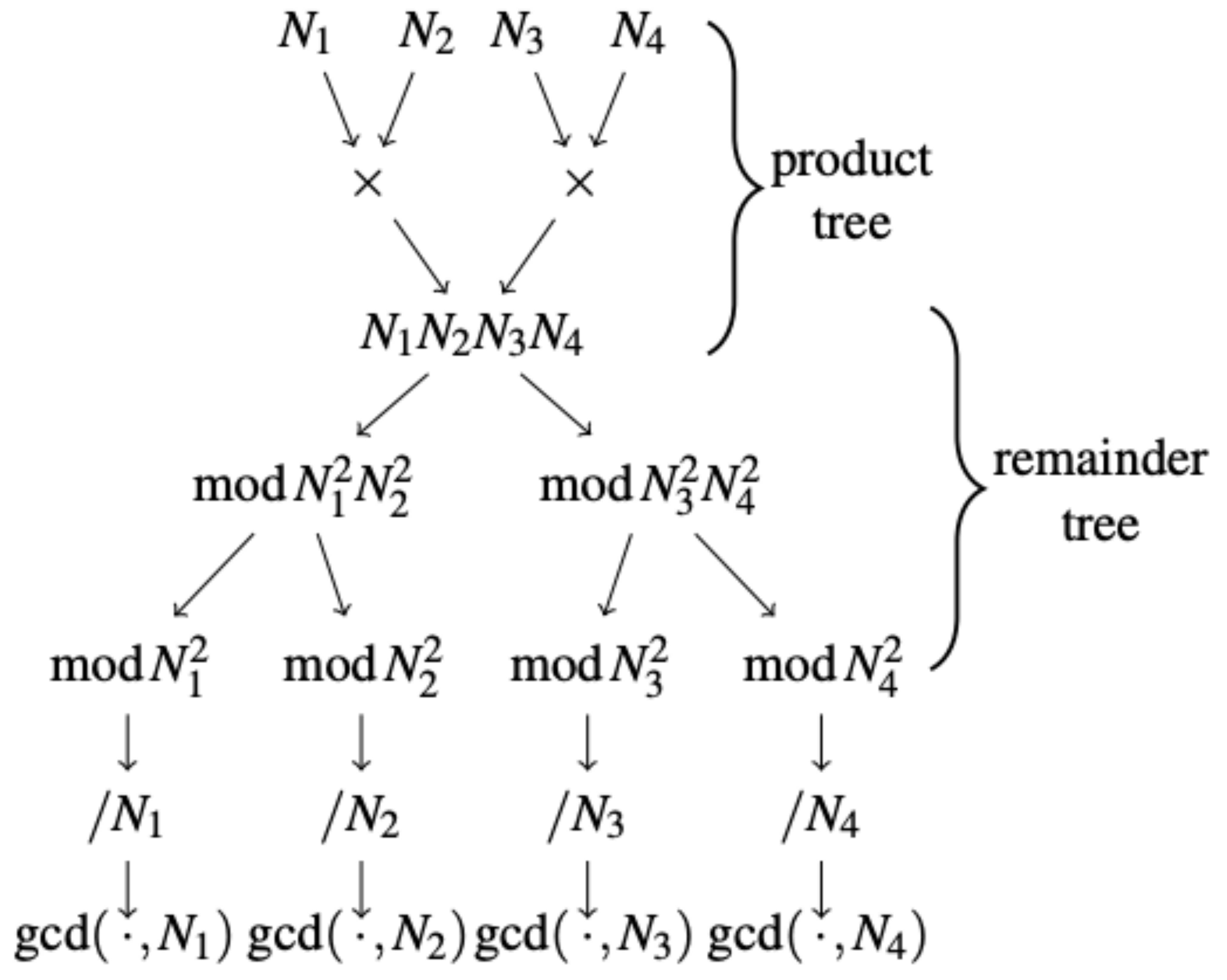


Figure 1: **Computing all-pairs GCDs efficiently** — We computed the GCD of every pair of RSA moduli in our dataset using an algorithm due to Bernstein [6].

Repeated Keys are Common

- Authors found 61% of TLS hosts **served the same key**
- <u>Why does this happen in practice?</u>

Repeated Keys are Common

- Authors found 61% of TLS hosts **served the same key**
- Why does this happen in practice?
  - Hosting providers might share keys for ease of deployment
  - Some keying material is embedded in firmware as *default* – **5.23%** of hosts were manufacturer defaults

Who is putting bad keys into the world?

- The authors identified many vulnerable device vendors and models. How?

Who is putting bad keys into the world?

- The authors identified many vulnerable device vendors and models. How?
  - There's **a lot** of information volunteered in TLS certificates…

Who is putting bad keys into the world?

- The authors identified many vulnerable device vendors and models. How?
  - There's **a lot** of information volunteered in TLS certificates…
- Authors identified vulnerable devices from 27 manufacturers. What were these manufacturers creating?

Who is putting bad keys into the world?

- The authors identified many vulnerable device vendors and models. How?
  - There's **a lot** of information volunteered in TLS certificates…
- Authors identified vulnerable devices from 27 manufacturers. What were these manufacturers creating?
  - Enterprise-grade routers, server management, VPN devices, security systems, consumer routers… things you *totally* want having bad crypto

What did the authors find?

	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
... using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
... using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
... using default certificates or default keys	670,391	(5.23%)		
... using low-entropy repeated keys	43,852	(0.34%)		
... using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
... using DSA keys we could compromise			105,728	(1.03%)
... using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
... using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
... identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
... model using low-entropy repeated keys	314,640	(2.45%)		

Table 2: **Summary of vulnerabilities** — We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

What did the authors find?

	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
. . . using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
. . . using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
. . . using default certificates or default keys	670,391	(5.23%)		
. . . using low-entropy repeated keys	43,852	(0.34%)		
. . . using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
. . . using DSA keys we could compromise			105,728	(1.03%)
. . . using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
. . . using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
. . . identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
. . . model using low-entropy repeated keys	314,640	(2.45%)		

Table 2: **Summary of vulnerabilities**—We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

Why is this happening?

- What is entropy?

Why is this happening?

- What is entropy?
  - “The amount of unpredictable randomness” in a physical system

Why is this happening?

- What is entropy?
  - “The amount of unpredictable randomness” in a physical system
- Where does entropy come from?

Why is this happening?

- What is entropy?
  - “The amount of unpredictable randomness” in a physical system
- Where does entropy come from?
  - Uninitialized contents of memory when the kernel starts, startup clock time, disk access timings, “old” entropy

Why is this happening?

- What is entropy?
  - "The amount of unpredictable randomness" in a physical system
- Where does entropy come from?
  - Uninitialized contents of memory when the kernel starts, startup clock time, disk access timings, "old" entropy
- What did the authors discover about headless / embedded devices?

Implementations are tricky

- Linux provides two sources of randomness: /dev/random and /dev/urandom. What's the difference between the two?

Implementations are tricky

- Linux provides two sources of randomness: /dev/random and /dev/urandom. What's the difference between the two?
  - In 2012, /dev/random was **blocking**, /dev/urandom was **non-blocking,** now they're mostly the same after initialization

Implementations are tricky

- Linux provides two sources of randomness: /dev/random and /dev/urandom. What's the difference between the two?
    - In 2012, /dev/random was **blocking**, /dev/urandom was **non-blocking,** now they're mostly the same after initialization
- People preferred the **non-blocking** interface for randomness (even when the randomness was predictable). Why?

Meta-thoughts on the paper

- What do we think about this paper? Did we enjoy it, why or why not?
- Why do we think this paper won so many awards when the results only impacted such a small % of hosts?
- What were some limitations of the study you can think of?

Next time…

- More TLS! Two papers, focused on **certificates** and **authenticity**
  - One of them is my paper
- Work on your projects. Midpoint check-ins are soon :)