601.445/601.645: Side Channels 2

Side Channel Attacks

Instructor: Matthew Green

Housekeeping:

- Midterm Weds
- This week, the weekly assignment will be the midterm
- New weekly assignment on Monday
- Programming Assignment 3 out next Weds

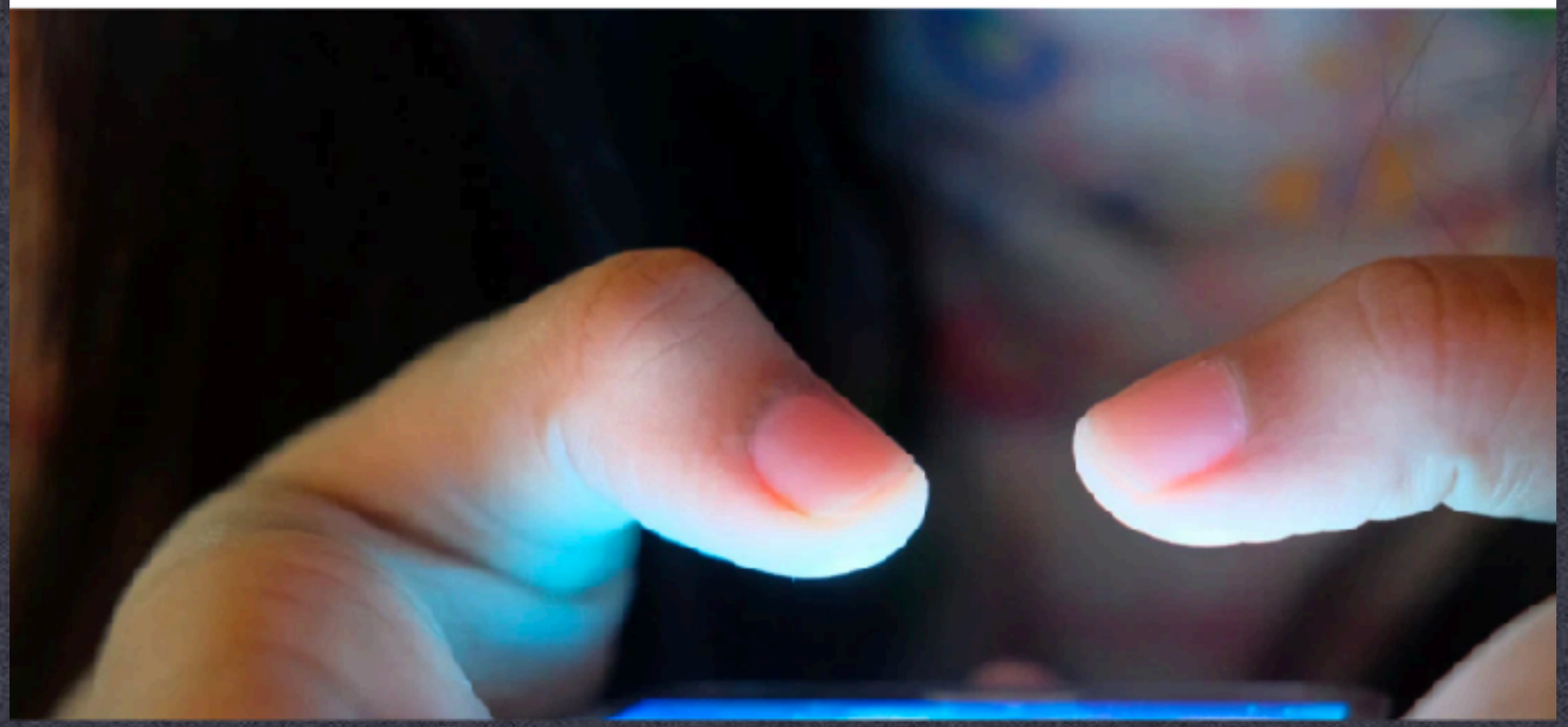
Current Events

Current Events

Israel passes emergency law to use mobile data for COVID-19 contact tracing

Natasha Lomas @riptari / 6:25 am EDT • March 18, 2020

Com



Current Events

- How do we get smartphone location data?

Model 1: Tower data

- How do we get smartphone location data?
  - Some combination of cell tower triangulation & GPS/A-GPS (GPS with an assist from towers)
  - In U.S. we have E-911 laws that require A-GPS "pings" in selected circumstances
  - OS makers (e.g., Apple, Google) collect WiFi SSIDs, Bluetooth "beacon" signals

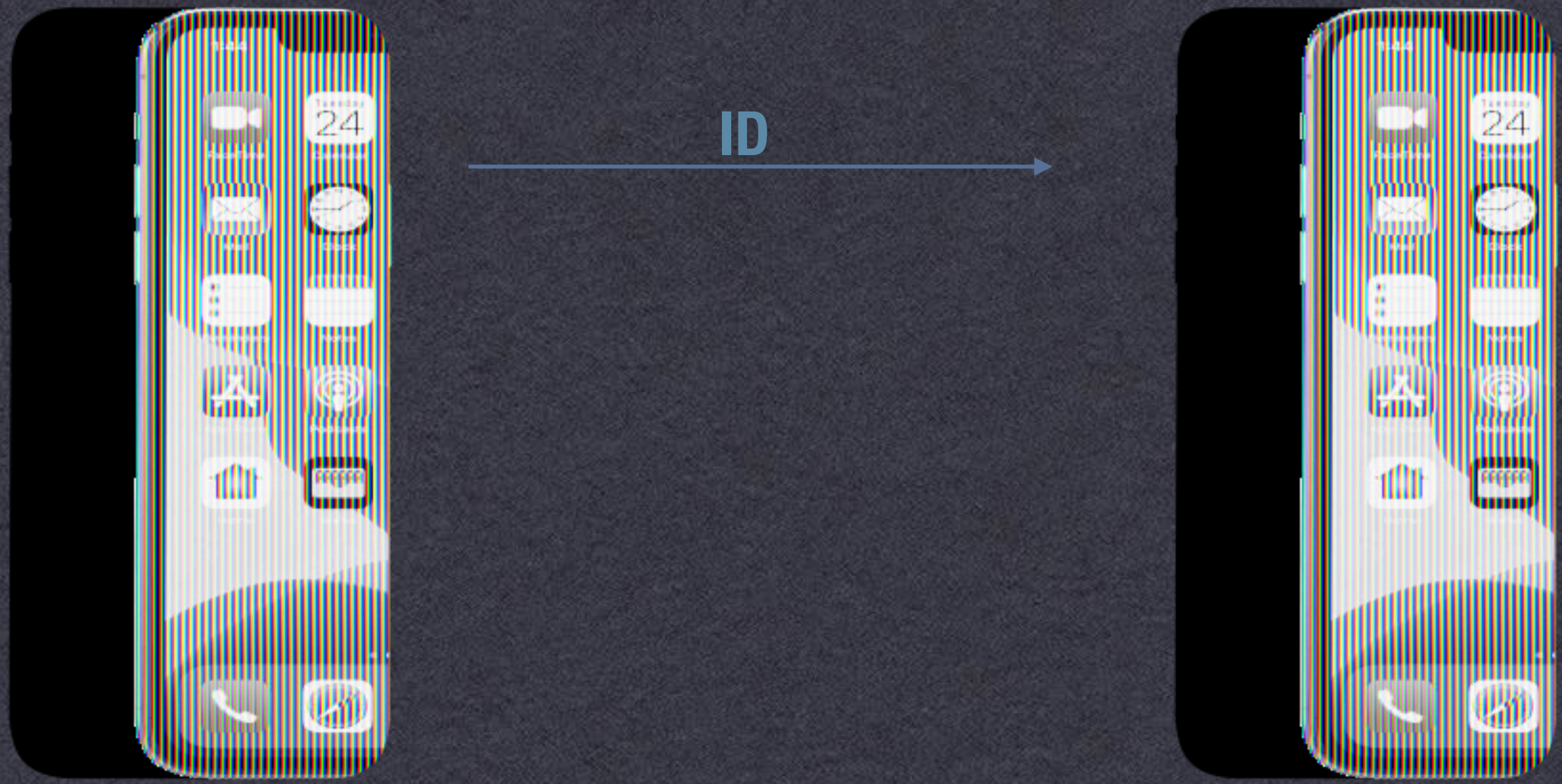
    (Big privacy questions around this)

Model 2: Proximity Data

- **Use Bluetooth beacons to measure proximity between phones**

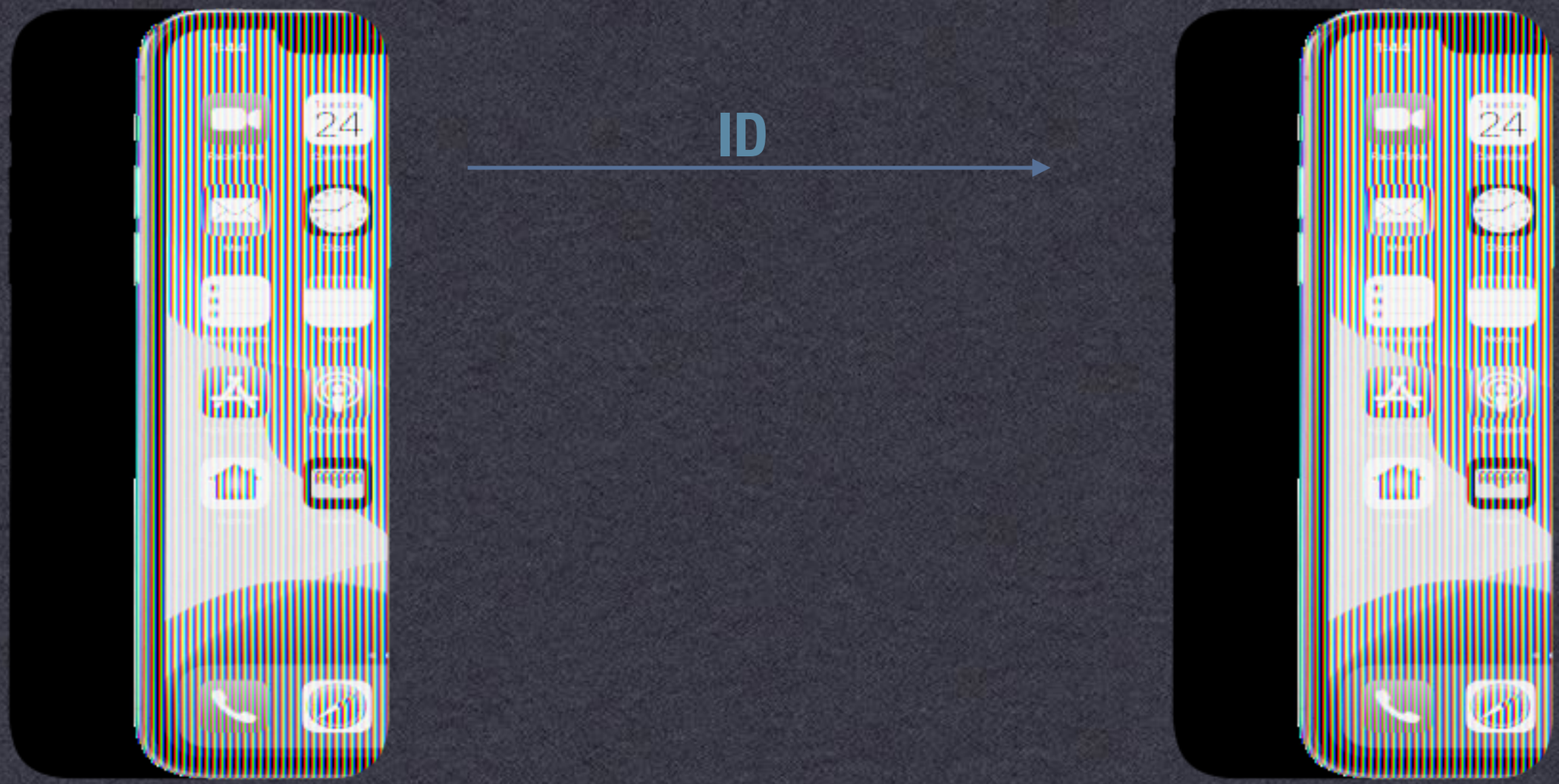
Model 2: Proximity Data

- **Use Bluetooth beacons to measure proximity between phones**



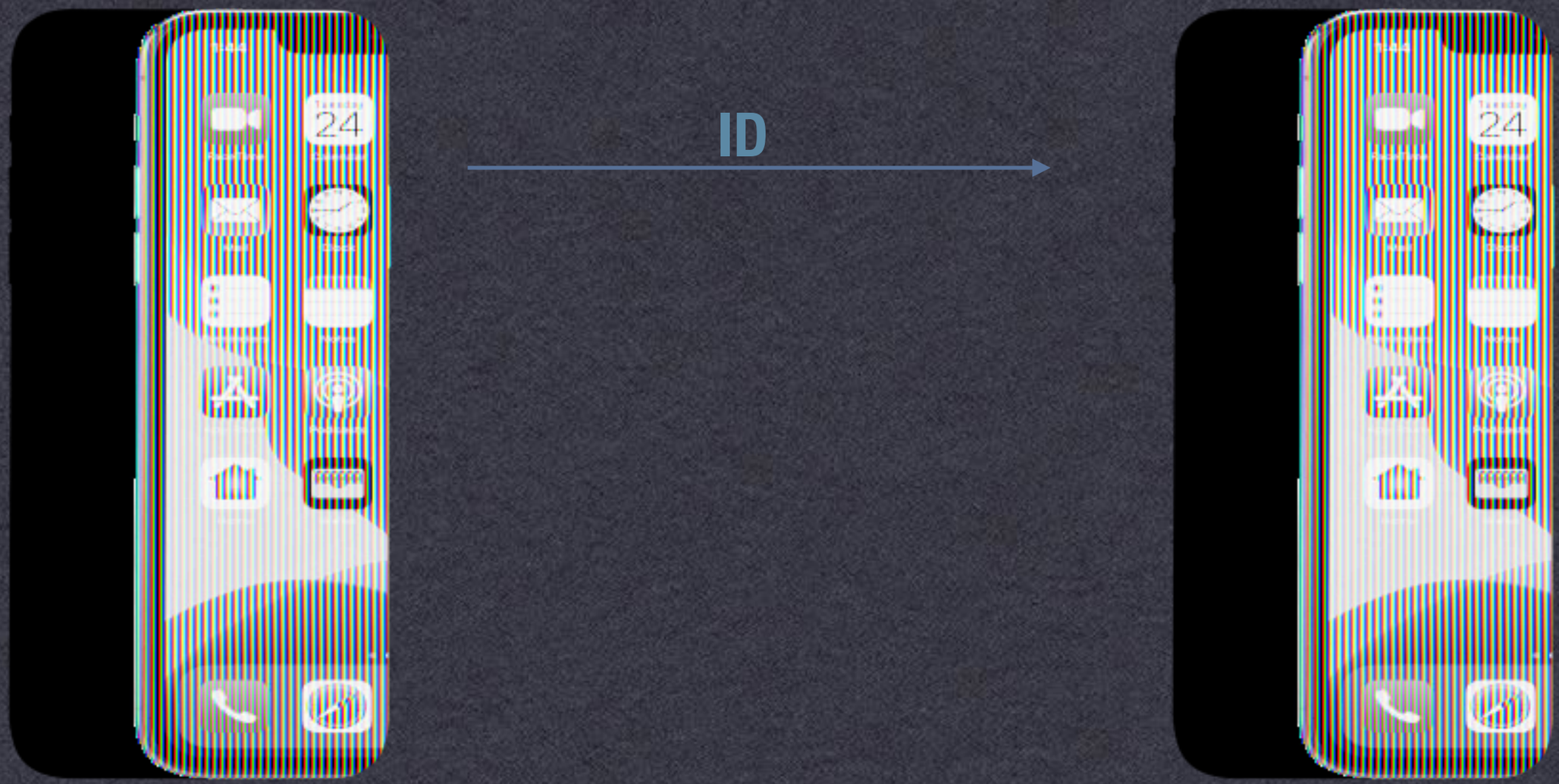
Model 2: Proximity Data

- Each observed identifier is stored <u>encrypted</u> on the receiving phone



Model 2: Proximity Data

- If a contact tests positive, the identifier is broadcast to all users



Question

- How do we solve the privacy problems?
  - Cell provider version (hard)
  - Contact tracing version (possible)

Question

- How do we solve the privacy problems?
  - <u>For cell providers</u>
    this is fundamentally hard
  - Every phone has an IMSI (identifier)
    (Newer 5G specs add a temporary identifier for roaming only)
  - Provider basically knows roughly where you are at all times

    (But there are multiple providers)

Set Intersection

ID	Loc		
ID1	L1		
ID2	L4		
ID3	L3		
ID3	L6		

AT&T

T-M

ID	Loc		
ID1	L1		
ID9	L2		
ID3	L3		
ID8	L4		

Sprint

ID	Loc		
ID1	L1		
ID2	L2		
ID3	L3		
ID4	L4		

Private Set Intersection

ID	Loc		
ID1	L1		
ID2	L4		
ID3	L3		
ID3	L6		

AT&T

T-M

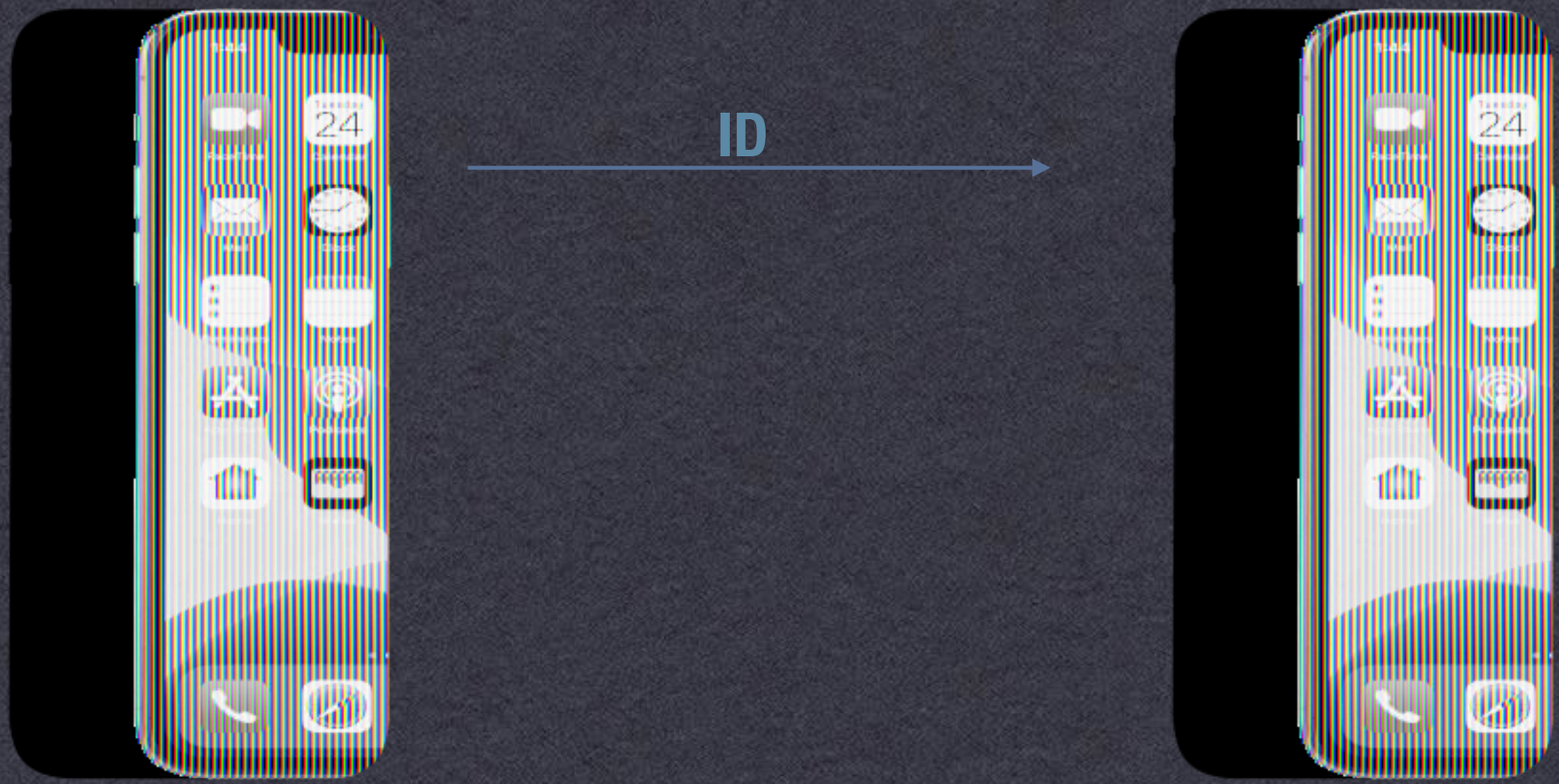
ID	Loc		
ID1	L1		
ID9	L2		
ID3	L3		
ID8	L4		

Sprint

ID	Loc		
ID1	L1		
ID2	L2		
ID3	L3		
ID4	L4		

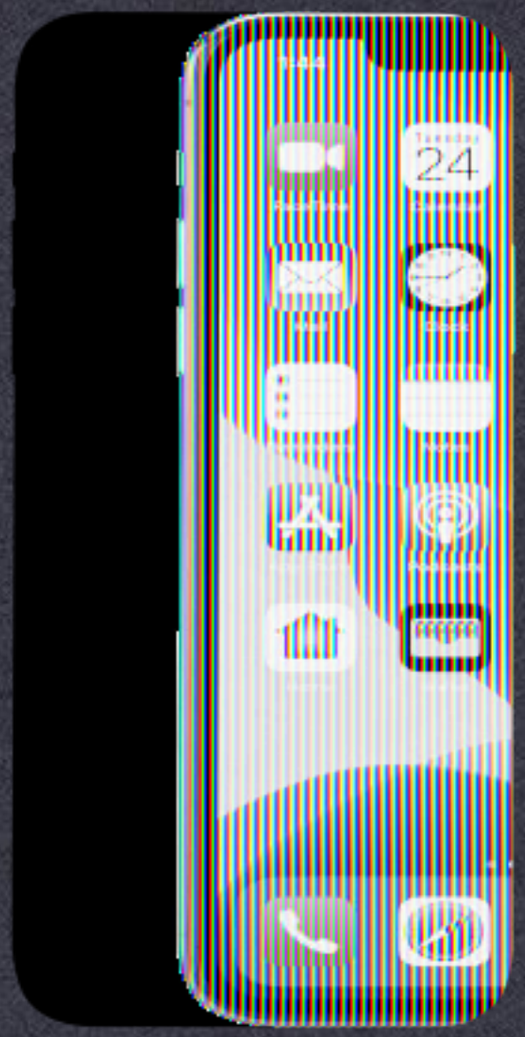
Model 2: Proximity Data

- How do we deal with the tracking risk here?

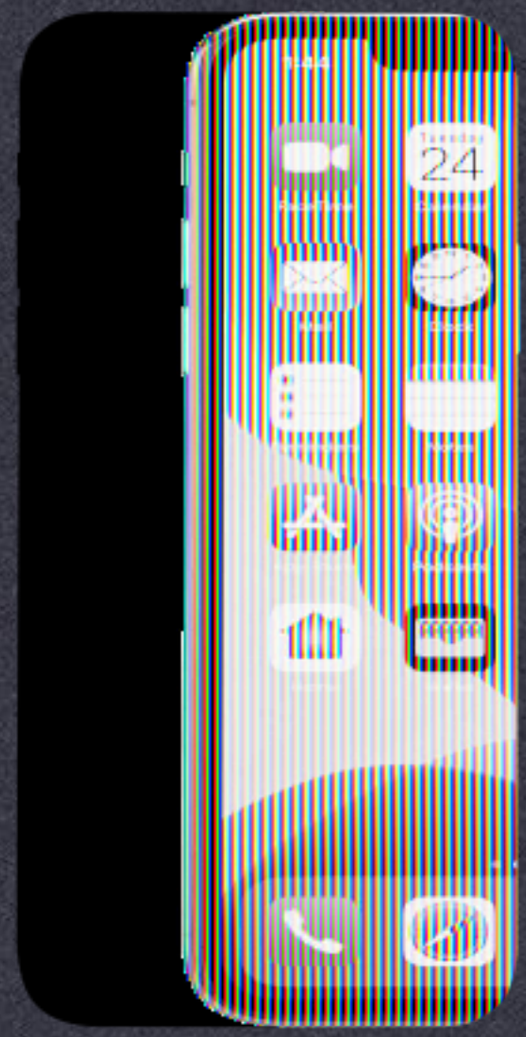


Apple: FindMy

- Constantly changing identifiers, changes every few minutes



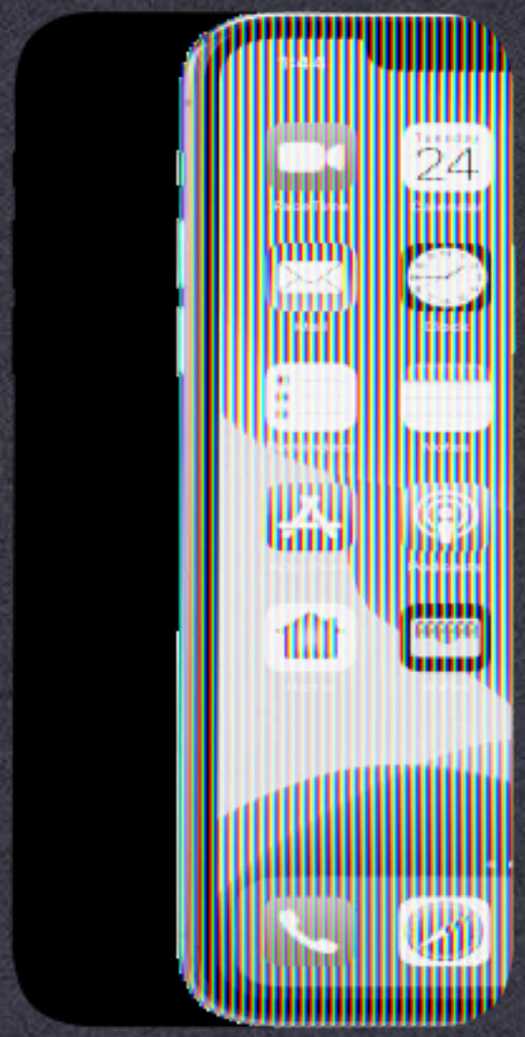
Pseudonym



Generated from a single “seed”, Changes every five minutes

Apple: FindMy

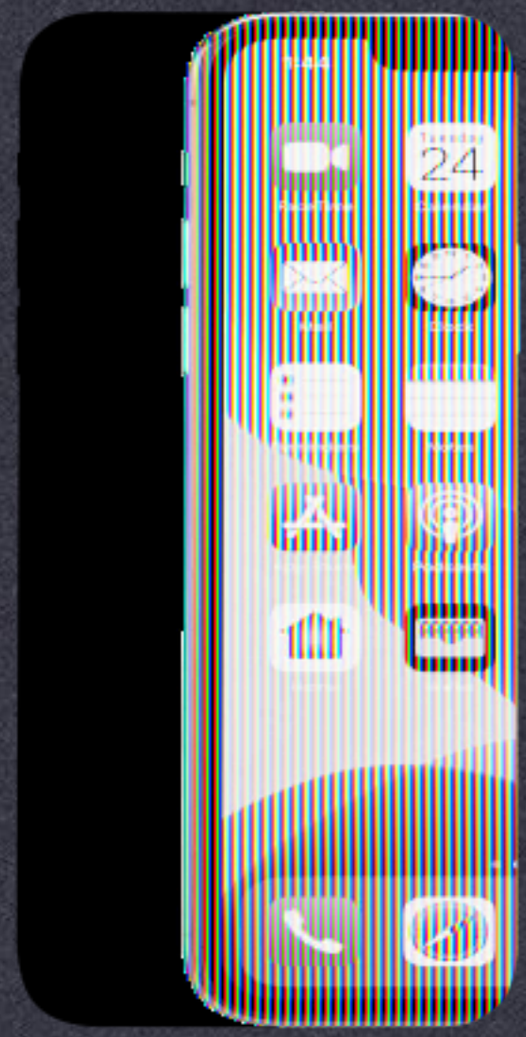
- **Constantly changing identifiers, changes every few minutes**



Pseudonym

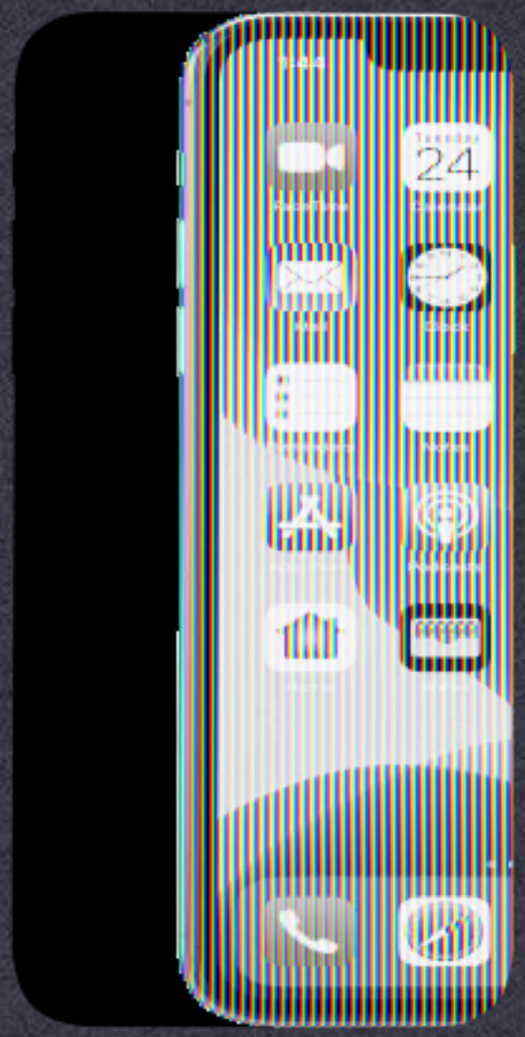
(pk_pseudo)

Generate a separate public/secret encryption keypair for each pseudonym



Apple: FindMy

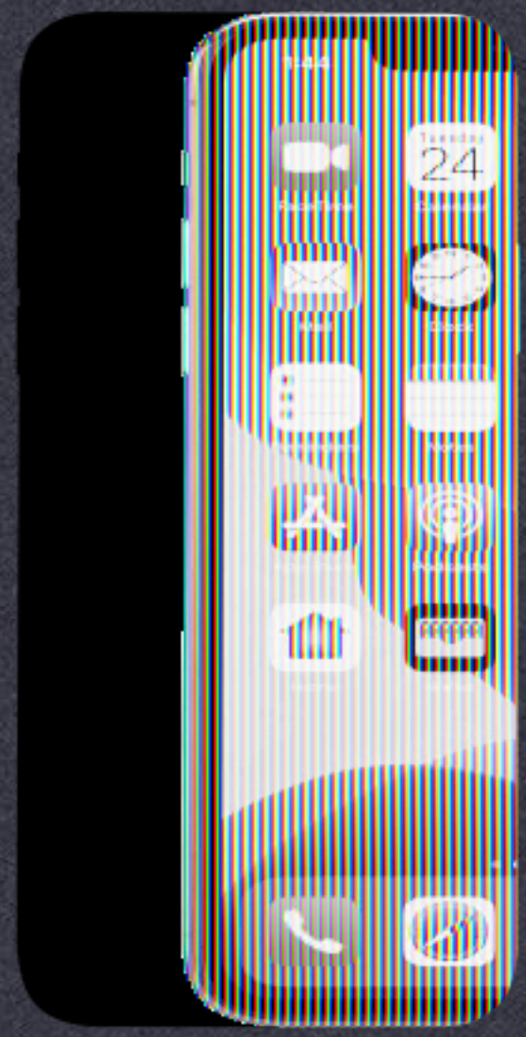
- Receiver can encrypt a location under public key, send to cloud



Pseudonym
(pk_pseudo)

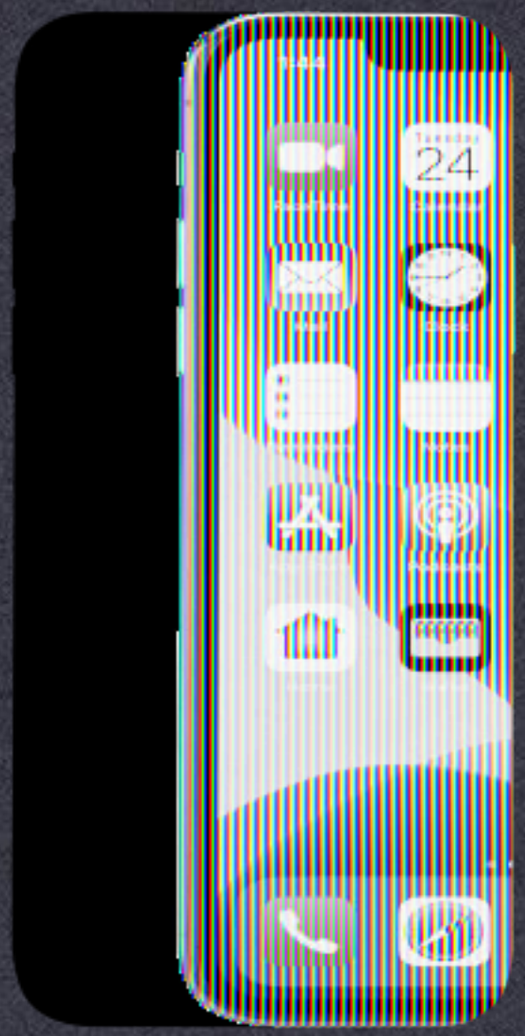
Phone can encrypt
Its location and identifier
under public key

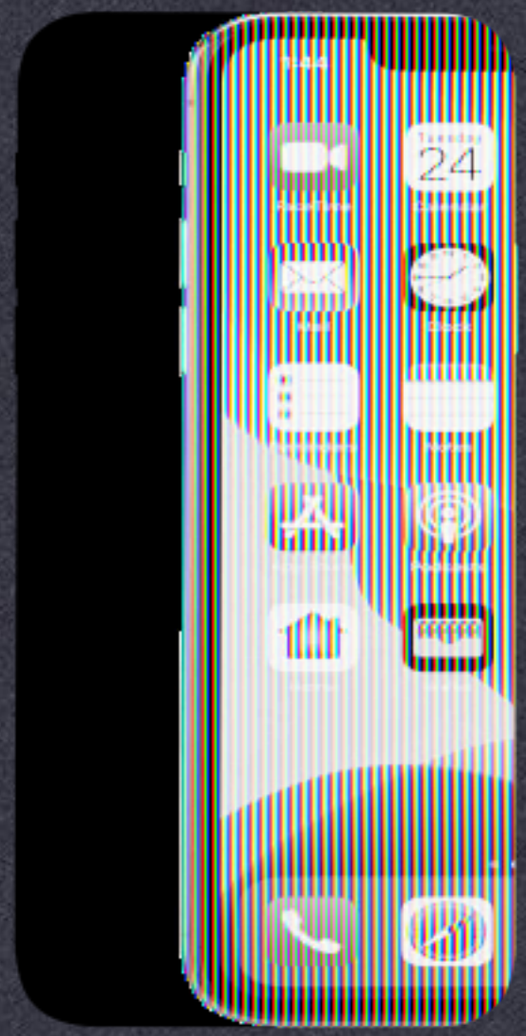
Send ciphertext to cloud



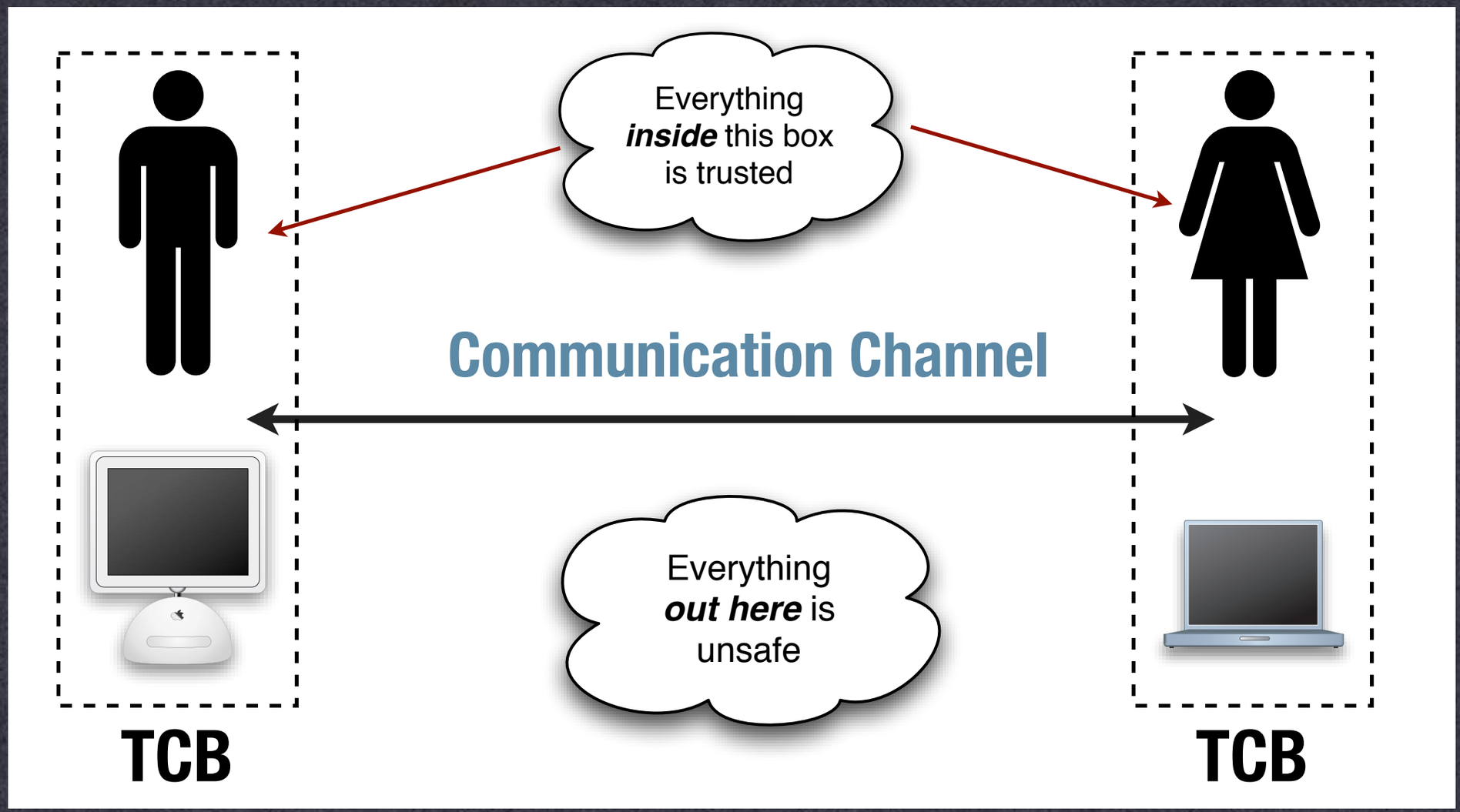
Apple: FindMy

- **Reveal seed, secret keys, allows discovery and decryption of all locations/IDs**

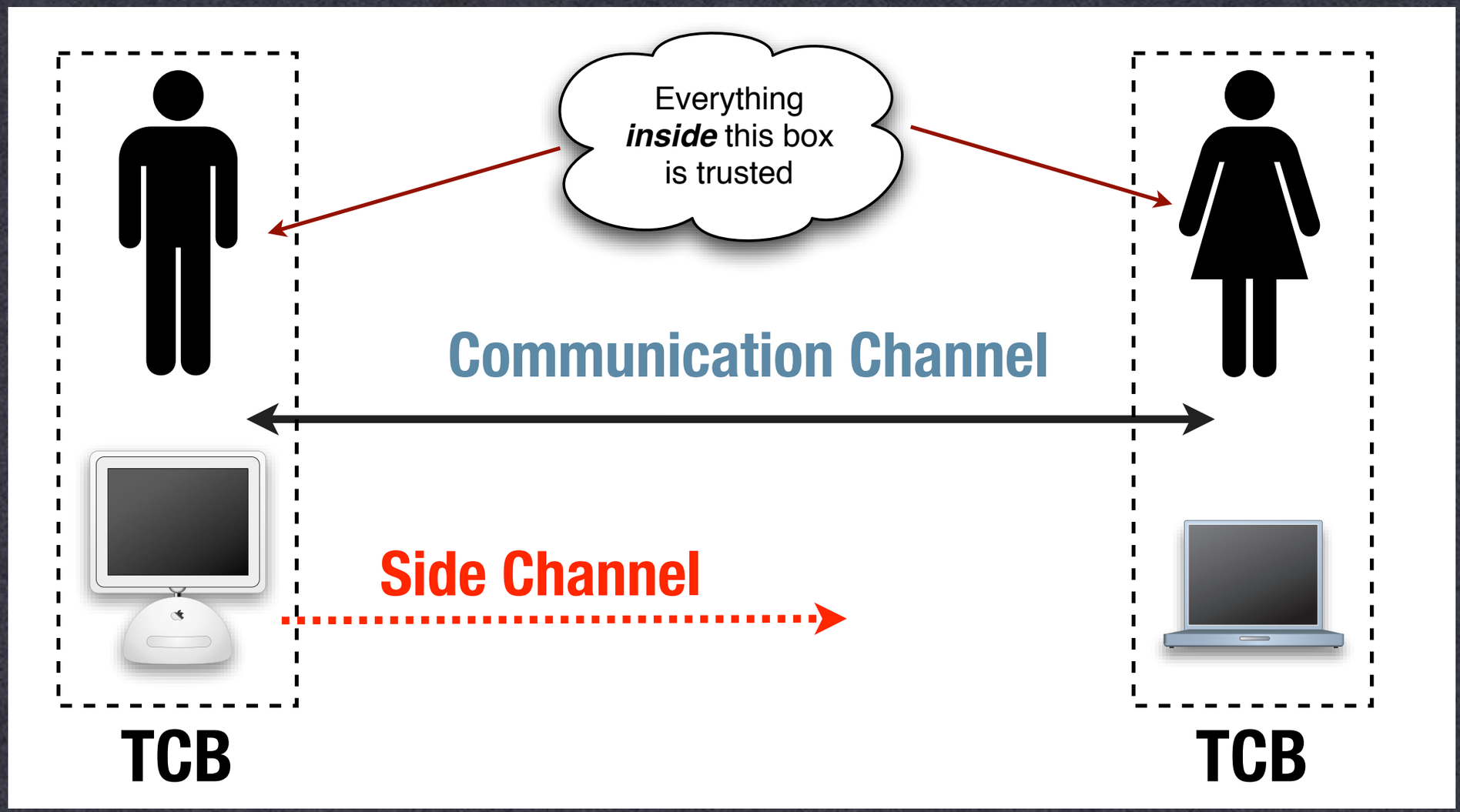




Review


Everything
inside this box
is trusted
Communication Channel
Everything
out here is
unsafe
TCB
TCB

Side Channels



Today

- A new class of side-channel attacks
  - Not specific to cryptography
  - But worth mentioning because they are so powerful

Today

- A new class of side-channel attacks
  - Not specific to cryptography
  - But worth mentioning because they are so powerful
  - Spectre, Meltdown, Foreshadow

How do we make CPUs faster?

- Basically all of the hardware gains are used up
- Today, we get major performance from parallelization and optimization: reduce memory latency, pipelining, speculative execution

How do we make CPUs faster?

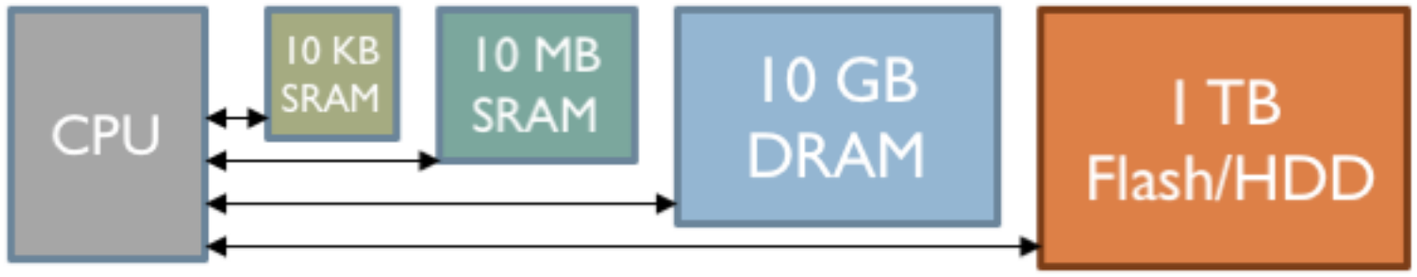
- Basically all of the hardware gains are used up
- Today, we get major performance from parallelization and optimization: reduce memory latency, pipelining, speculative execution

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Memory Hierarchy Interface

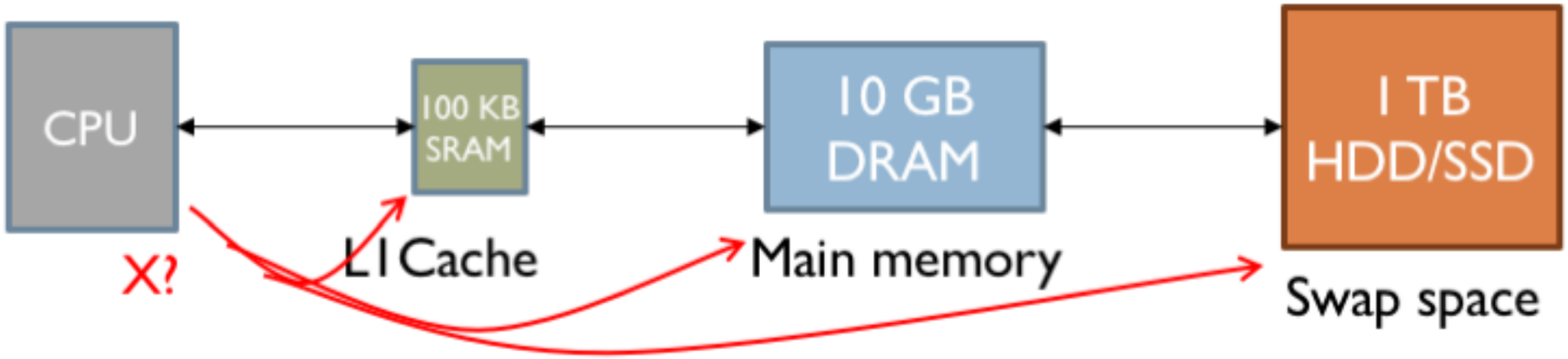
Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: "Use them cleverly"



Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns



From: https://computationstructures.org/lectures/caches/caches.html#15

Consider this program

- What do we do if x is not in cache yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

Consider this program

- What do we do if x is not in cache yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

slow solution: wait until x loads from DRAM

Consider this program

- What do we do if x is not in memory yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

fast solution: speculatively execute both branches (or one guess), discard invalid work

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attack scenario:

- Code runs in a trusted context
- Adversary wants to read memory and controls unsigned integer `x`
- Branch predictor will expect `if()` to be true (e.g. because prior calls had `x < array1_size`)
- `array1_size` and `array2[]` are not in cache

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:

8 bytes of data (value doesn't matter)

[... lots of memory up to `array1` base+N...]

`09 F1 98 CC 90...` (something secret)

```
array2[ 0*512]
array2[ 1*512]
array2[ 2*512]
array2[ 3*512]
array2[ 4*512]
array2[ 5*512]
array2[ 6*512]
array2[ 7*512]
array2[ 8*512]
array2[ 9*512]
array2[10*512]
array2[11*512]
  ...
```

Contents don't matter

only care about cache ***status***

Uncached Cached

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides

https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attacker calls victim code with `x`=N (where N > 8)

- Speculative exec while waiting for `array1_size`
  - Predict that if() is true
  - Read address (`array1` base + `x`) w/ out-of-bounds `x`
  - Read returns secret byte = **09** (fast – in cache)
  - Request memory at (`array2` base + **09***512)
  - Brings `array2[`**09**`*512]` into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker times reads from `array2[i*512]`

- Read for `i`=**09** is fast (cached), revealing secret byte

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[... lots of memory up to `array1` base+N...]
09 F1 98 CC 90... (something secret)

```
array2[ 0*512]
array2[ 1*512]
array2[ 2*512]
array2[ 3*512]
array2[ 4*512]
array2[ 5*512]
array2[ 6*512]
array2[ 7*512]
array2[ 8*512]
array2[ 9*512]
array2[10*512]
array2[11*512]
  ...
```

Contents don't matter
only care about cache ***status***

Uncached Cached

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Why does this matter?

- We can learn information about data that we should not be able to access
- For example:
  - Kernel secrets
  - Secrets in the same process (keys)
  - Other applications
  - It <u>should</u> run user space code
    But attacker can cause it to leak information about data in the kernel

Intel SGX

- "Virtual trusted hardware" inside of an Intel CPU
- Basic ideas:
  - To run an SGX program ("enclave") turn in an isolation mode that is even stronger than Ring0
  - OS cannot read enclave memory, enclave cannot read OS/app memory
  - Enforced by microcode

What software can we run?

- Enclave software has no access to I/O (network, keyboard, mouse). It has to interface with a real application via a strict API
- Every SGX (“enclave program”) is hashed, then digitally signed under a certificate chain held by Intel

How does the enclave store data persistently?

- Enclave can’t access disks directly
- It has to send data out to an application via the API
- But if that program gets compromised, how does the enclave verify it’s safe?

How does the enclave store data persistently?

- Answer: lots of crypto
- Ever enclave gets its own secret encryption keys (generated by SGX base OS)
- Keys are generated as a combination of:
  Enclave software hash +
  System unique identifier +
  A system root secret key known to SGX

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots
- What about replay attacks?

Intel SGX attestation

- This all works great when I'm running software on my own computer
- What happens if I want to use Intel SGX in the cloud?
- How do I know my software is really being operated by SGX and not some malicious software pretending to be SGX?

Solution: attestation

- Every Intel processor has a secret signing key baked in at the factory*
- An enclave can ask the SGX OS to sign any block of data (“attestation”)
- Signature includes:
  - Enclave hash (enforced by SGX OS)
  - Arbitrary data blob specified by application
  - Any other measurements, e.g., RAM hash
  - Certificate signed by Intel

So TL;DR

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

Foreshadow

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

FORESHADOW

Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Read the paper | Cite | Watch a demo

Introduction

Foreshadow is a speculative execution attack on Intel processors which allows an attacker to steal sensitive information stored inside personal computers or third party clouds. Foreshadow has two versions, the original attack designed to extract data from SGX enclaves and a Next-Generation version which affects Virtual Machines (VMs), hypervisors (VMM), operating system (OS) kernel memory, and System Management Mode (SMM) memory.

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine Intel attestation that can be verified against Intel Attestation Service (IAS).

8 23 50

Show this thread

Replies

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

Here is your attestation that "RT: Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine " is a genuine SGX enclave github.com/TeeAaas/Foresh...

1 1