

# You Can Rand but You Can't Hide: A Holistic Security Analysis of Google Fuchsia's (and gVisor's) Network Stack

**Network and Distributed System Security (NDSS) Symposium, 2025**

**Inon Kaplan** (independent researcher)

**Ron Even** (independent researcher)

**Amit Klein** (Hebrew University of Jerusalem – School of CS and Eng.)

# Background: Google Fuchsia and gVisor

# Background: Google Fuchsia and gVisor

- Google **Fuchsia** is:
  - “A general purpose **operating system** designed to power a diverse ecosystem of hardware and software” (Google)
  - Targeting Mobile/Tablets/IoT, etc.
  - Deployed to millions of Google Nest Hub devices
  - Conjectured by many to (eventually?) replace Android
  - **Fuchsia’s TCP/IP stack (“NetStack”) is cloned from gVisor**



# Background: Google Fuchsia and gVisor

- Google **Fuchsia** is:

- “A general purpose **operating system** designed to power a diverse ecosystem of hardware and software” (Google)
- Targeting Mobile/Tablets/IoT, etc.
- Deployed to millions of Google Nest Hub devices
- Conjectured by many to (eventually?) replace Android
- **Fuchsia’s TCP/IP stack (“NetStack”) is cloned from gVisor**



- Google **gVisor** is:

- “an **application kernel** for containers” (Google)
- Used in **Google Cloud** offerings: App Engine, Cloud Functions, Cloud ML Engine, Cloud Run, Google Kubernetes Engine (GKE)



# Approach: Holistic Security Analysis


# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse

# Approach: Holistic Security Analysis



- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID
  - IPv6 Flow Label
  - UDP source port
  - TCP source port
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)

# Approach: Holistic Security Analysis



- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label
  - UDP source port
  - TCP source port
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)






# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label 
  - UDP source port
  - TCP source port
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)





# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port
  - TCP source port
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)






# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port 
  - TCP source port
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)







# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port 
  - TCP source port 
  - TCP timestamp (TS)
  - TCP initial sequence number (ISN)







# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port 
  - TCP source port 
  - TCP timestamp (TS) 
  - TCP initial sequence number (ISN)

# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port 
  - TCP source port 
  - TCP timestamp (TS) 
  - TCP initial sequence number (ISN) 

# Approach: Holistic Security Analysis

- Analyzing the entire Fuchsia/gVisor TCP/IP stack en-masse
- Focus on “high entropy” protocol header fields:
  - IPv4 (and IPv6) ID 
  - IPv6 Flow Label  (always 0...)
  - UDP source port 
  - TCP source port 
  - TCP timestamp (TS) 
  - TCP initial sequence number (ISN) 
- Combine otherwise-weak vulnerabilities in separate network protocol header fields into powerful attacks

**IPv4**

**TCP/IPv4**

**TCP/IPv6**

**UDP**



## IPv4

Small **IP ID** hash table,  
small hash key, ++

## TCP/IPv4

Small PRNG seed  
(**UDP source port**, TCP  
secrets)

**TCP TS** weak hash

**TCP source port** weak  
hash, global counter

**TCP ISN** weak hash

## TCP/IPv6

## UDP

PRNG advancing  
weakness (**UDP source  
port**)

## IPv4

## TCP/IPv4

## TCP/IPv6

## UDP

Small IP ID hash table,  
small hash key, ++

Small PRNG seed  
(UDP source port, TCP  
secrets)

TCP TS weak hash

TCP source port weak  
hash, global counter

TCP ISN weak hash

Reverse to  
seed  
(4 packets)

PRNG advancing  
weakness (UDP source  
port)

Net. stack PRNG seed  
(31 bits)

IPv4

TCP/IPv4

TCP/IPv6

UDP

Small IP ID hash table,  
small hash key, ++

Small PRNG seed  
(UDP source port, TCP  
secrets)

TCP TS weak hash

TCP source port weak  
hash, global counter

TCP ISN weak hash

PRNG advancing  
weakness (UDP source  
port)

Reverse to  
seed  
(4 packets)

Reverse to  
seed  
(2 packets)

Net. stack PRNG seed  
(31 bits)

IPv4

TCP/IPv4

TCP/IPv6

UDP

Small IP ID hash table,  
small hash key, ++

Small PRNG seed  
(UDP source port, TCP  
secrets)

TCP TS weak hash

TCP source port weak  
hash, global counter

TCP ISN weak hash

PRNG advancing  
weakness (UDP source  
port)

Reverse to  
seed  
(4 packets)

Reverse to  
seed  
(2 packets)

Rainbow table  
attack  
(0 packets)

Net. stack PRNG seed  
(31 bits)

IPv4

TCP/IPv4

TCP/IPv6

UDP

Small IP ID hash table,  
small hash key, ++

Small PRNG seed  
(UDP source port, TCP  
secrets)

TCP TS weak hash

TCP source port weak  
hash, global counter

TCP ISN weak hash

PRNG advancing  
weakness (UDP source  
port)

Reverse to  
seed  
(4 packets)

Reverse to  
seed  
(2 packets)

Rainbow table  
attack  
(0 packets)

IPv4 internal address

Time since boot

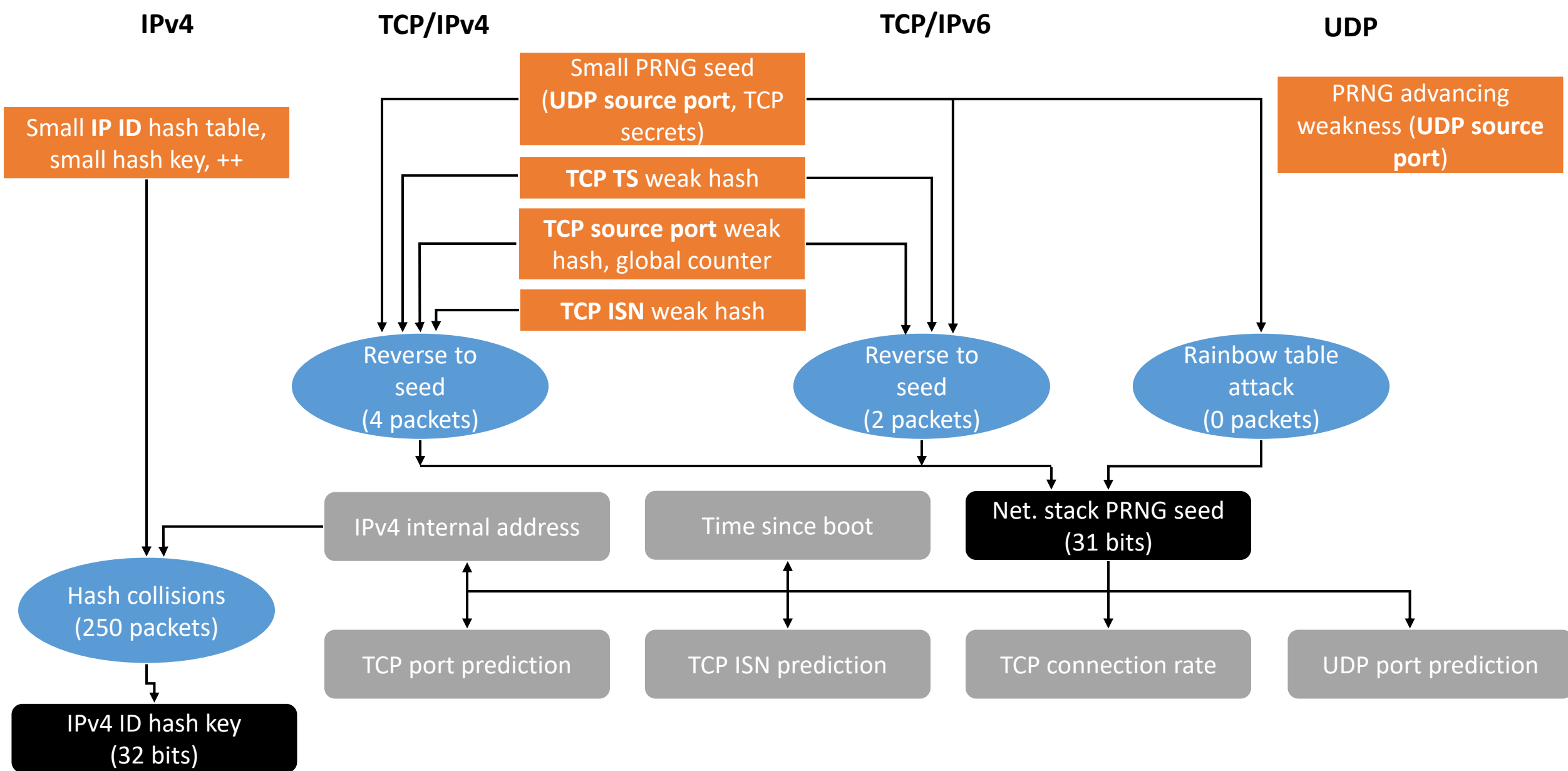
Net. stack PRNG seed  
(31 bits)

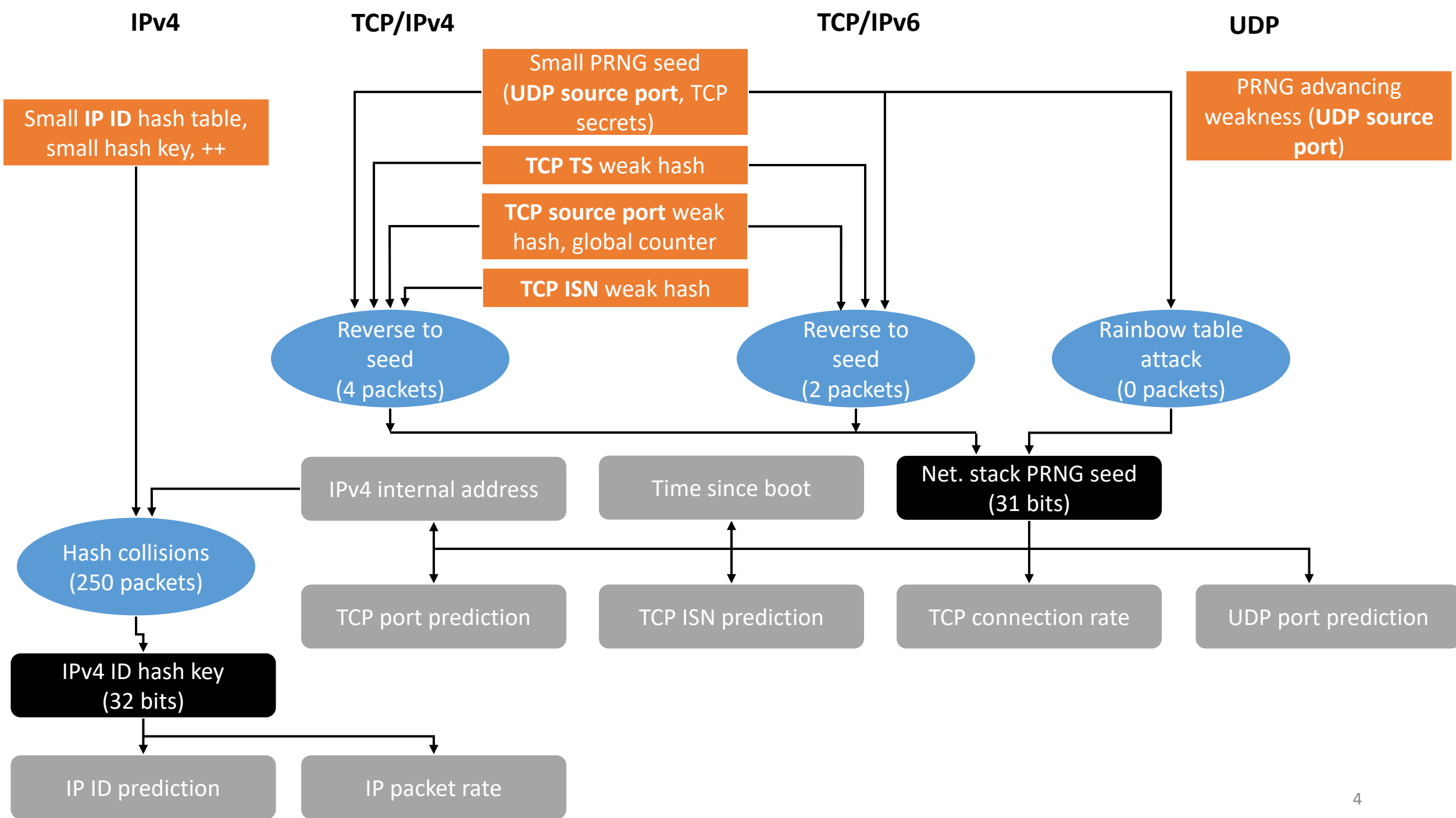
TCP port prediction

TCP ISN prediction

TCP connection rate

UDP port prediction



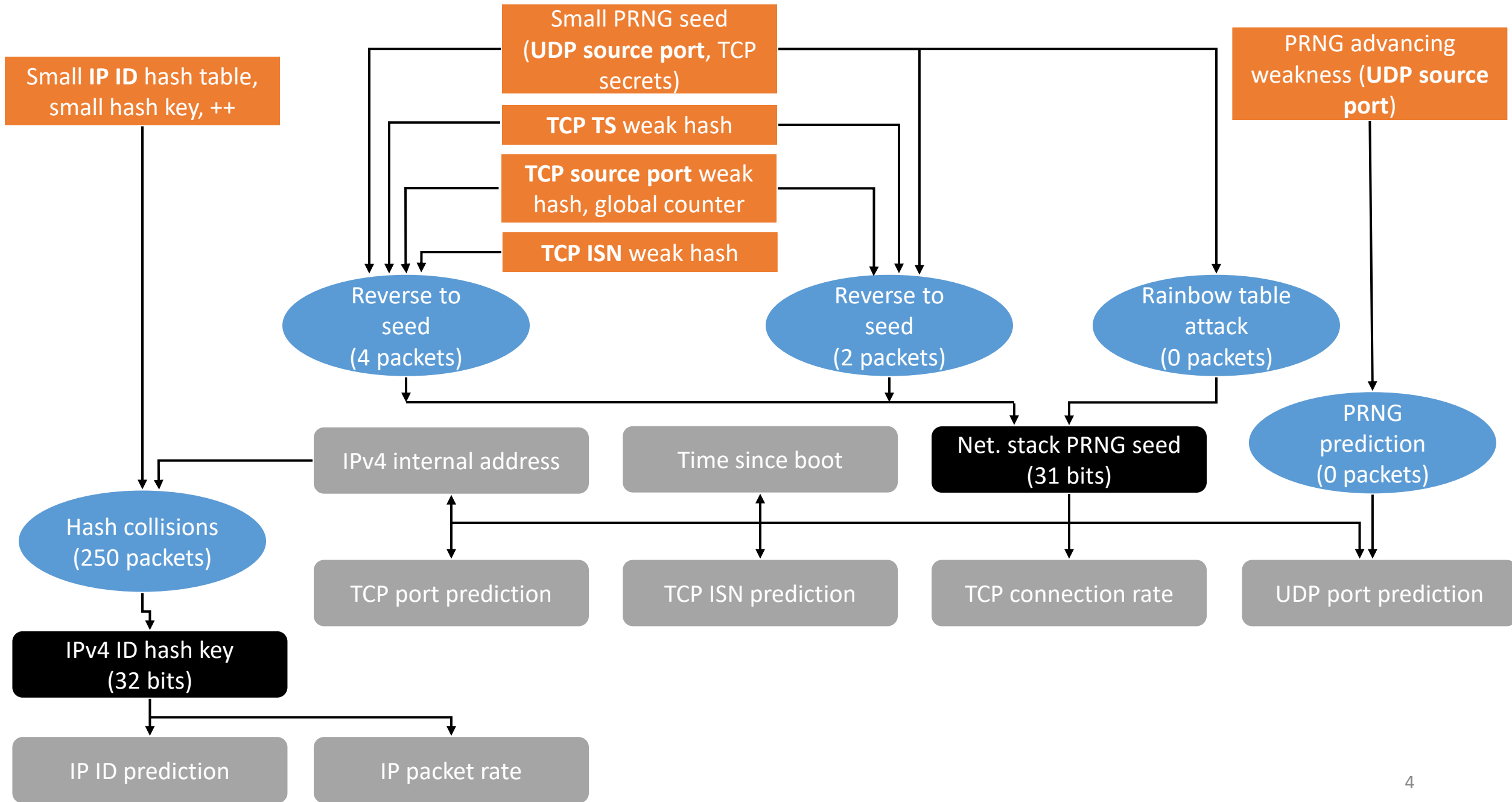


## IPv4

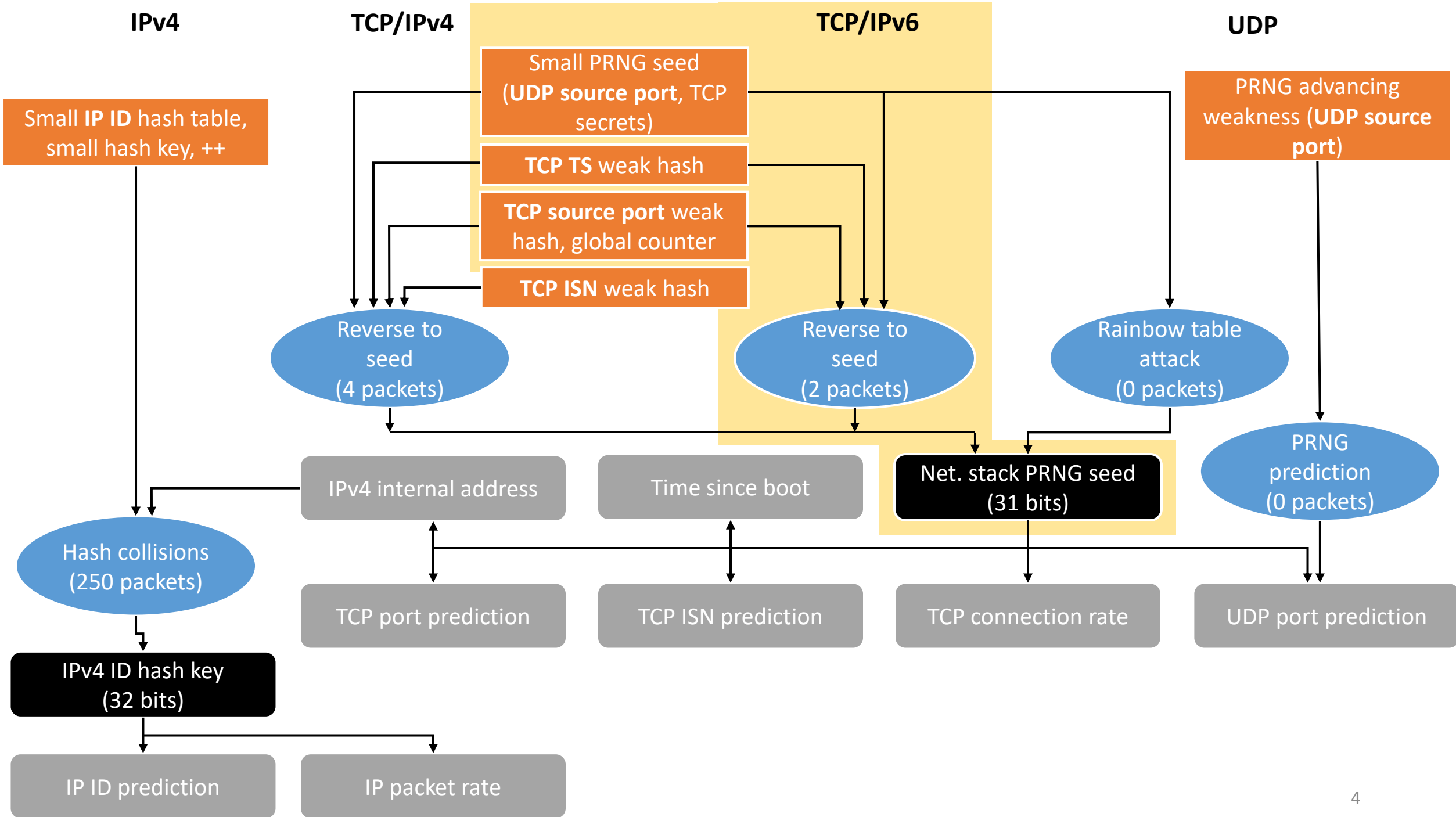
## TCP/IPv4

## TCP/IPv6

## UDP







# Definitions and Observations

- TCP Timestamp generation:  $TS = Hash(IP_{src}|IP_{dst}, key) + t_{[ms]} \bmod 2^{32}$
- $Hash(X, state)$  is (**simplified here for ease of discussion**) Jenkins' "one-at-a-time hash"
  - $X$  is a byte array
  - $state$  is a 32-bit internal state/key
  - For ease of discussion, we ignore an easily reversible final function ( $Sum32(\cdot)$ )
- $key$  is a kernel (TCP/IP stack) 32-bit key ("secret") generated deterministically from a 31-bit *seed* at system startup
  - Valid from system startup to system shutdown
- It's weak:
  - Small internal state (32 bits)
  - $Hash(X, state)$  - given  $X$  and  $Hash(X, state)$  we can easily find  $state$  (we call this "**peeling**")
  - $Hash(X|Y, state) = Hash(Y, Hash(X, state))$  (we call this "**chaining rule**")
- We define  $J = Hash(IP_{src}, key)$ , so (by **chaining rule**)  
 $TS = Hash(IP_{dst}, J) + t_{[ms]} \bmod 2^{32}$

# Finding $J$

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):  
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):  
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):  
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)
- RHS: all known except  $J$



# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):  
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)
- RHS: all known except  $J$
- So:  $known = f_{known}(J)$  where  $f_{known}(\cdot)$  is computable in offline!

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d) \quad TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d) \quad TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)
- RHS: all known except  $J$
- So:  $known = f_{known}(J)$  where  $f_{known}(\cdot)$  is computable in offline!
- In offline, invert it: Go over all  $2^{32}$   $J$  values, create a multimap  $Q$  ( $2^{32}$  entries):
$$Q : (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32} \rightarrow J$$

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):  
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)
- RHS: all known except  $J$
- So:  $known = f_{known}(J)$  where  $f_{known}(\cdot)$  is computable in offline!
- In offline, invert it: Go over all  $2^{32}$   $J$  values, create a multimap  $Q$  ( $2^{32}$  entries):  
$$Q : (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32} \rightarrow J$$
- Find  $J$  candidates in few opcodes (20 table lookups in  $Q$ ).

# Finding $J$

- We force the device to send 2 TCP SYN packets, **rapidly** (ms away):
  - Packet 1:  $(IP_{src} : p_1) \rightarrow (IP_{Attacker1} : d)$        $TS_1 = Hash(IP_{Attacker1}, J) + t_1 [ms] \bmod 2^{32}$
  - Packet 2:  $(IP_{src} : p_2) \rightarrow (IP_{Attacker2} : d)$        $TS_2 = Hash(IP_{Attacker2}, J) + t_2 [ms] \bmod 2^{32}$
- Subtracting and rearranging (with  $\Delta t = t_2 [ms] - t_1 [ms]$ ):
$$TS_2 - TS_1 - \Delta t \bmod 2^{32} = (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32}$$
- LHS: all known (up to  $\Delta t$  which is small – 0-19 ms)
- RHS: all known except  $J$
- So:  $known = f_{known}(J)$  where  $f_{known}(\cdot)$  is computable in offline!
- In offline, invert it: Go over all  $2^{32}$   $J$  values, create a multimap  $Q$  ( $2^{32}$  entries):
$$Q : (Hash(IP_{Attacker2}, J) - Hash(IP_{Attacker1}, J)) \bmod 2^{32} \rightarrow J$$
- Find  $J$  candidates in few opcodes (20 table lookups in  $Q$ ).
- Output: a few (=20)  $J$  candidates

# Finding and Verifying *seed*

- From  $J$  (candidate) we **peel** (with  $IP_{src}$ ) to find  $key$
- We find  $seed$  using another offline computed multi-map table  $W$ :

$$W: key \rightarrow seed$$

- From  $seed$ , we can generate the TCP source port secret  $key'$  and use it to eliminate false positives ( $d$  is the attacker destination port):

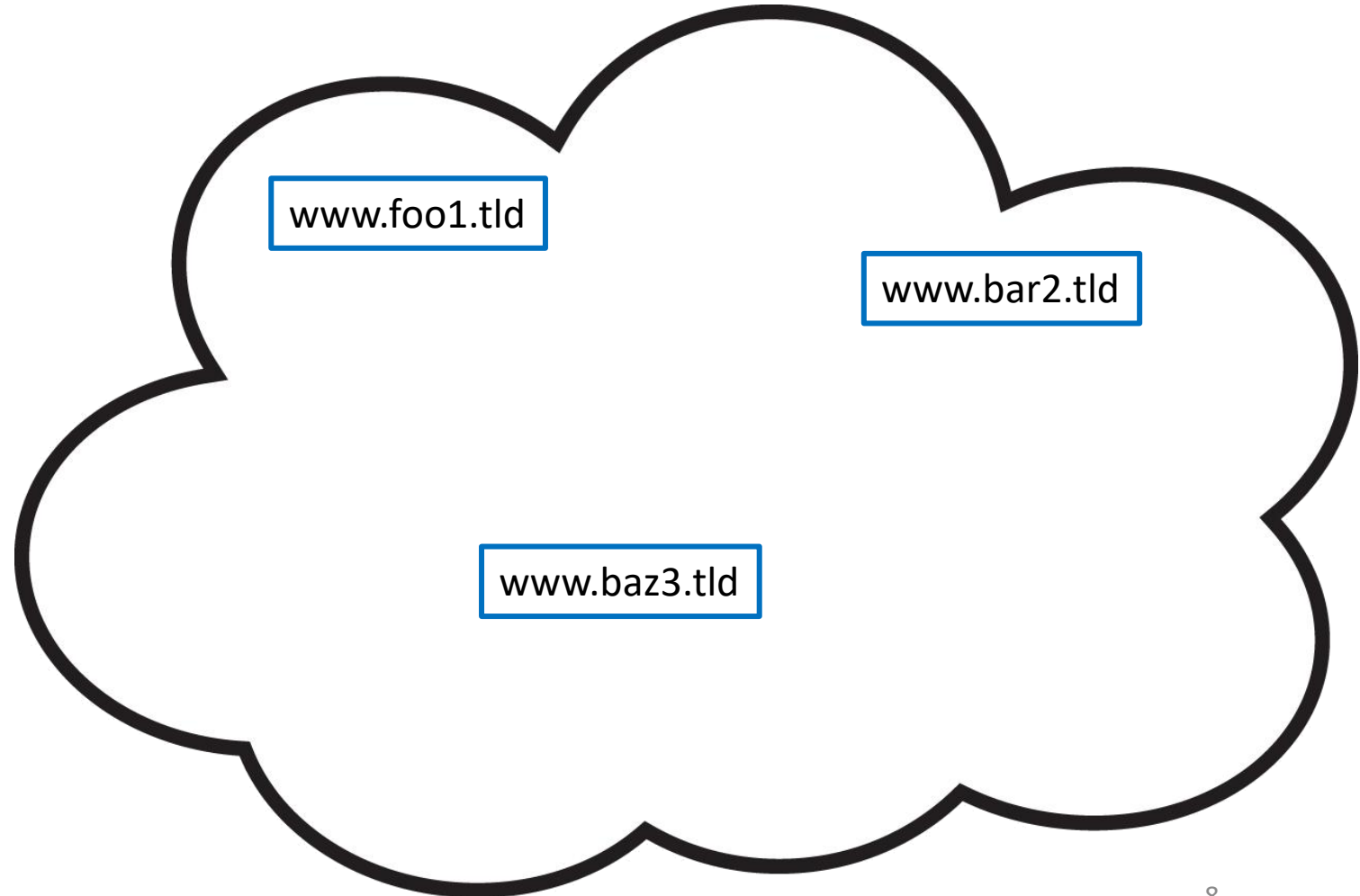
$$p_1 = (c + Hash(IP_{src}|Attacker_1|d, key')) \bmod 49536 + 16000$$

$$p_2 = (c + \mathbf{1} + Hash(IP_{src}|Attacker_2|d, key')) \bmod 49536 + 16000$$

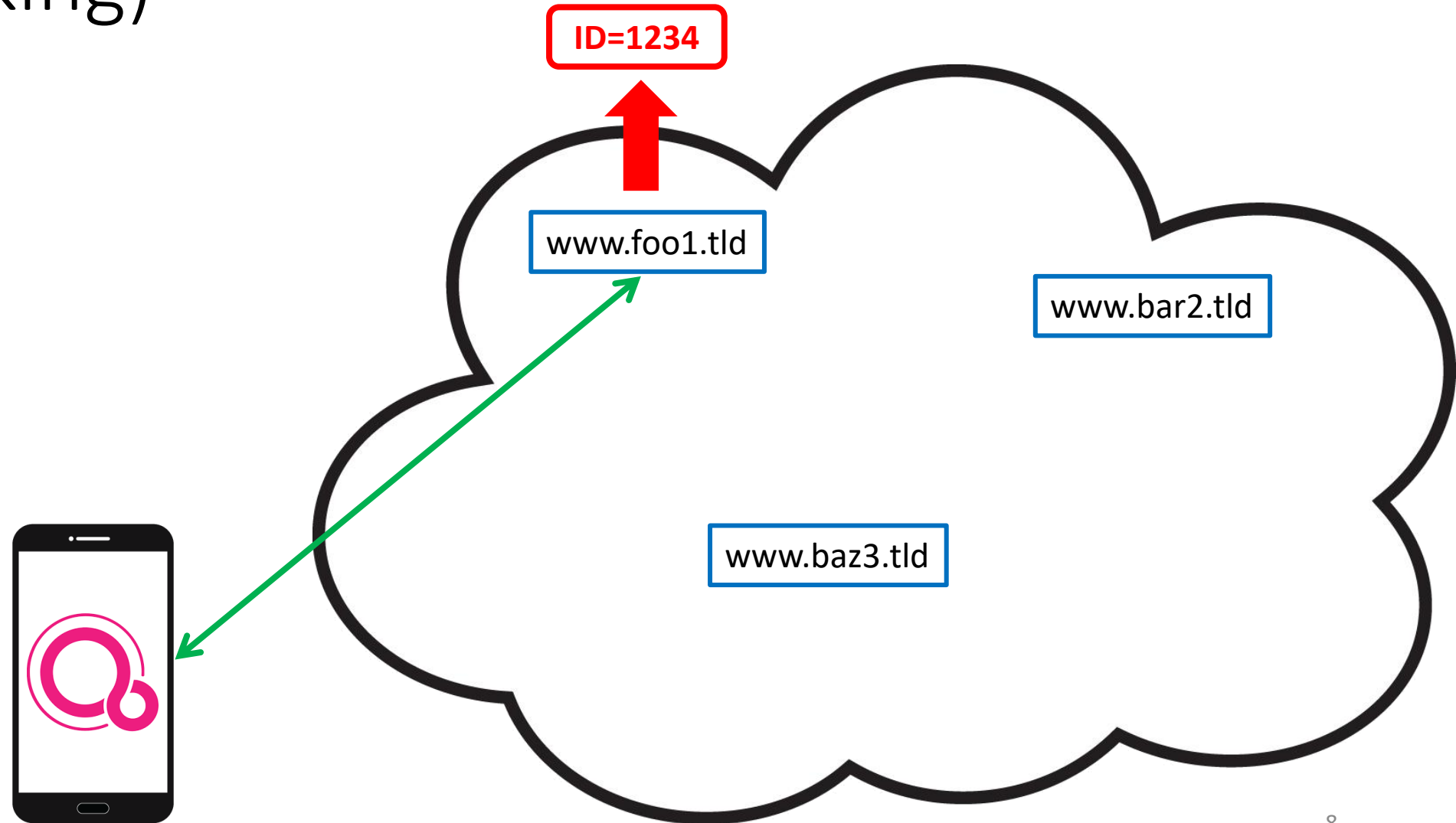
(subtract, compare and verify)

- Net result: a single (correct)  $seed$

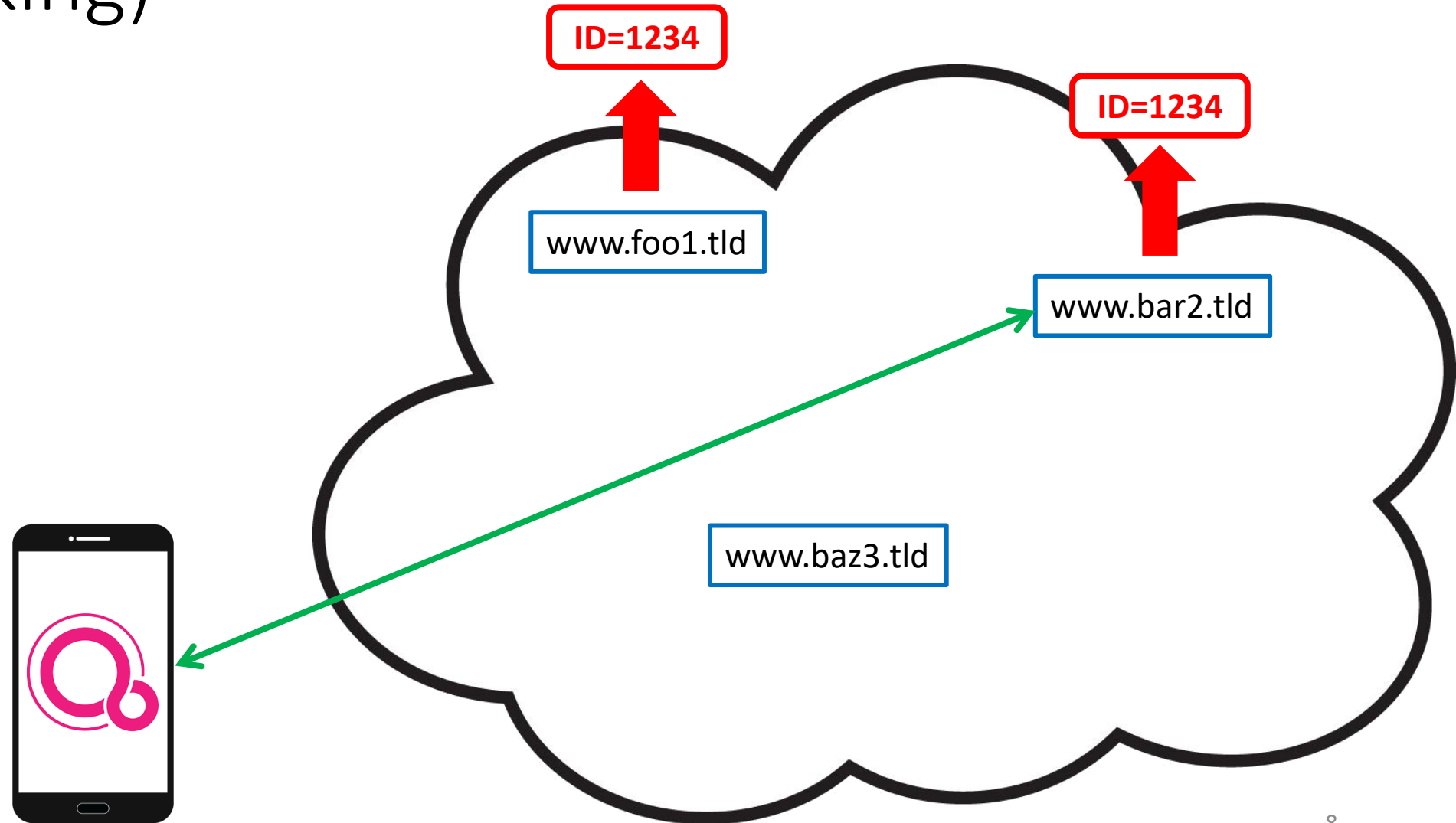
# Use Case: Web-Based Device Tracking (Cross-Site Tracking)



# Use Case: Web-Based Device Tracking (Cross-Site Tracking)

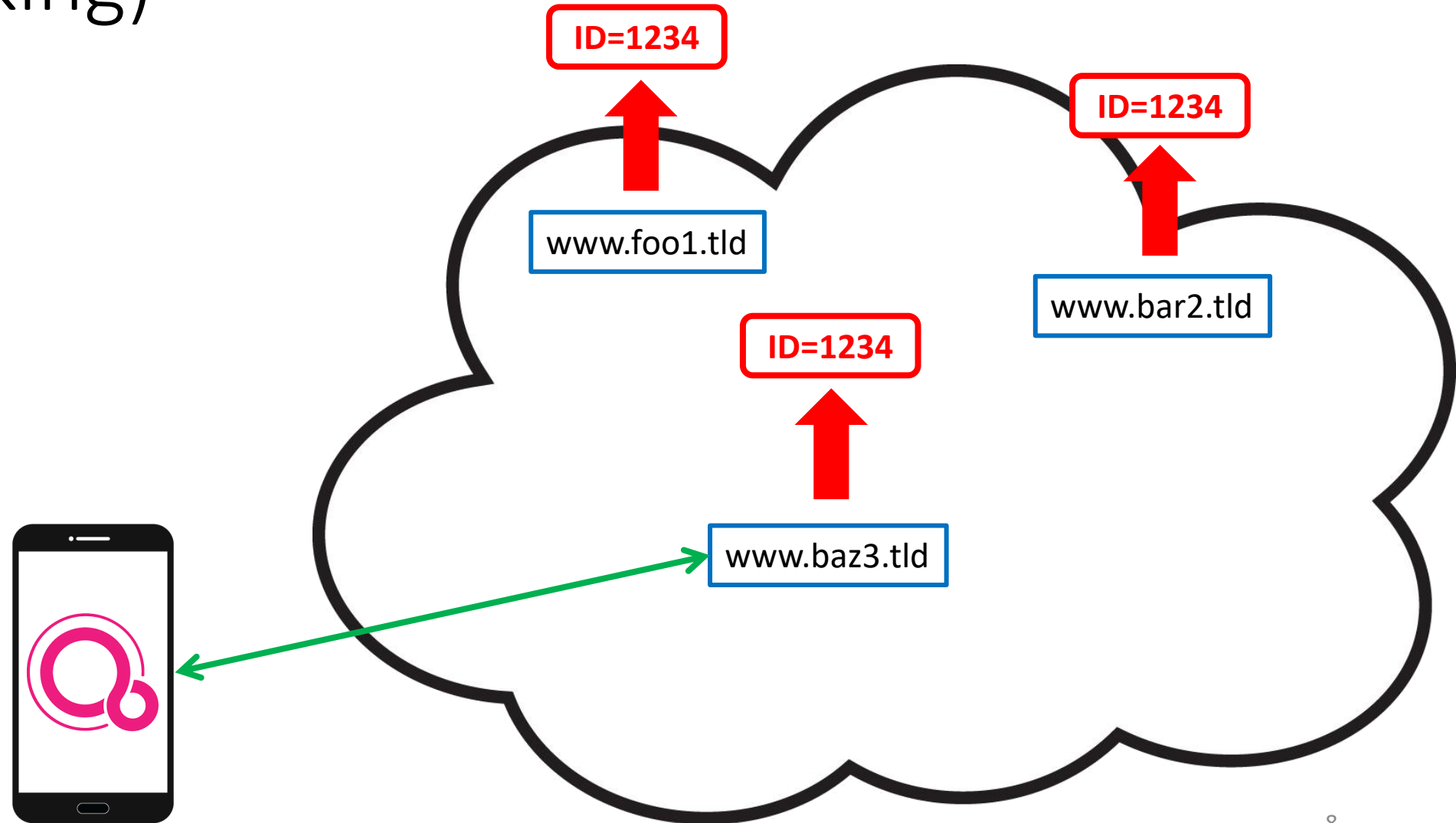


# Use Case: Web-Based Device Tracking (Cross-Site Tracking)



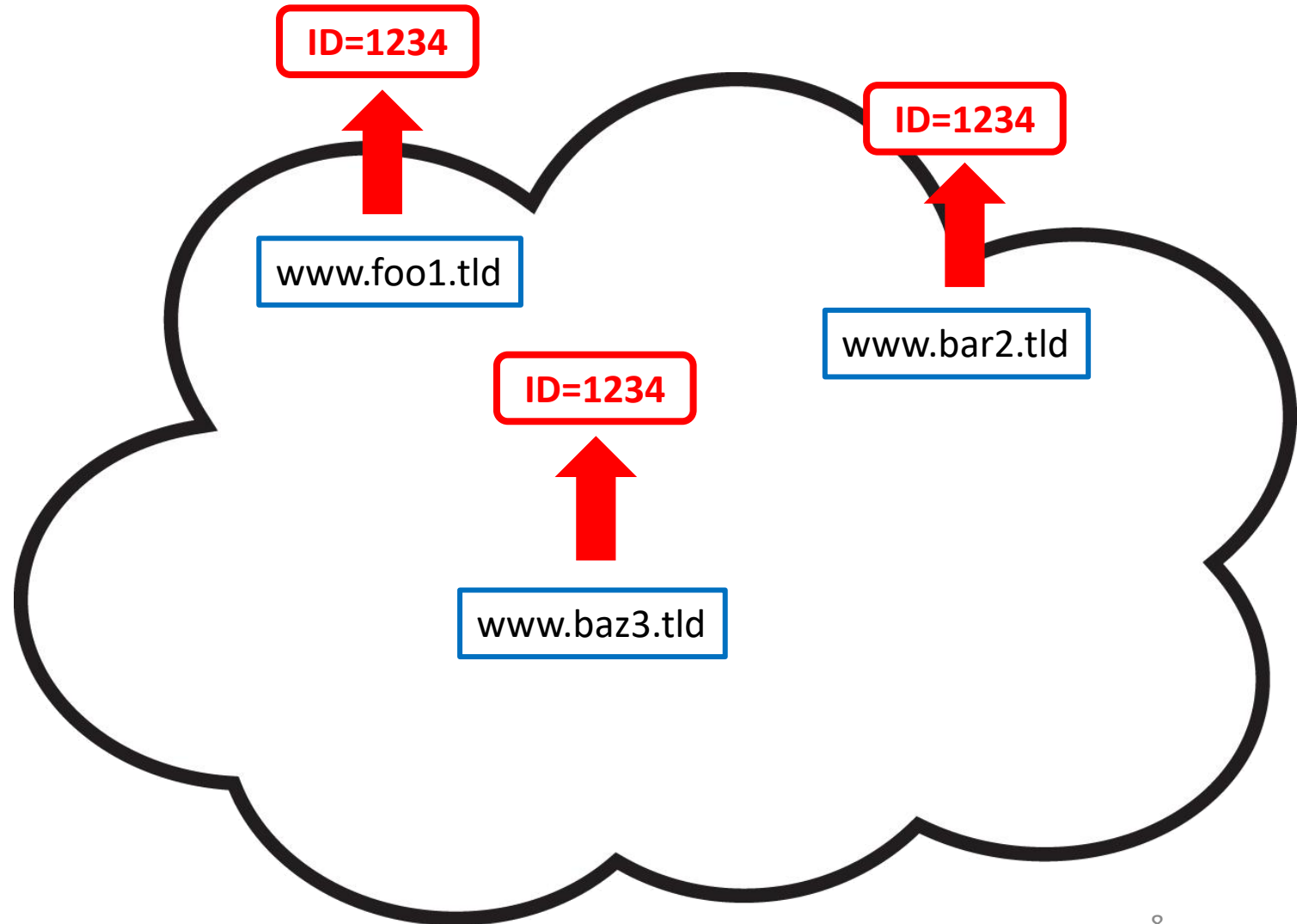


# Use Case: Web-Based Device Tracking (Cross-Site Tracking)



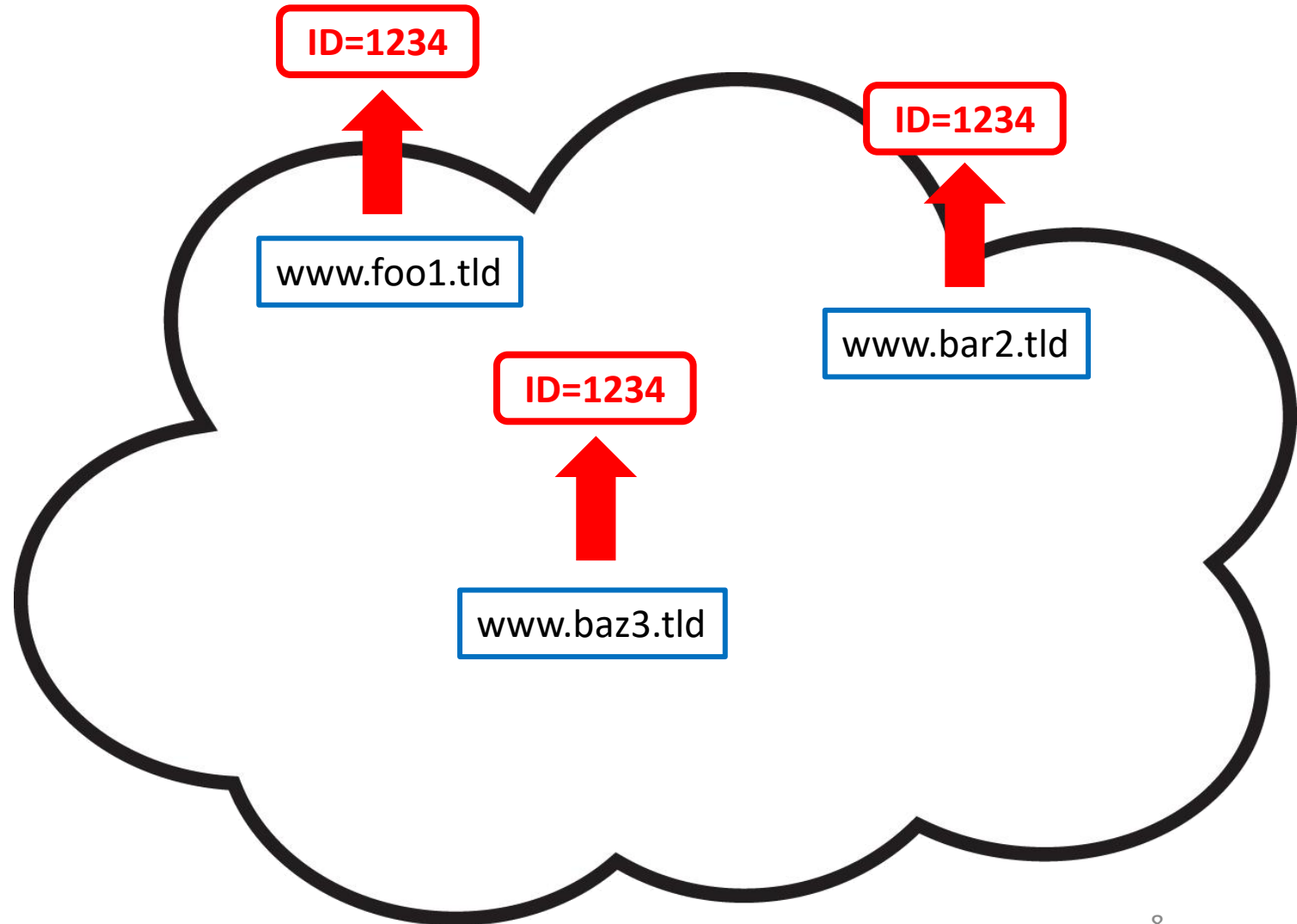
# Use Case: Web-Based Device Tracking (Cross-Site Tracking)

- *seed* is our device ID!



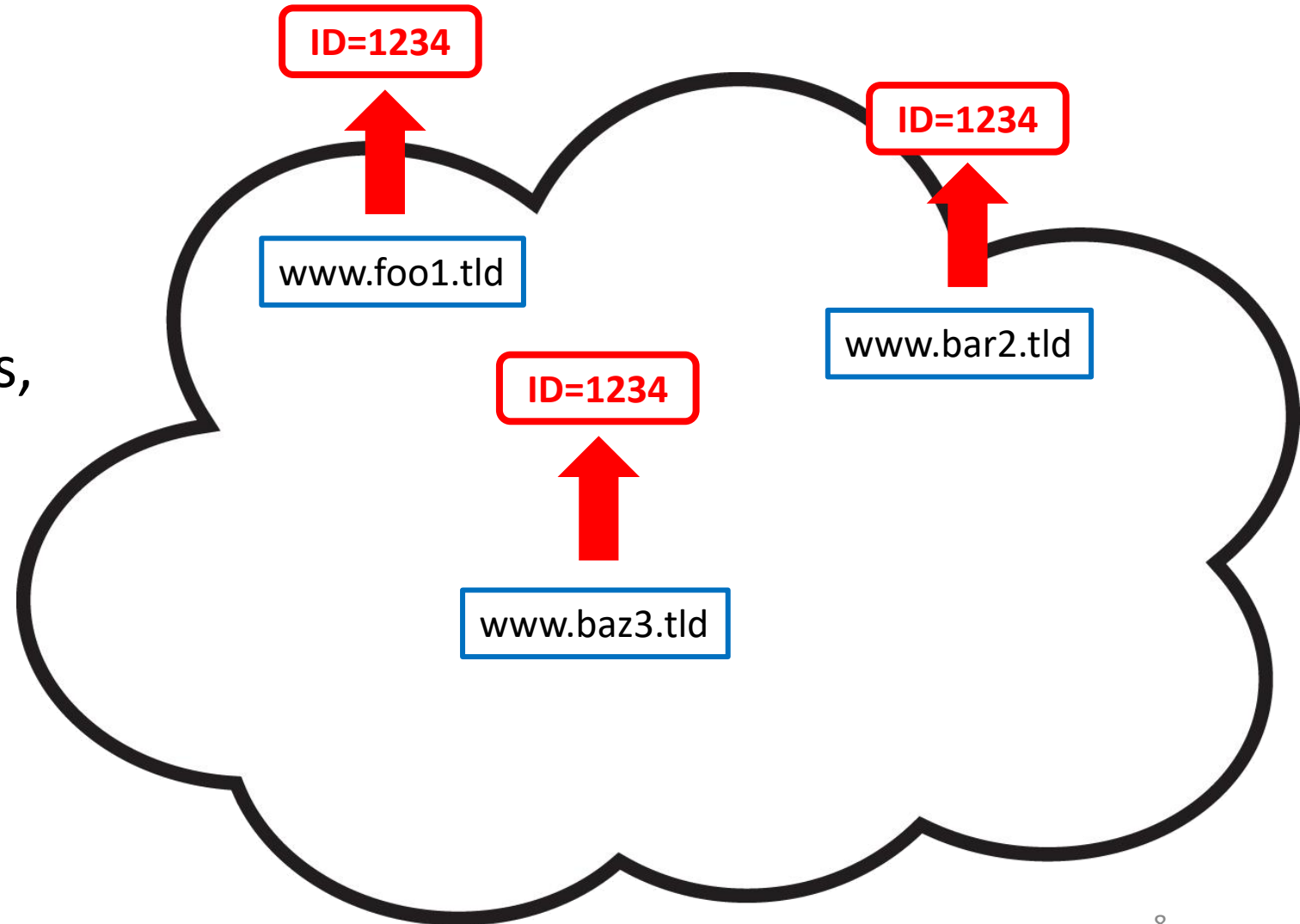
# Use Case: Web-Based Device Tracking (Cross-Site Tracking)

- *seed* is our device ID!
- 31 bit unique (up to the birthday paradox)



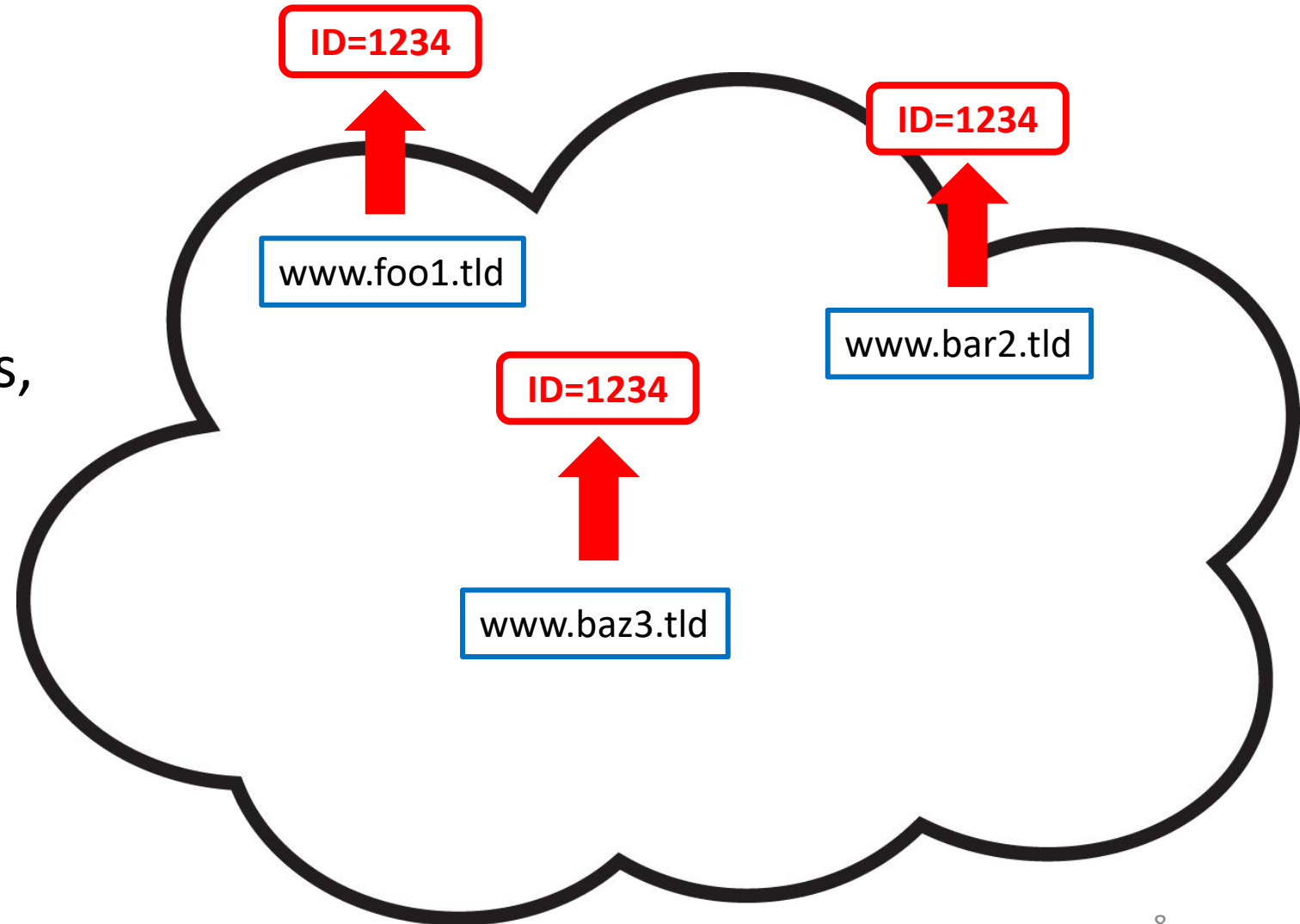
# Use Case: Web-Based Device Tracking (Cross-Site Tracking)

- *seed* is our device ID!
- 31 bit unique (up to the birthday paradox)
- Stable across sites, browsers, networks, privacy mode, ...



# Use Case: Web-Based Device Tracking (Cross-Site Tracking)

- *seed* is our device ID!
- 31 bit unique (up to the birthday paradox)
- Stable across sites, browsers, networks, privacy mode, ...
- Only re-generated on boot!



# TCP/IPv6 Attack Summary

# TCP/IPv6 Attack Summary

- We obtain:
  - PRNG *seed*
    - = Stable device ID (until next reboot) – **31 bits**
    - *key* → TCP TS secret
    - *key'* → TCP source port secret
    - *key''* → TCP ISN secret
    - UDP source port prediction
  - TCP outbound connection *counter mod 49536* (information leakage)
  - Machine time (*t*) since boot (information leakage)

# TCP/IPv6 Attack Summary

- We obtain:
  - PRNG *seed*
    - = Stable device ID (until next reboot) – **31 bits**
    - *key* → TCP TS secret
    - *key'* → TCP source port secret
    - *key''* → TCP ISN secret
    - UDP source port prediction
  - TCP outbound connection *counter mod 49536* (information leakage)
  - Machine time (*t*) since boot (information leakage)
- Runtime:
  - Few opcodes (20 table lookups in *Q*, 20 table lookups in *W*)



# TCP/IPv6 Attack Summary

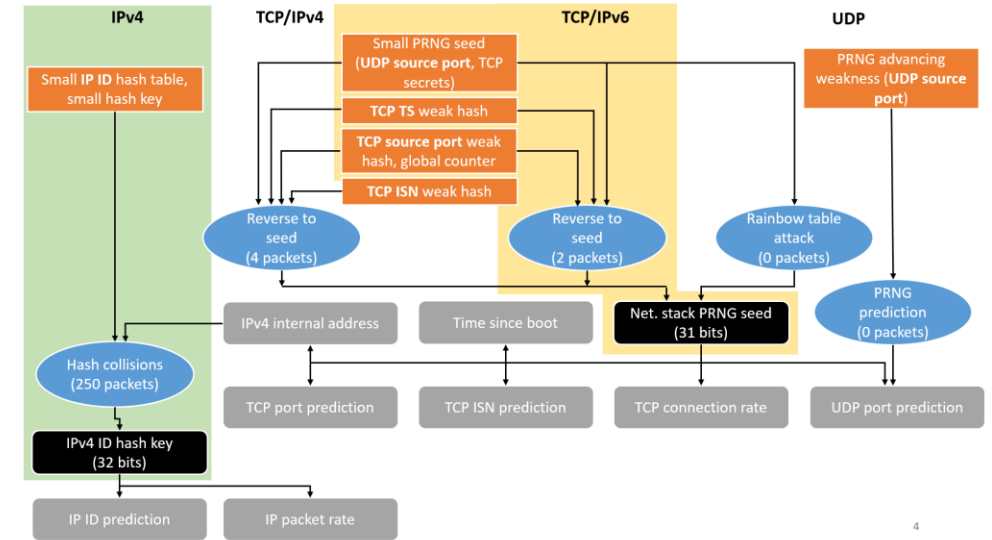
- We obtain:
  - PRNG *seed*
    - = Stable device ID (until next reboot) – **31 bits**
    - *key* → TCP TS secret
    - *key'* → TCP source port secret
    - *key''* → TCP ISN secret
    - UDP source port prediction
  - TCP outbound connection *counter mod* 49536 (information leakage)
  - Machine time (*t*) since boot (information leakage)
- Runtime:
  - Few opcodes (20 table lookups in *Q*, 20 table lookups in *W*)
- RAM: 56GiB ( $|Q| = 32GiB$ ,  $|W| = 24GiB$ )

# TCP/IPv6 Attack Summary

- We obtain:
  - PRNG *seed*
    - = Stable device ID (until next reboot) – **31 bits**
    - *key* → TCP TS secret
    - *key'* → TCP source port secret
    - *key''* → TCP ISN secret
    - UDP source port prediction
  - TCP outbound connection *counter mod* 49536 (information leakage)
  - Machine time (*t*) since boot (information leakage)
- Runtime:
  - Few opcodes (20 table lookups in *Q*, 20 table lookups in *W*)
- RAM: 56GiB ( $|Q| = 32GiB$ ,  $|W| = 24GiB$ )
- Dual stack? Using a TCP/**IPv4** SYN packet, we can obtain the IPv4 internal address

# TCP/IPv6 Attack Summary

- We obtain:
  - PRNG *seed*
    - = Stable device ID (until next reboot) – **31 bits**
    - *key* → TCP TS secret
    - *key'* → TCP source port secret
    - *key''* → TCP ISN secret
    - UDP source port prediction
  - TCP outbound connection *counter*  $\text{mod } 49536$  (information leakage)
  - Machine time (*t*) since boot (information leakage)
- Runtime:
  - Few opcodes (20 table lookups in *Q*, 20 table lookups in *W*)
- RAM: 56GiB ( $|Q| = 32\text{GiB}$ ,  $|W| = 24\text{GiB}$ )
- Dual stack? Using a TCP/**IPv4** SYN packet, we can obtain the IPv4 internal address
- Which enables the IPv4 ID attack – **32 more device ID bits**, etc.



# Experiments and Results (All Attacks)

- Google Nest Hub Max (smart home display+speaker)
- Google Pixelbook Go laptop
- Intel NUC mini-PC model NUC8BEH
- Virtual device (over QEMU for x64)

Table I. NETWORKS

Network Name	Technology	IPv4/IPv6 Support	TCP Source Port
Bezeq	VDSL	Both	Override
Eduroam (HUJI)	Fiber (DWDM)	IPv4 Only	Intact
Triple C	VDSL	IPv4 Only	Override
Bezeq Fiber	Fiber	Both	Intact
Hot Cable	Cable	IPv4 Only	Override
Golan Telecom	Cellular	Both	Override
Partner	Cellular	IPv4 Only	Override

Table II. EXPERIMENT DESCRIPTIONS

Attack/Vulnerability	Paper Section	Attack Object	Bits	Packets	Dwell Time [ms]	Compute Time [ms]	Additional Data Exposed
TCP fields (IPv4)	§ IV-B	PRNG Seed	31	4	7 (avg) 18 (max)	2937 (avg) 5776 (max)	Private IP address TCP connection counter
TCP fields (IPv6)	§ IV-C	PRNG Seed	31	2	2 (avg) 3 (max)	0.5 (avg) 1 (max)	TCP connection counter
UDP source port	§ V	Next source port	15.6	0	negligible	0.5 (avg) 1 (max)	
IPv4 ID (straightforward)	§ VI-A	<i>hashIV</i>	32	250	116 (avg) 170 (max)	5397 (avg) 5464 (max)	
IPv4 ID (independent)	§ VI-B	<i>hashIV</i>	32	Thousands	59109 (avg) 73075 (max)	20762 (avg) 20775 (max)	Private IP address

# Root Causes, Recommendations and Fixes

Root Cause	Affected Fields	Recommendation	Fix
<b>Weak hash function</b> <ul style="list-style-type: none"><li>• byte-by-byte reversible</li><li>• small state (32 bits)</li></ul>	TCP TS TCP ISN TCP source port	Cryptographic Hash	CVE 2024-10026
<b>Weak PRNG</b> <ul style="list-style-type: none"><li>• small effective seed space (31 bits)</li><li>• weak advancement algorithm</li></ul>	TCP secrets ( <i>key</i> , <i>key'</i> , <i>key''</i> ) UDP source port	Cryptographic PRNG	CVE-2024-10603 CVE-2024-10604
<b>Global counter</b>	TCP source port	Fully random TCP source port	CVE-2024-10603 CVE-2024-10604
<b>Weak ID generation scheme</b> <ul style="list-style-type: none"><li>• small hashing key space</li><li>• small hash table size</li><li>• deterministic update scheme for table cells (“++”))</li></ul>	IPv4 ID IPv6 ID	Fully random IP ID	CVE-2024-10603

# Summary

# Summary

- Holistic approach – the whole is bigger than its parts
  - “Smaller” vulnerabilities in separate functionalities/components, and across layers, used in concert to mount powerful attacks

# Summary

- Holistic approach – the whole is bigger than its parts
  - “Smaller” vulnerabilities in separate functionalities/components, and across layers, used in concert to mount powerful attacks
- Clean slate design  $\neq$  secure code base
  - The design may be clean and good – but security is in the details ;-)
  - Not learning from other (mature) kernel mistakes



# Summary

- Holistic approach – the whole is bigger than its parts
  - “Smaller” vulnerabilities in separate functionalities/components, and across layers, used in concert to mount powerful attacks
- Clean slate design  $\neq$  secure code base
  - The design may be clean and good – but security is in the details ;-)
  - Not learning from other (mature) kernel mistakes
- Device tracking via network stack objects (a concept from our recent papers):
  - Find a vulnerable protocol header field
  - Extract the key/seed used by the kernel to generate it
  - This key/seed is the device fingerprint/ID

# Thank you!

Questions?