Secure Messaging
(and current attacks against it)

Håck ma's Castle, 2024-08-29 20:00 (UTC+2), Schloss Ottenschlag

Univ.-Prof. Dr. **René Mayrhofer**
Institute of Networks and Security, LIT Secure and Correct Systems Lab & CDL Digidow

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

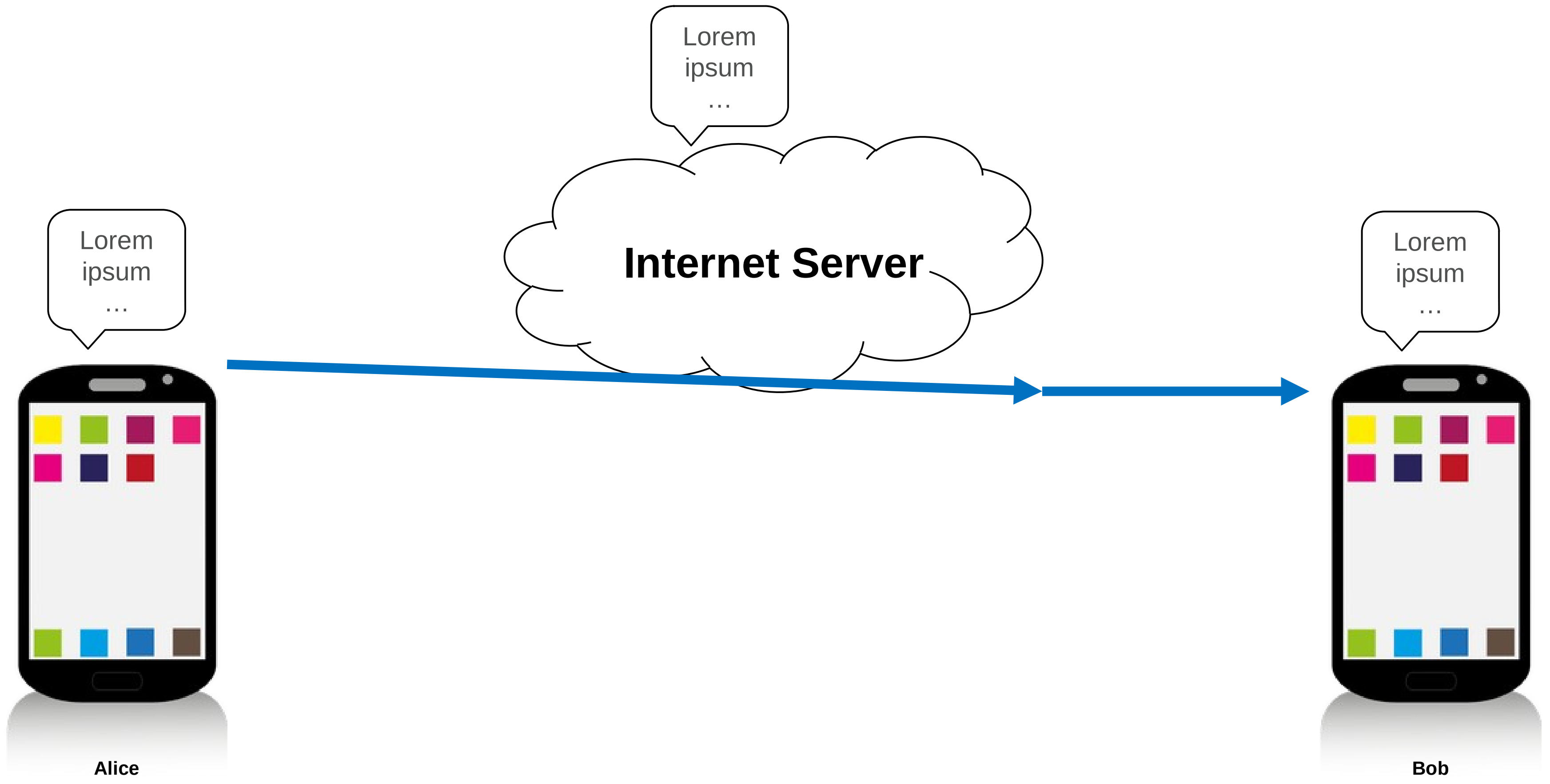
So you are concerned about bad content in social media / messenger apps?

Are you also concerned about “bad” content communicated in private, face-to-face?

The definition of “bad” depends on the policy of the day, and can change quickly with (or without) a single election…

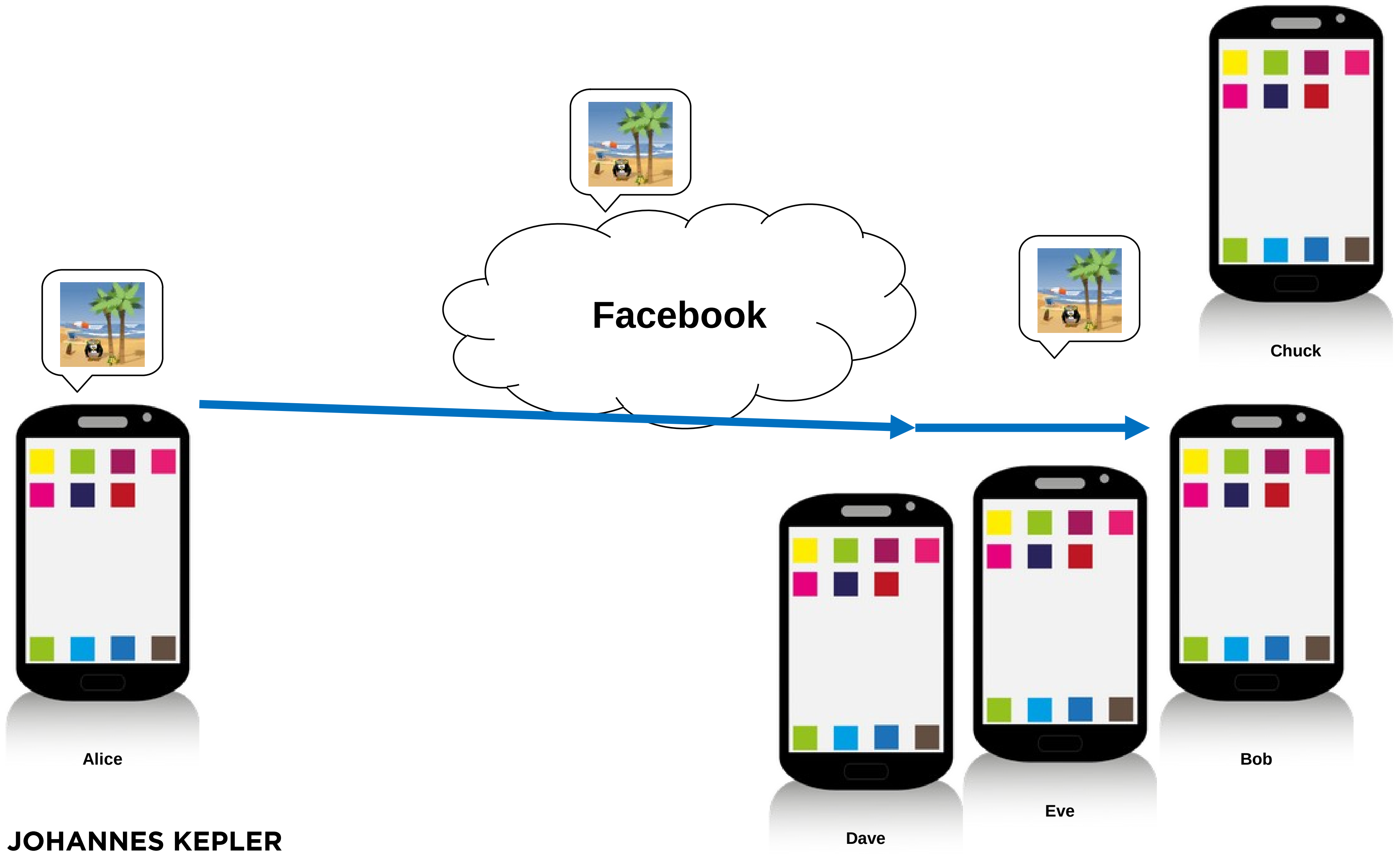
Which data can we extract from their network communication?

Traces through Xitter, Instagram, WeChat, etc.

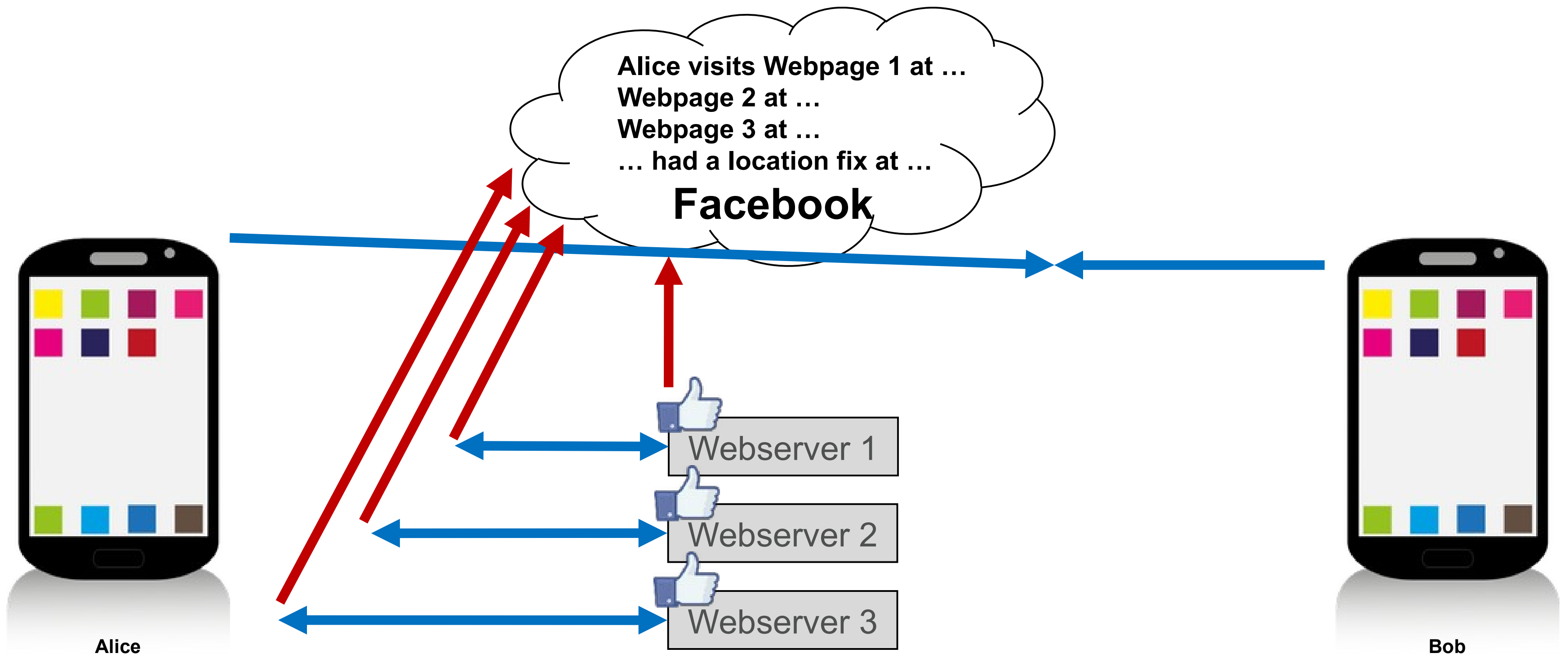


JOHANNES KEPLER UNIVERSITY LINZ

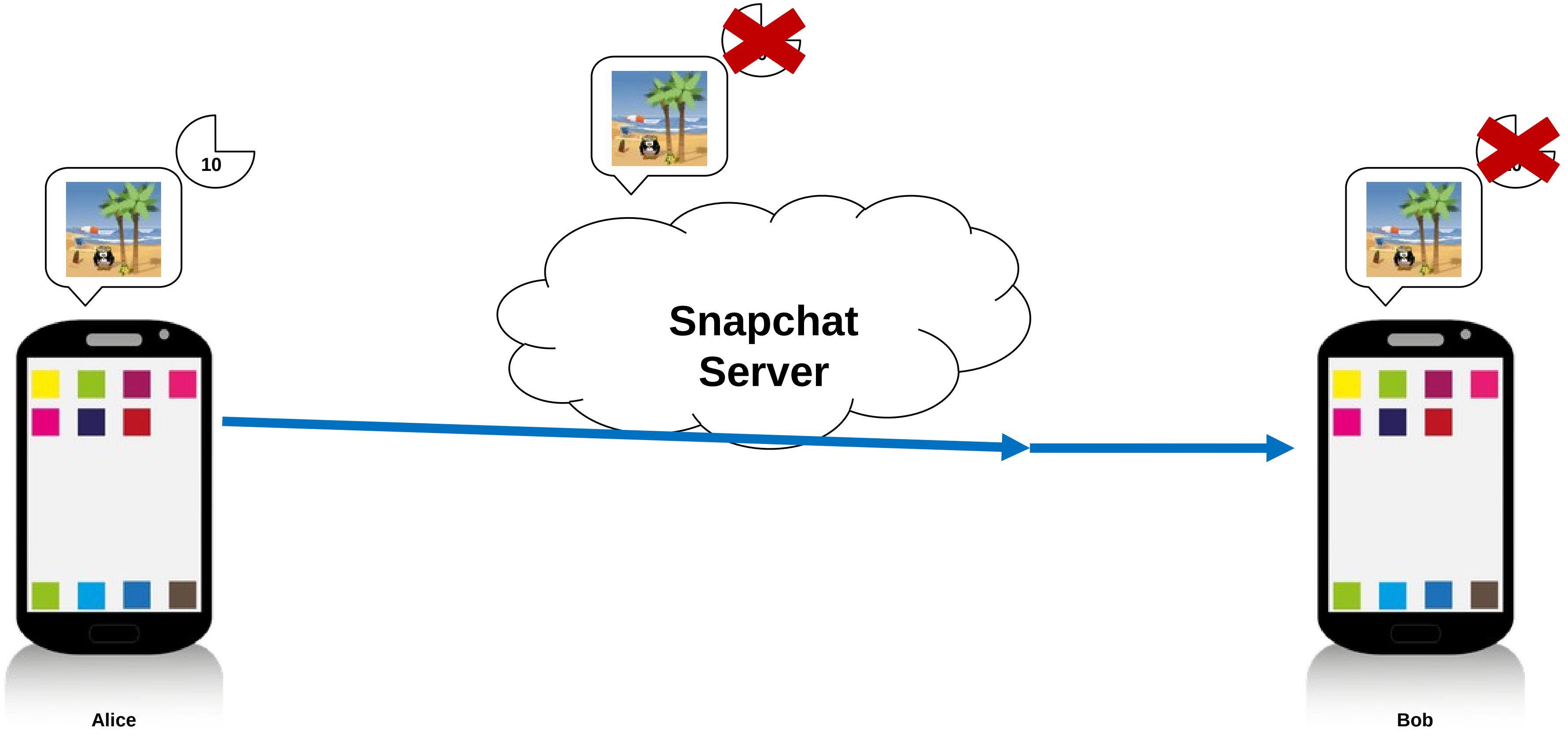
(Even more) traces through Facebook, Xitter, etc.



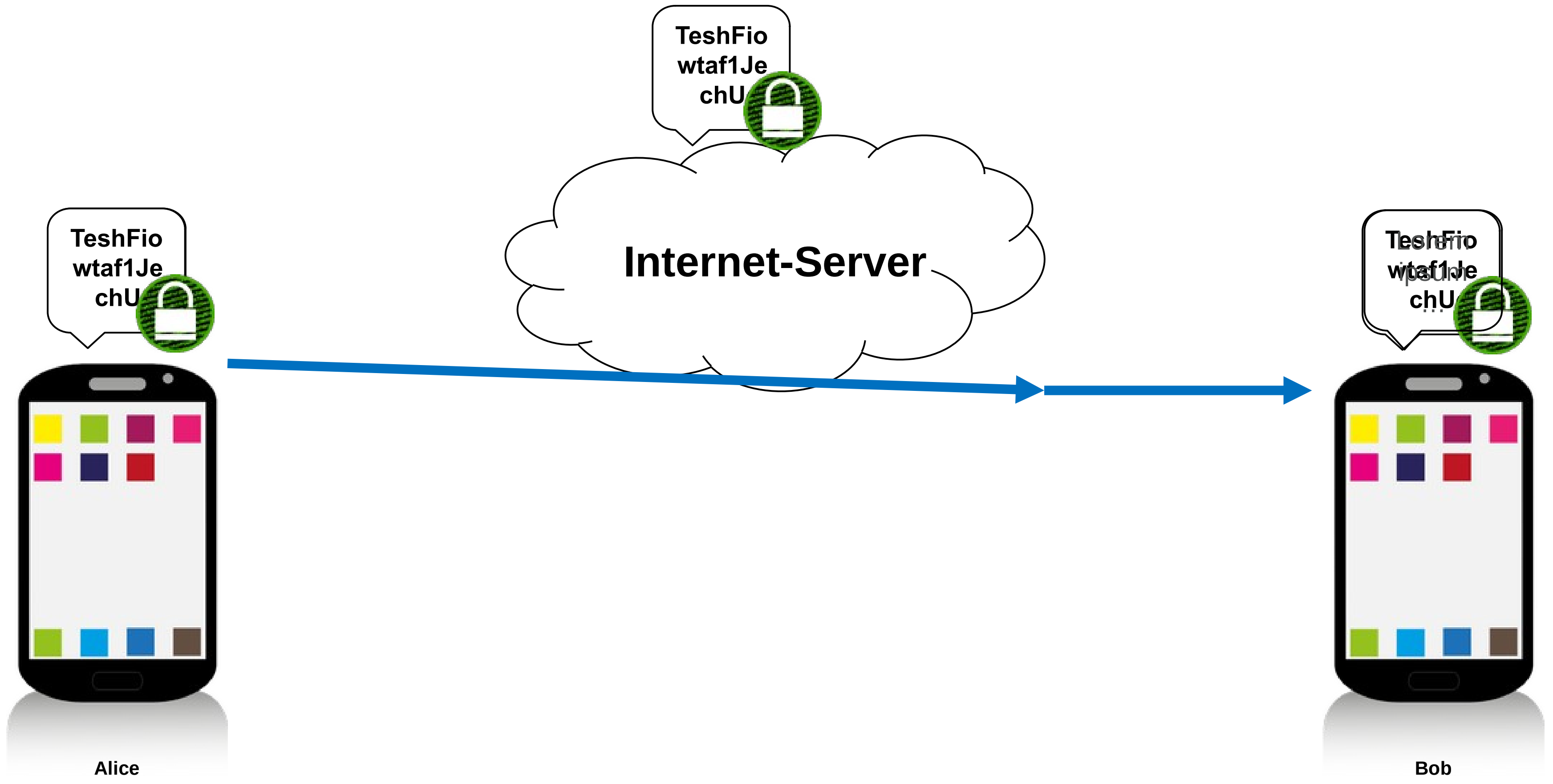
(Even more) traces through Facebook, Xitter, etc.



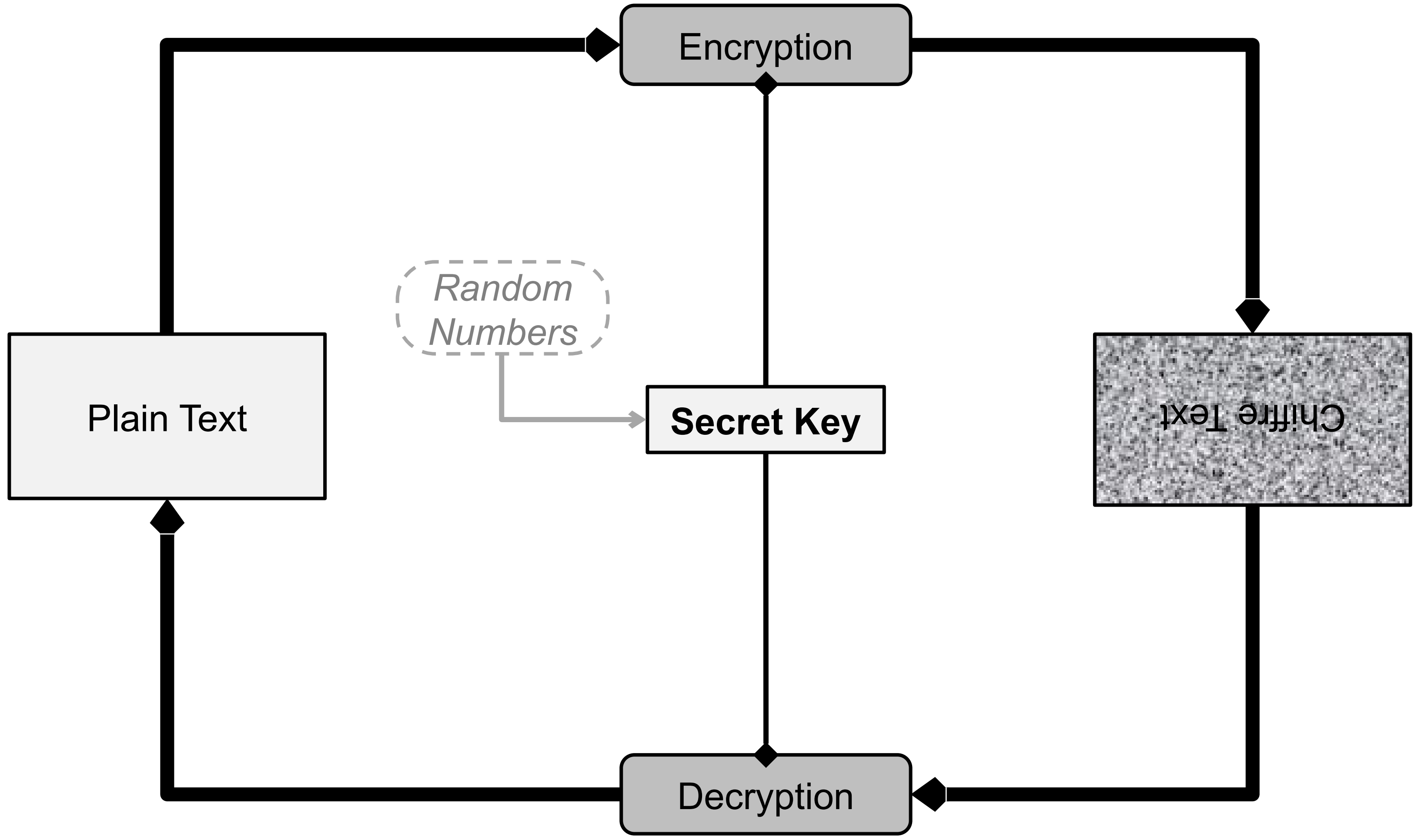
Traces through Snapchat, WhatsApp, etc.



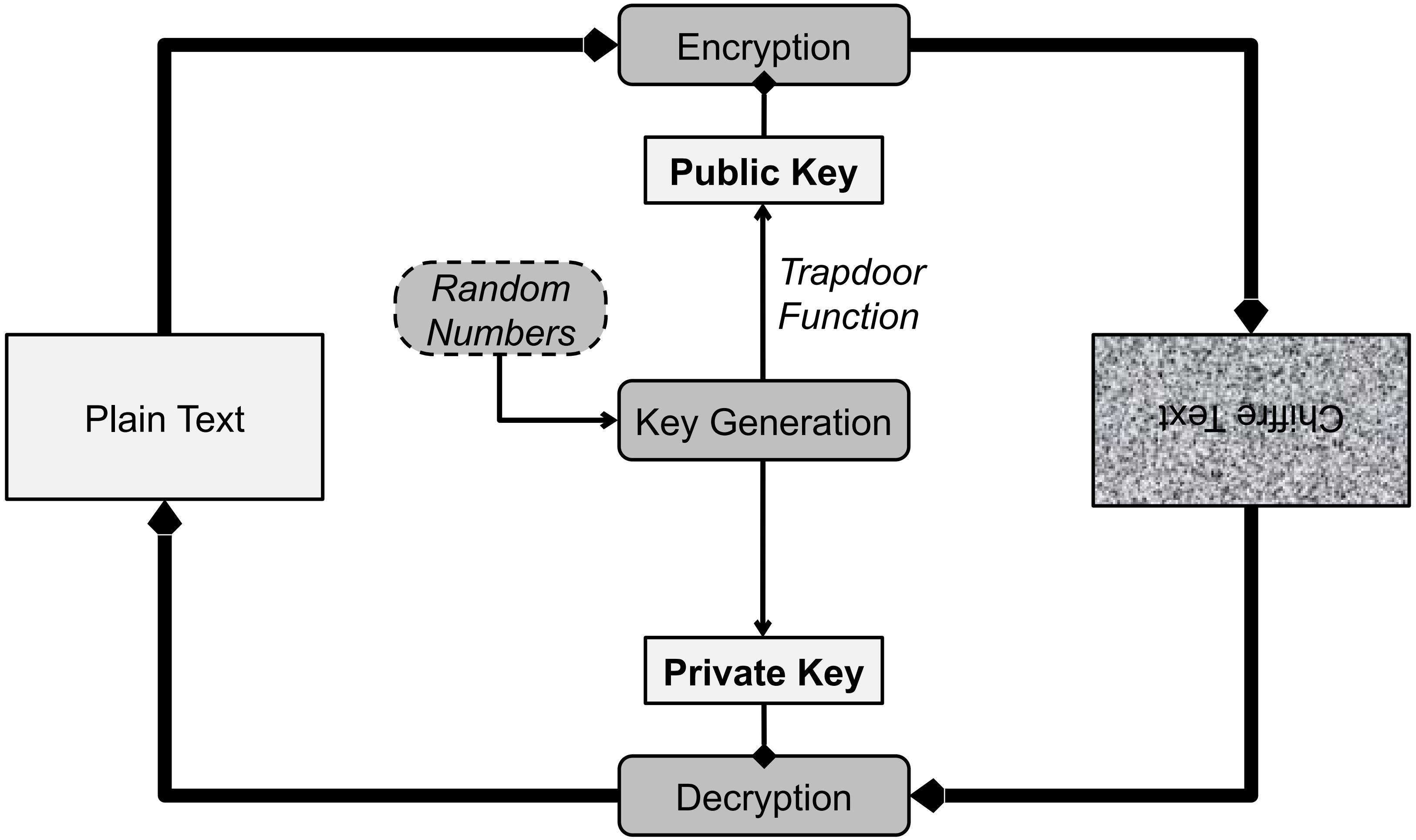
Traces through Signal, Wire, Threema, etc.: End-to-End Encryption (E2EE)



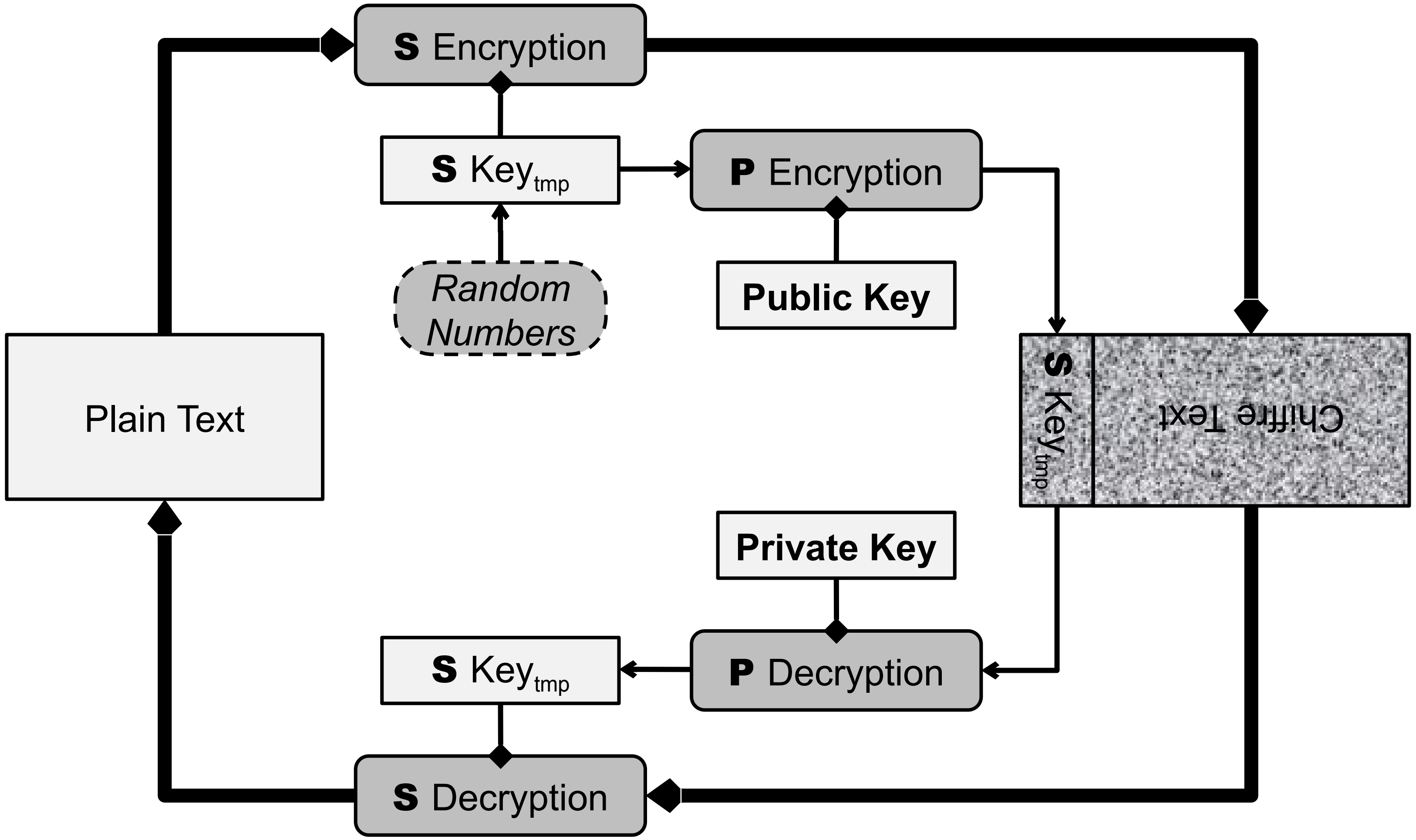
Encryption: symmetric



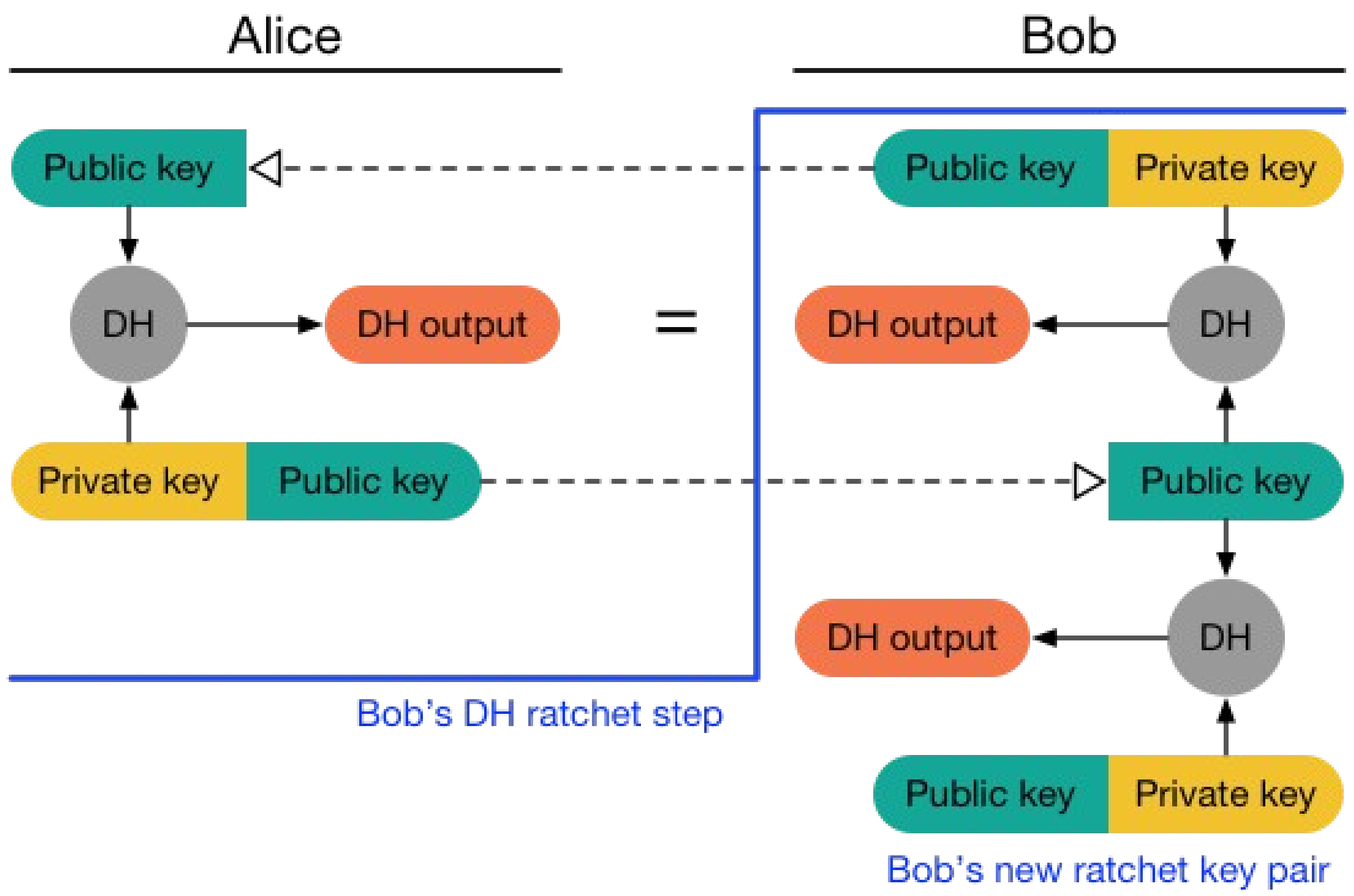
Encryption: asymmetric



Encryption: hybrid (partially b/c of performance)



Encryption:
Signal Protocol Double Ratchet



https://signal.org/docs/specifications/doubleratchet/

Data vs. meta data

- Content (= data) is well protected by Signal protocol double-ratched and IETF MLS
  - Signal
  - Wire (MLS, mostly)
  - Android Messages over RCS (and maybe iMessage interop, at some point...?)
  - WhatsApp (presumably, but closed source…)
  - Threema (neither Signal nor MLS, but considered *mostly* ok)
  - Matrix with Olm/MegOlm (but some concerns about protocol implementations)
  - NOT: Telegram
- Metadata is generally **not protected**
  - *Who* sends to *whom*, *which kinds* of messages, how *many*, how *often*, *when*, …
  - *Who* is a member of *which group(s)*, …
  - **“*We kill people based on metadata*”** (General Michael Hayden, former director of the NSA and the CIA)

Ok, network based content extraction is hard...

Can we just scan the endpoints (=apps) for plaintext messages?

Context: The Android ecosystem

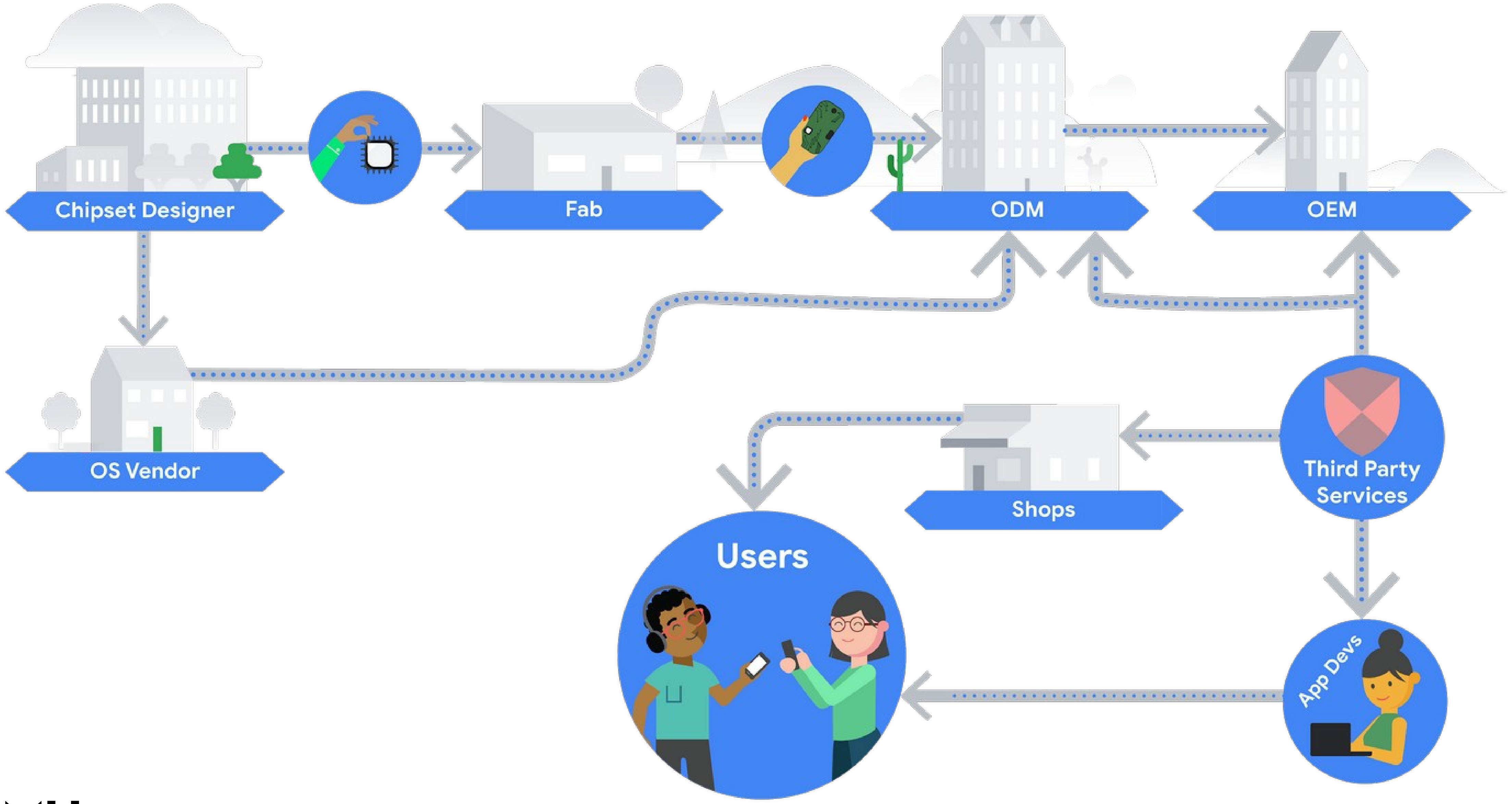


Image credit: Google

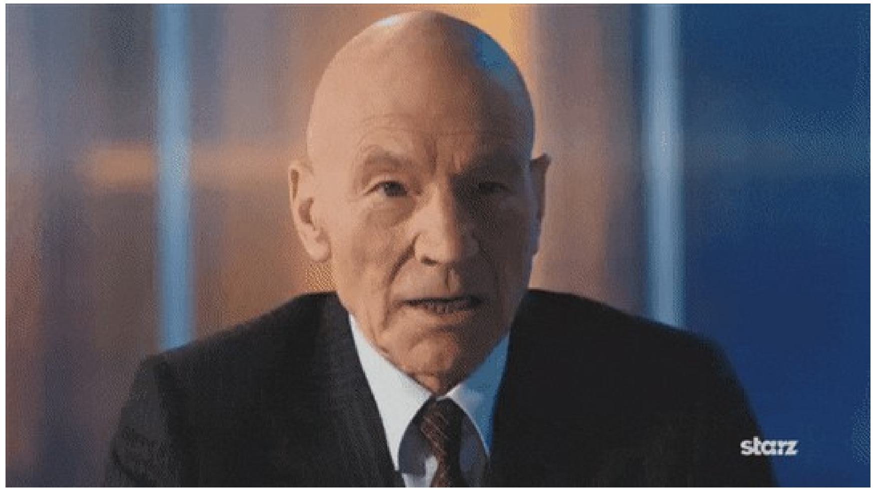
Goal for the Android ecosystem: Keep people safe

“*Make things so secure we’re not needed anymore.*”

— The Android platform security team

The Android Platform Security Model: Threat Model

- Adversaries can get **physical access** to Android devices (lost, stolen, borrowed, etc.)
  - Physical proximity
  - Powered off
  - Screen locked
  - Screen unlocked by different user
- **Network communication** and **sensor data** are untrusted
  - Passive eavesdropping
  - Active On-Path Attacker (OPA) / MITM
- **Untrusted code** is executed on the device
  - Includes all forms of OS/app API abuse
  - Includes misdirection, deception, etc. through UI
- **Untrusted content** is processed by the device
- New: Insiders can get access to signing keys



[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

The Android Platform Security Model: Rules

- Rule 1: **Multi-party consent**

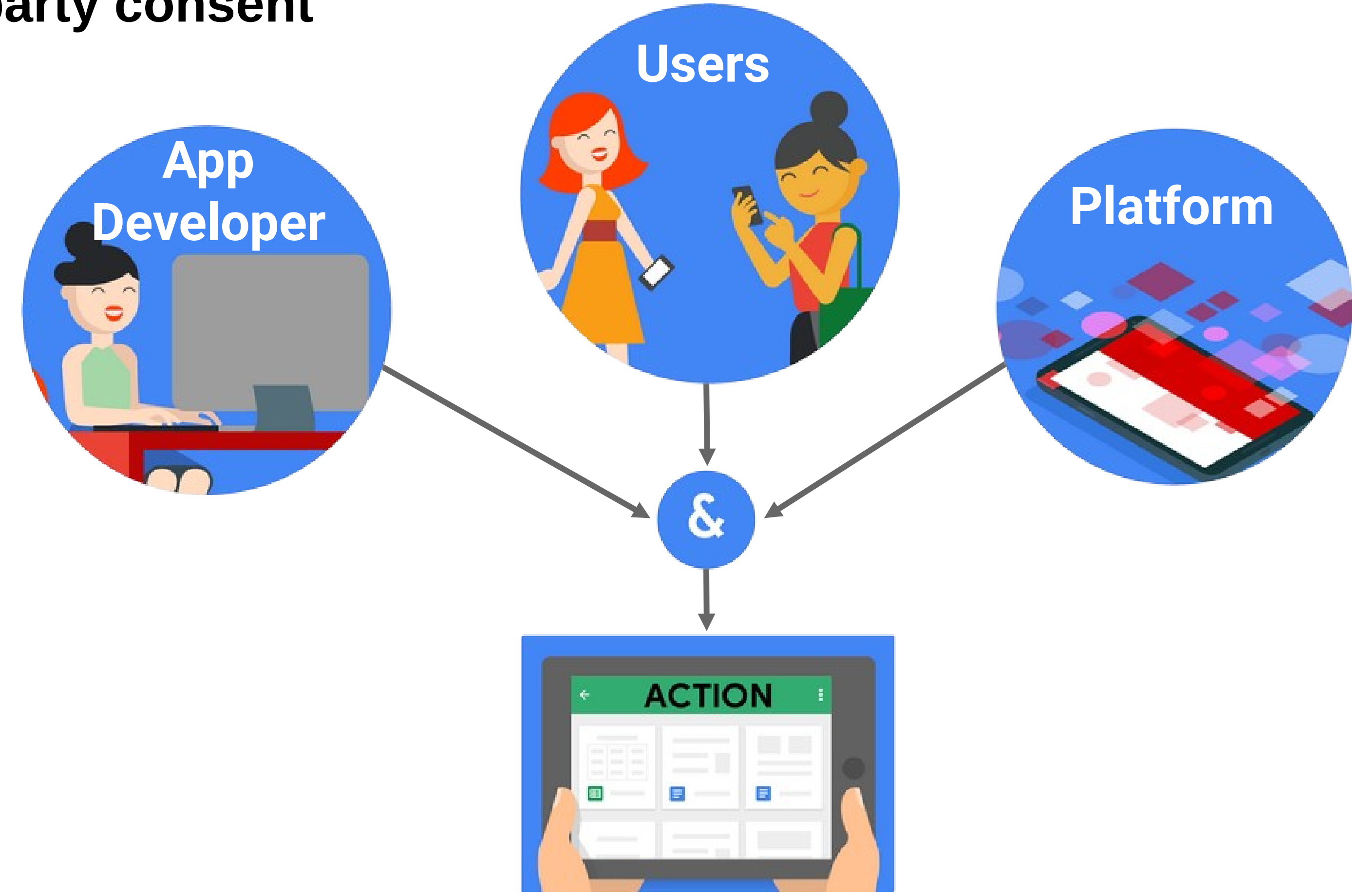


Image credit: Google

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

The Android Platform Security Model: Rules

- Rule 2: **Open ecosystem access**
- Rule 3: **Security is a compatibility requirement**
- Rule 4: **Factory reset restores the device to a safe state**
- Rule 5: **Applications are security principals**

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

Android app security principles

- **Applications must be signed for installation**
  - May be self-signed by the developer, therefore no strict requirement for centralized application Q/A or control (Google Play Signing manages keys for developers)
  - Signature supports non-repudiability (if the public key/certificate is known)
  - Signature by same private key allows applications to share data and files
  - Automatic application updates possible when signed by same private key

- **Otherwise, open eco-system**
  - Users may install arbitrary applications (directly from APK files or from different markets)
  - **Apps can be written in any language**
  - DRM and application copy protection available (Android 2.2 and newer market/PAI API), but optional

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

Android security architecture

Upon installation, package manager creates a dynamic user ID for each application
⇒ Application sandbox

- All application files and processes are restricted to this UID
- Enforced by Linux kernel and therefore same restrictions for all code (Java + native)
- Starting with Android 4.4 (introduced in 4.3 with `permissive` mode, 4.4 switches to `enforcing`), augmented with **SELinux** policy for kernel level mandatory access control (MAC)
- By default, even the user and debugging shells are restricted to a special UID (`SHELL`)
- Permissions granted at installation time allow to call services outside the application sandbox

“rooting” to gain “root” access (super user / system level access on UNIX without further restrictions, but may be limited by SELinux MAC)

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. “The Android Platform Security Model‘, ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

Android security boundaries

Android sandbox has **two main layers of permissions models**

- **File system entries and some other kernel resources**
  - Enforced by DAC (standard filesystem permissions) and in newer versions MAC (SELinux) ⇒ enforced on kernel level
  - Very restrictive compared to standard Linux distributions (or Windows, MacOS, ...)
  - Android ID (AID) is used as both UID (user ID, for installed applications) and GID (group ID, for accessing resources)
  - Commonly referred to as "**Android sandbox**" (although this is not the full picture)
- **Permissions on API calls**
  - Enforced by DalvikVM/ART and Android framework/libraries, as well as specific apps
  - Allow bridging the security boundary created by the first layer enforced by kernel sandbox
- Plus other mechanisms for specific purpose (e.g. Linux capabilities and `seccomp` filters)

For interplay between DAC, MAC, and CAP see e.g. [Hernandez et al.: "*BigMAC: Fine-Grained Policy Analysis of Android Firmware*", USENIX Security 2020], online at https://www.usenix.org/conference/usenixsecurity20/presentation/hernandez

[R. Mayrhofer, J. Vander Stoep, C. Brubaker, N. Kralevich. "The Android Platform Security Model', ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, Article 19 (August 2021), https://doi.org/10.1145/3448609, previous and future versions at arXiv:1904.05572, April 2019 and later]

Ok, one app cannot simply scan another app’s data…

How do we get (OS) level privileges to override the sandbox?

Overriding app sandbox: different options

- Change OS image to get your own app into privileged level
  - Exploit vulnerability in OS (or the respective app)

Why exploiting OS vulnerabilities is not a good long-term plan

- Need to use publicly unreported bugs
  - Known bugs are “quickly” fixes by vendors
  - Need a constant stream of “new” unknown exploits
  - Fundamental mitigations get better over time → exploitability is measurably reduced
  - **This is expensive!**
- “NOBUS - nobody but us” is an illusion
  - Experience shows that the same vulnerabilities are found by different teams
  - Need to assume unreported bugs are also exploited for other malicious reasons
  - **Not reporting them leaves the public open to attacks!**
- The gray/black market of zero-day exploits is nasty
  - Can you really write all your own exploits? No? Go and buy them on the market!
  - Some political regimes use the same attacks against the opposition, journalists, …
  - Organized crime uses the same attacks for their purposes

→ **Public tax money funds criminal and politically unethical activities!**

JKU

Overriding app sandbox: different options

- Change OS image to get your own app into privileged level
  - ~~Exploit vulnerability in OS (or the respective app)~~
  - Modify base image
    - with local physical access, unlocked bootloader, reflashing tampered image

Android code signing

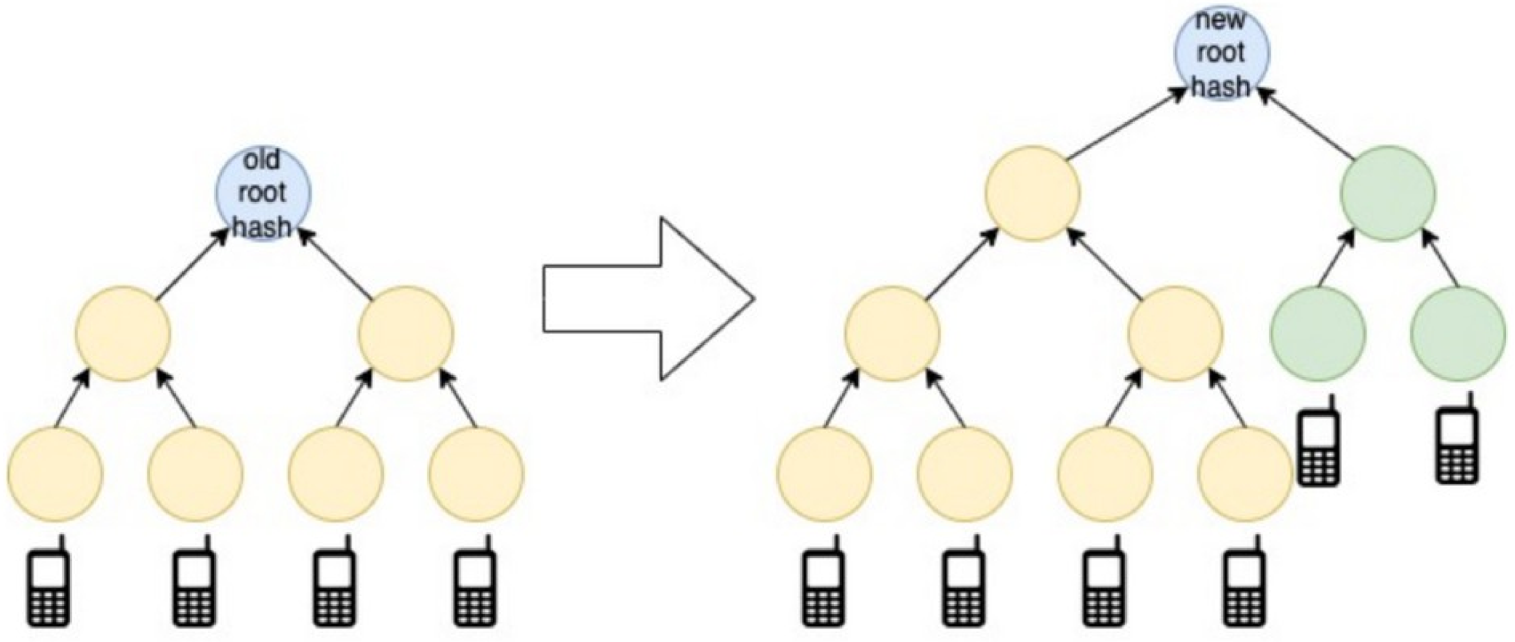
- **All Android apps (system and user-installed) must be signed**
  - Typically, firmware updates are also signed by OEM, boot loader may only allow to flash and/or boot "correctly" signed images
  - Recovery mode often applies only updates signed by same OEM
  - Newer Android versions **verify signatures during boot and run-time** (*dm-verity*)
- Signing is done with private keys held by developers / organizations, public keys embedded in individual apps, system image, and/or in boot loader for image signatures
  - **All the way down to the Android Verified Boot (AVB) chain**
- Signing key types:
  - Individual developer keys (self-signed) for apps
  - `platform`, `shared`, `media` and `testkey` in AOSP tree
    - `platform` is used for "core" Android components with elevated privileges
  - `releasekey` for release type image builds, must by kept private
  - More details at https://source.android.com/devices/tech/ota/sign_builds.html

Overriding app sandbox: different options

- Change OS image to get your own app into privileged level
  - ~~Exploit vulnerability in OS (or the respective app)~~
  - Modify base image
    - ~~with local physical access, unlocked bootloader, reflashing tampered image~~
    - through over-the-air upgrade (compelling OEM to create a targeted version)

Pixel Binary Transparency Log

For those who want to understand more about how this works, the Pixel Binary Transparency log is append-only thanks to a data structure called a Merkle tree, which is also used in blockchain, Git, Bittorrent, and certain NoSQL databases. The append-only property is derived from the single root hash of the Merkle tree—the top level cryptographic value in the tree. The root hash is computed by hashing each leaf node containing data (for example, metadata that confirms the security of your Pixel's software), and recursively hashing intermediate nodes.



The root hash on the right changes because of the addition of the green nodes (metadata of new devices).

The root hash of a Merkle tree should not change, if and only if, the leaf nodes do not change. By keeping track of the most recent root hash, you also keep track of all the previous leaves. You can read more about the details in the Pixel Binary Transparency documentation.

https://security.googleblog.com/2023/08/pixel-binary-transparency-verifiable.html

Overriding app sandbox: different options

- ~~Change OS image to get your own app into privileged level~~
  - ~~Exploit vulnerability in OS (or the respective app)~~
  - ~~Modify base image~~
    - ~~with local physical access, unlocked bootloader, reflashing tampered image~~
    - ~~through over-the-air upgrade (compelling OEM to create a targeted version)~~
- Read plaintext data directly from physical device
  - Device is switched on

User authentication (to their own devices)

- On most mobile devices, the “lock screen” is the primary method of authentication
- (Mostly) binary distinction: locked or unlocked
  - some nuance with notifications and other information on lock screen
  - some functions can be used on locked phones (e.g. camera or emergency call)
- Can integrate with key management (Keymint / StrongBox)
- But implemented by Android user space ⇒ cannot defend against root adversaries
  (Exception: authentication-bound keys imply that authentication state is verified in TEE and passed directly to Keymint in TEE and therefore resistant to root adversaries)

Overriding app sandbox: different options

- ~~Change OS image to get your own app into privileged level~~
  - ~~Exploit vulnerability in OS (or the respective app)~~
  - ~~Modify base image~~
    - ~~with local physical access, unlocked bootloader, reflashing tampered image~~
    - ~~through over-the-air upgrade (compelling OEM to create a targeted version)~~
- Read plaintext data directly from physical device
  - ~~Device is switched on~~
  - Device is switched off

On-device encryption

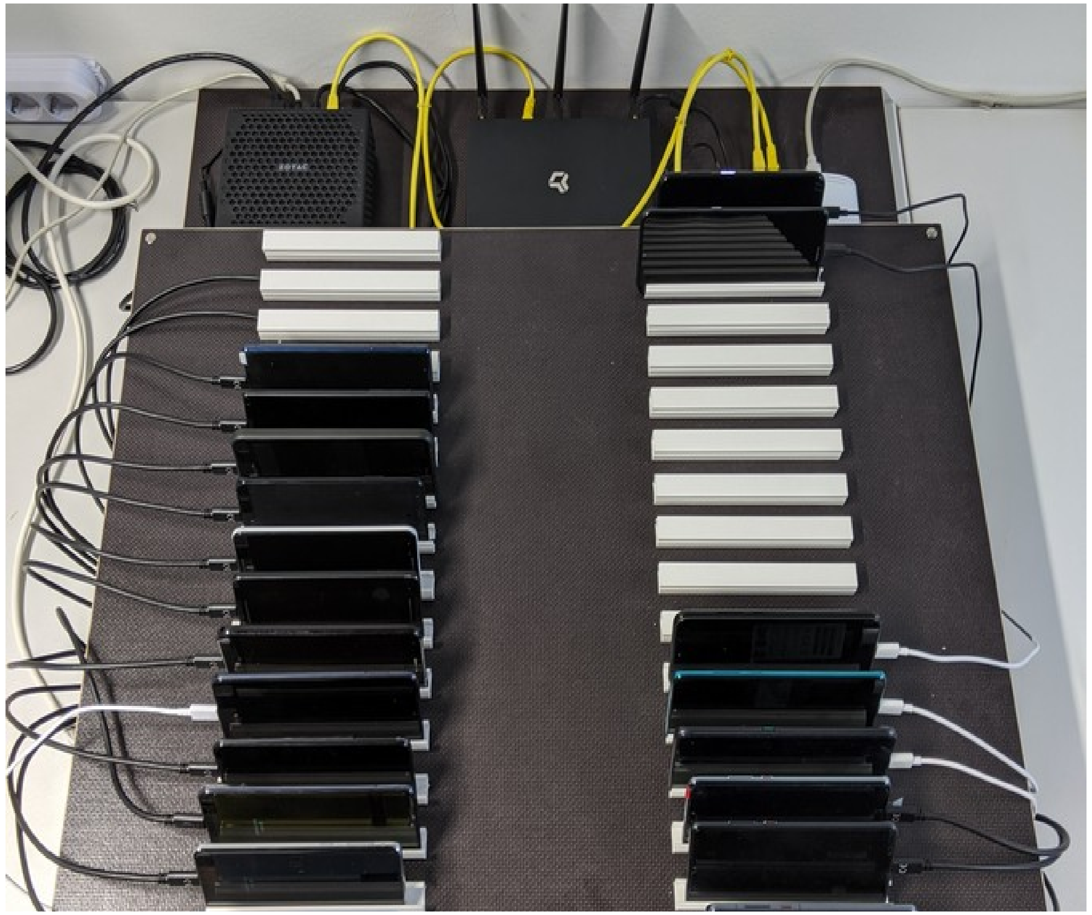
- Android 5.0 introduced **Full Disk Encryption** (FDE)
  - Entangled with user knowledge factor (PIN/password), but can potentially be disabled (then encryption key only depends on device-unique key kept in TrustZone)
  - Full data partition encrypted with same key, including meta data (e.g. file names)
  - All user accounts and profiles encrypted with same key
  - Most system functions inaccessible until knowledge factor entered during reboot
- Android 7.0 introduced **File Based Encryption** (FBE)
  - Different keys per users/profiles
  - Difference between “device encrypted” (DE, only bound to unique device key) and “credential encrypted” (CE, entangled with user knowledge factor)
  - Apps that are marked to use DE data storage can function after reboot before first unlock
  - Android 9 added meta data encryption
  - **Android 10 made FBE mandatory for all new devices**
  - Android 11 introduced Resume-on-Reboot

Overriding app sandbox: different options

- ~~Change OS image to get your own app into privileged level~~
  - ~~Exploit vulnerability in OS (or the respective app)~~
  - ~~Modify base image~~
    - ~~with local physical access, unlocked bootloader, reflashing tampered image~~
    - ~~through over-the-air upgrade (compelling OEM to create a targeted version)~~
- ~~Read plaintext data directly from physical device~~
  - ~~Device is switched on~~
  - ~~Device is switched off~~

Android-Device-Security.org: First lab at JKU Linz

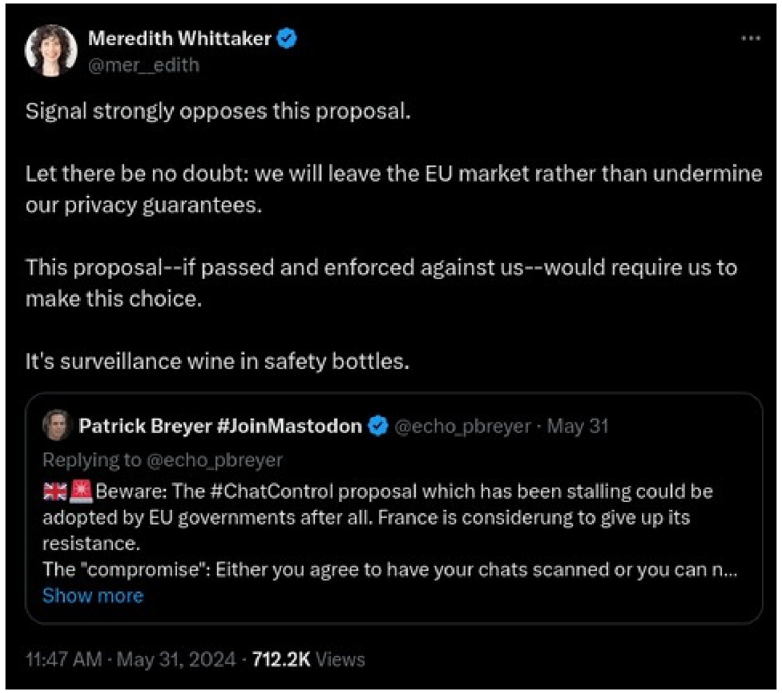
- 27 different devices so far
  - Focus on European market, 9 different OEMs
  - Low-end, mid range, and flagship devices
  - Unmodified, stock system images
- Controlled through ADB
  - Reading system properties, list of apps, etc.
  - Installing test apps, collecting results
  - Daily reboot to force applying updates
- Connected through custom WiFi access point
  - One VLAN per device (selected by 802.1x)
  - Allows tracking all network traffic including layer 2 addresses (MAC randomization)
- **Looking for collaboration with more labs**



Ok, we can’t scan app content from another app, but can we force the app to do its own scanning?

Letting apps do their own scanning: Client-Side Scanning (CSS)

- Can legally compel apps to implement scanning inside the app
  - Has access to plaintext messages and all media
  - Proprietary apps can implement a mandated **secret filter**
    - with our without enforced automatic reporting
- Technical challenges
  - **Filter has non-negligible error rate**
    - Many, many, many false positives to be expected
  - **Keeping filter secret** → even if non-extractable (which is hard), can use as oracle
    - Training input recovery is a thing with more complex filter models → CSAM material???
  - **No way to technically enforce on all apps** → take e.g. Signal source code, compile without filter, use within organized crime group
  - **Added complexity** → added attack surface for app
- Legal challenges
  - Mass surveillance “pre-crime” scanning
  - Self censorship based on existence of filter


Meredith Whittaker @mer_edith

Signal strongly opposes this proposal.

Let there be no doubt: we will leave the EU market rather than undermine our privacy guarantees.

This proposal--if passed and enforced against us--would require us to make this choice.

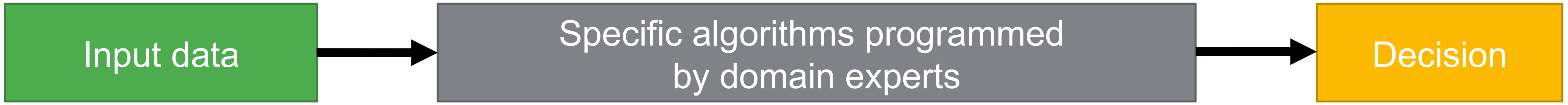
It's surveillance wine in safety bottles.

Patrick Breyer #JoinMastodon @echo_pbreyer · May 31
Replying to @echo_pbreyer
🇬🇧🚨Beware: The #ChatControl proposal which has been stalling could be adopted by EU governments after all. France is considerung to give up its resistance.
The "compromise": Either you agree to have your chats scanned or you can n...
Show more

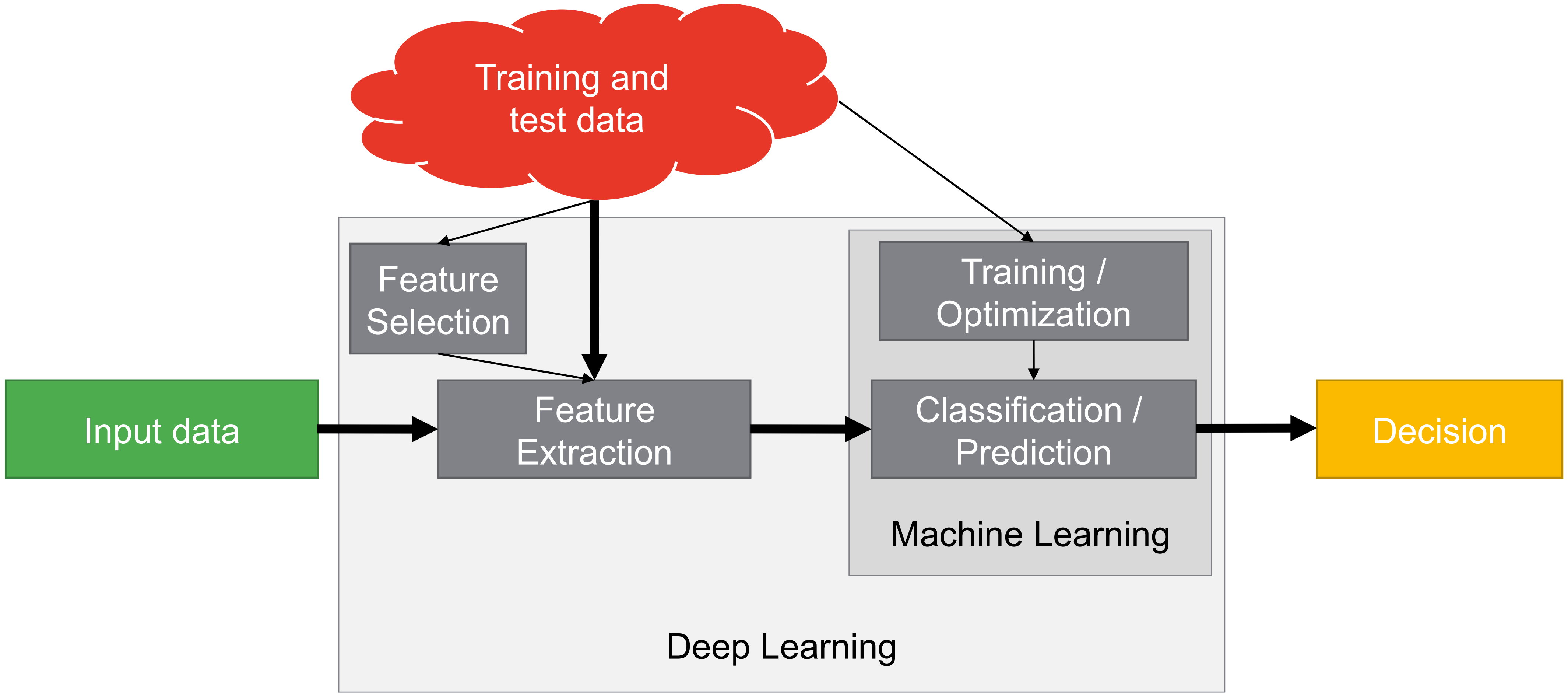
11:47 AM · May 31, 2024 · 712.2K Views

See also https://www.ins.jku.at/chatcontrol/

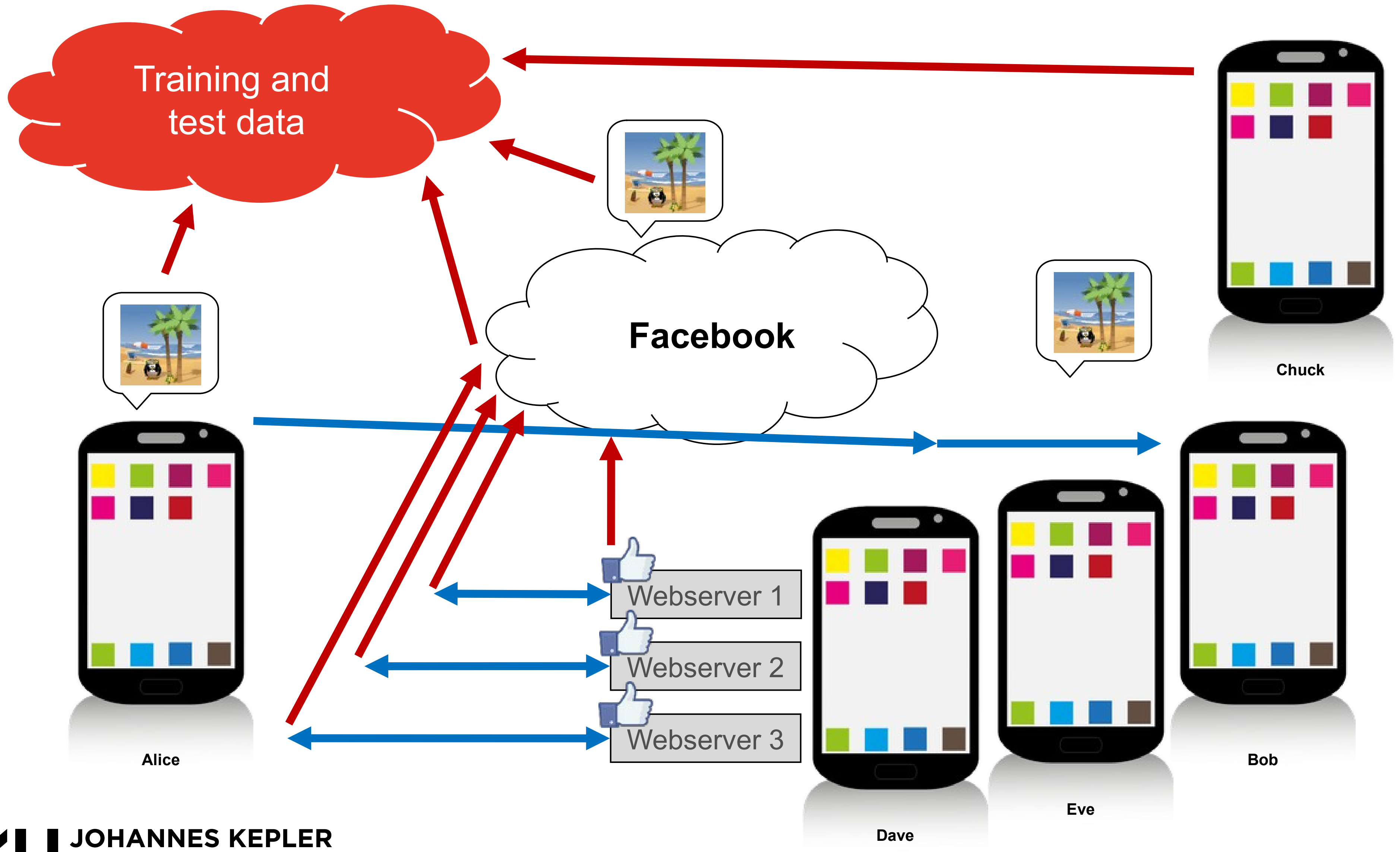
What is AI?
Better describe as Machine Learning (ML)



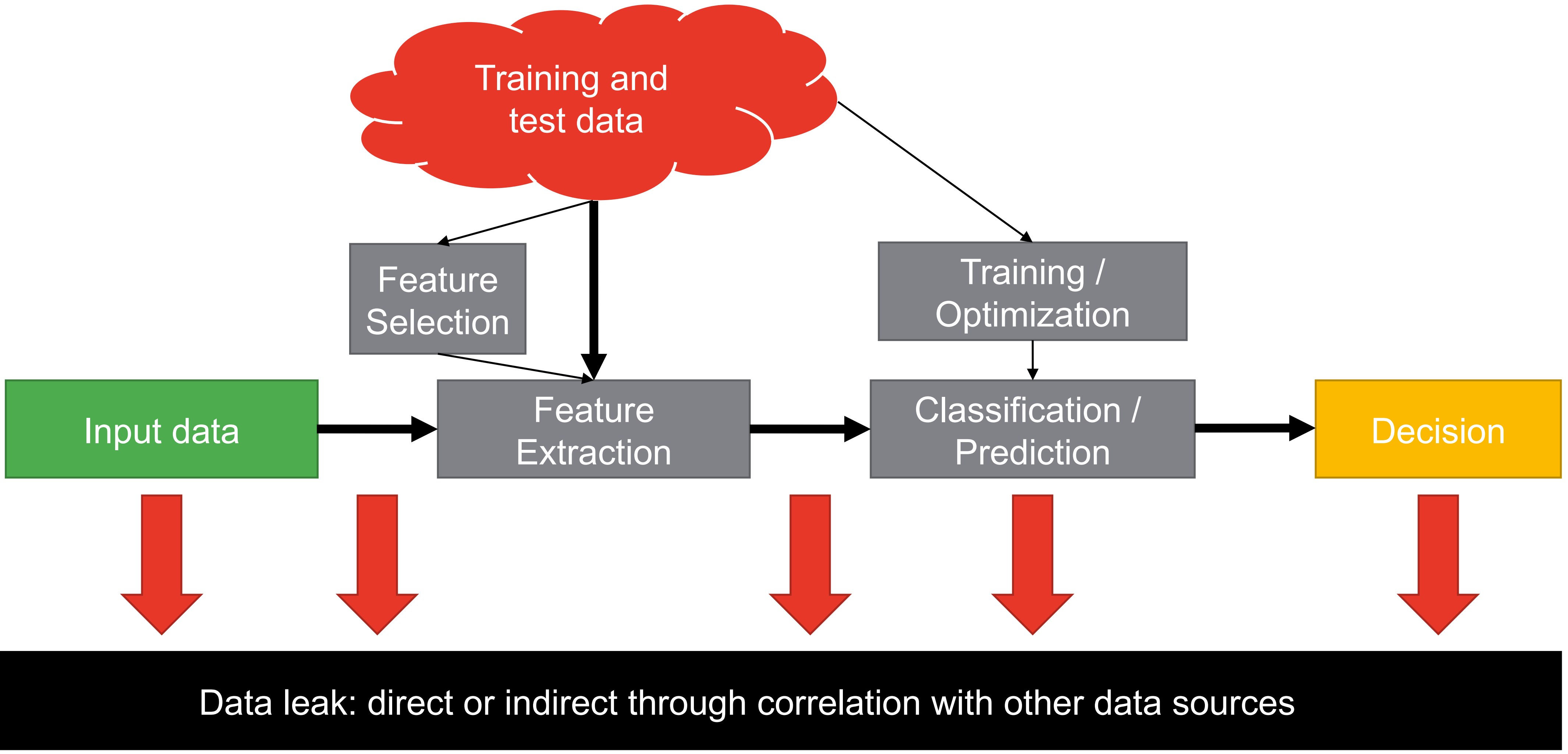
What is AI?
Better describe as Machine Learning (ML)



ML at Facebook, Xitter, etc.



Possible data leaks when using ML

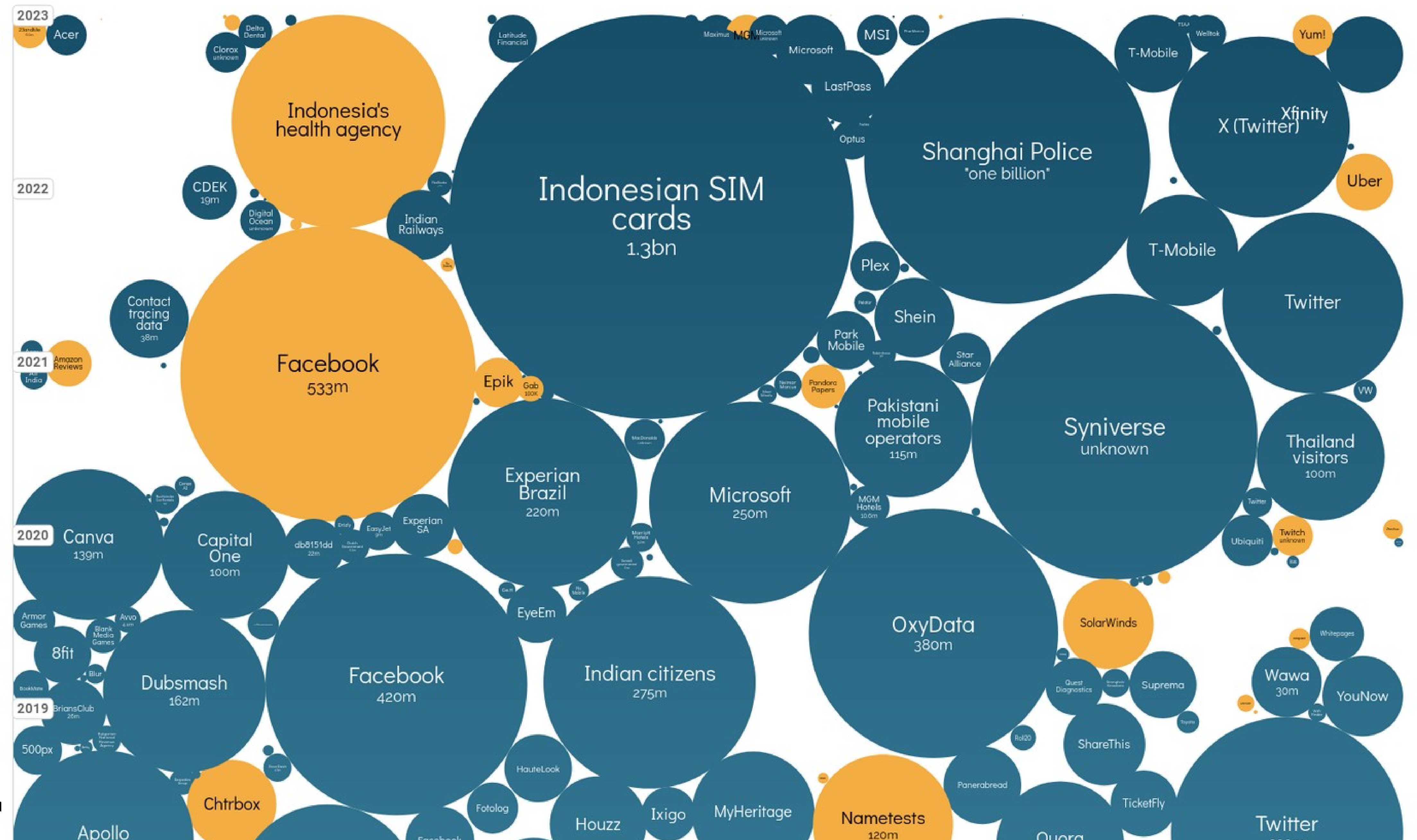

Training and test data
Feature Selection
Training / Optimization
Input data
Feature Extraction
Classification / Prediction
Decision
Data leak: direct or indirect through correlation with other data sources

Selected events over 30,000 records stolen

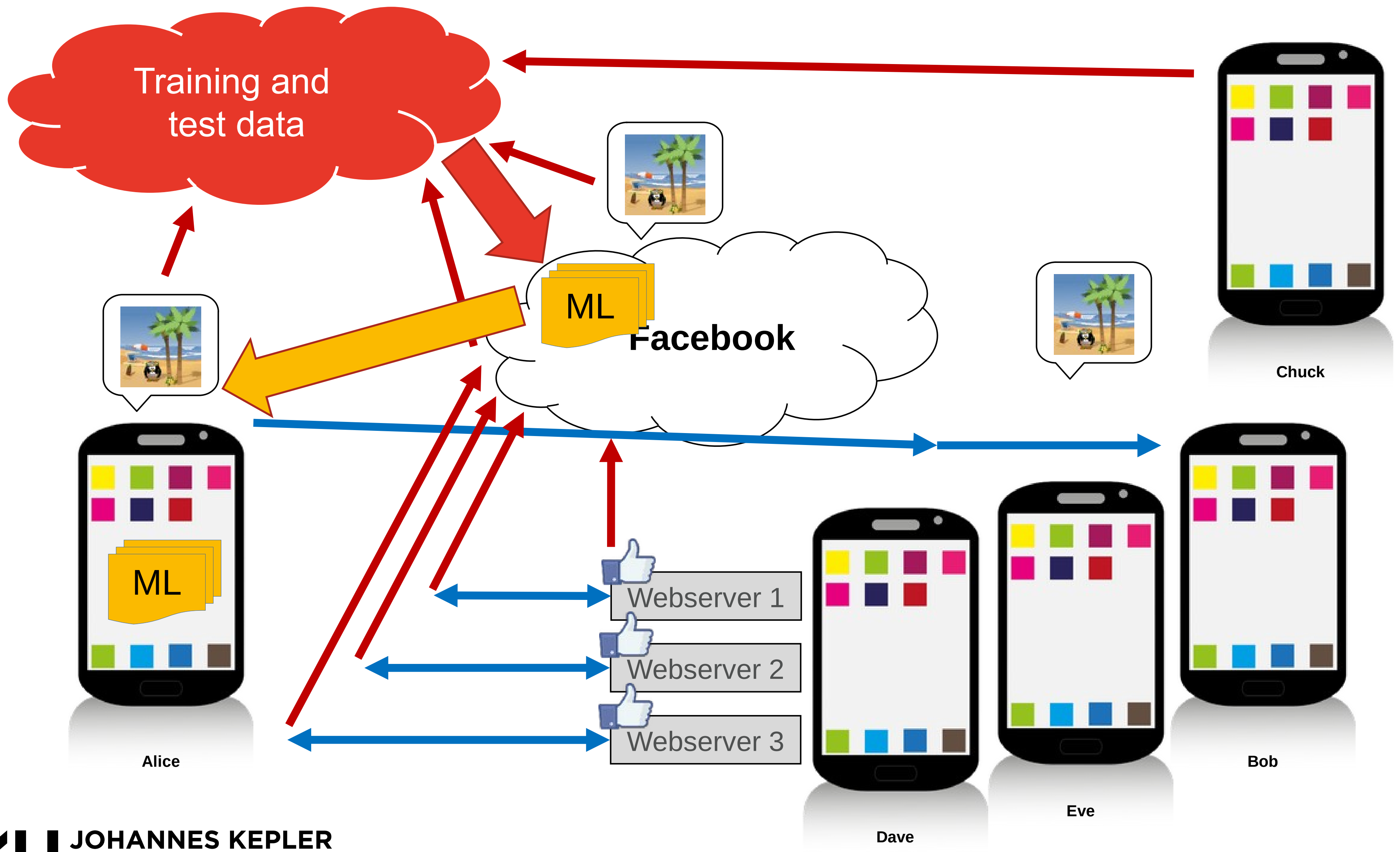
UPDATED: Jan 2024

https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

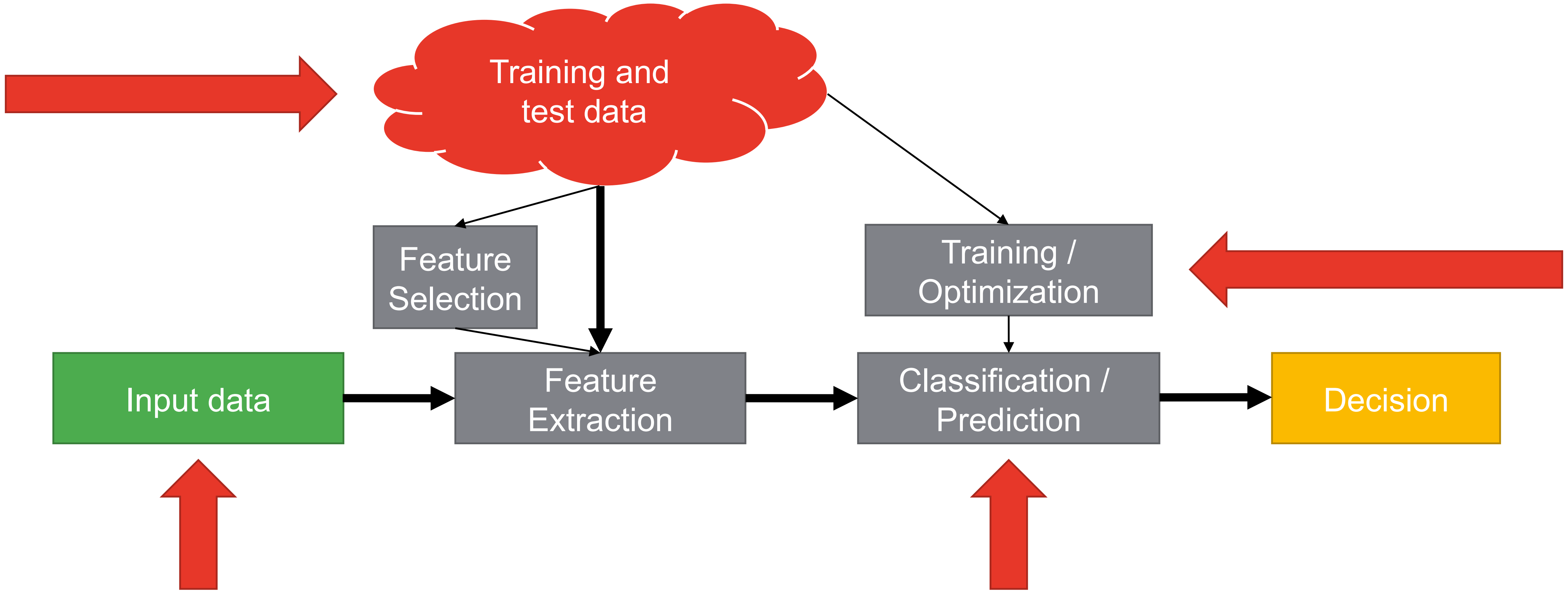
size: records lost **filter** search...



ML at Facebook, Xitter, etc.

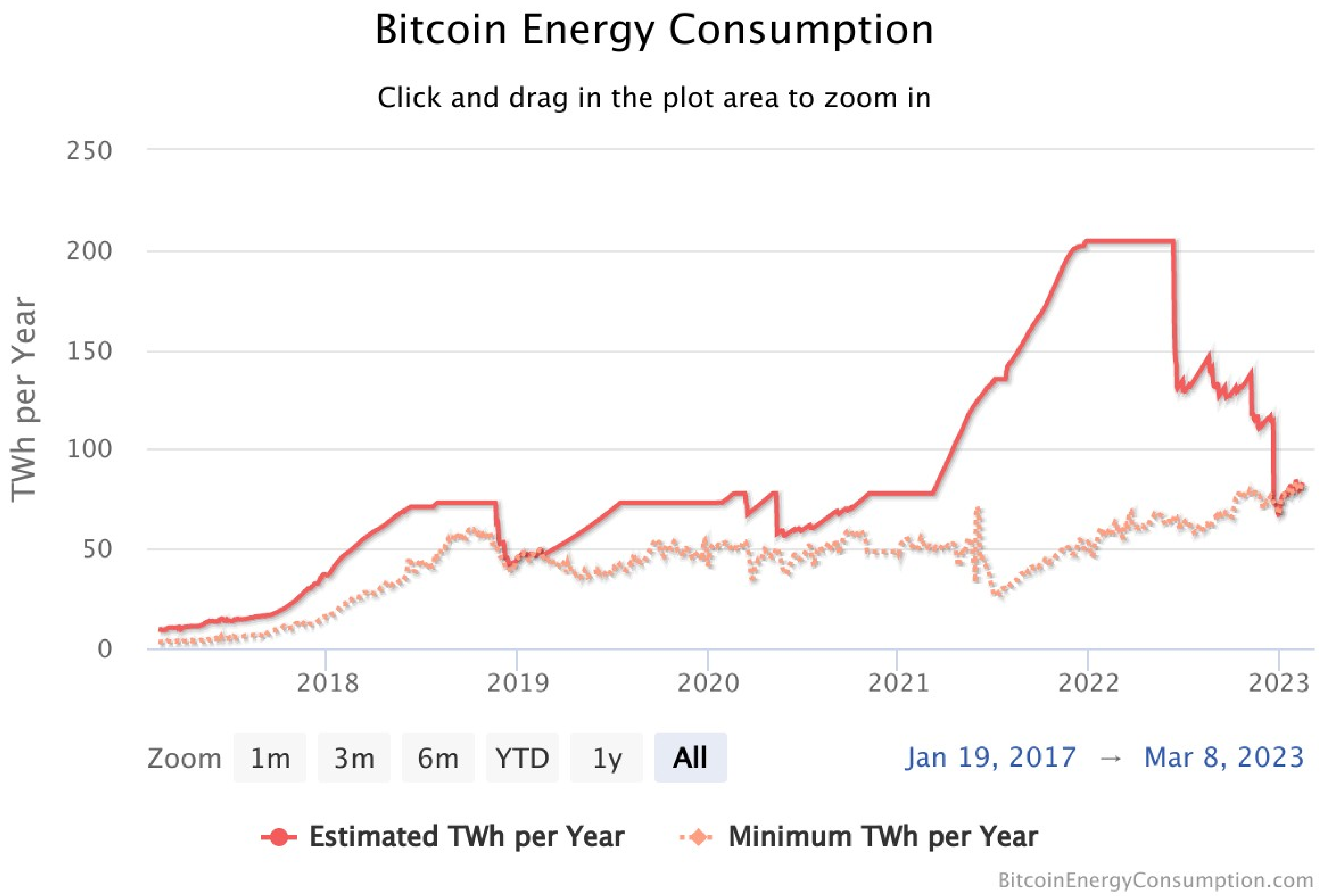


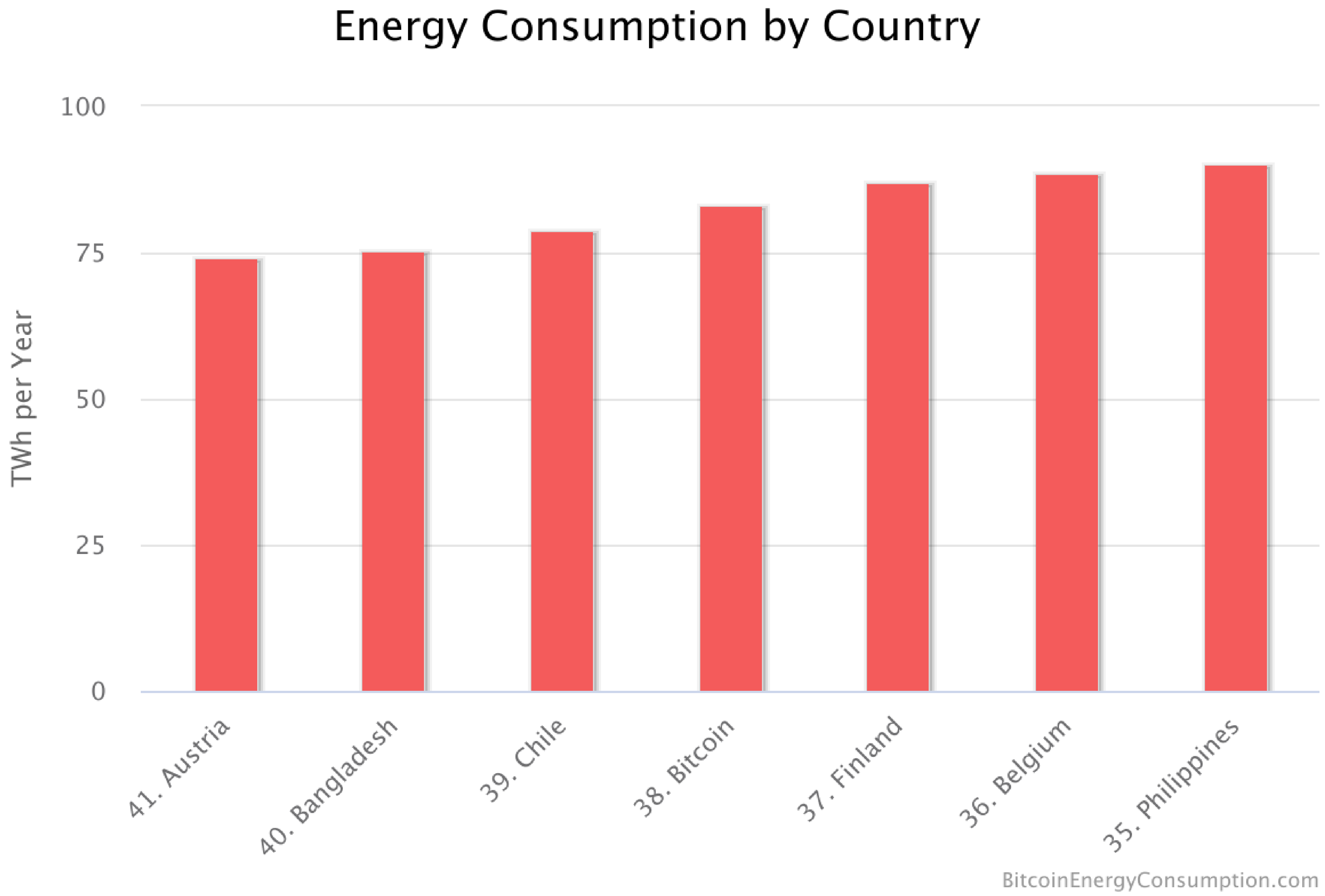
Possible attacks when using ML



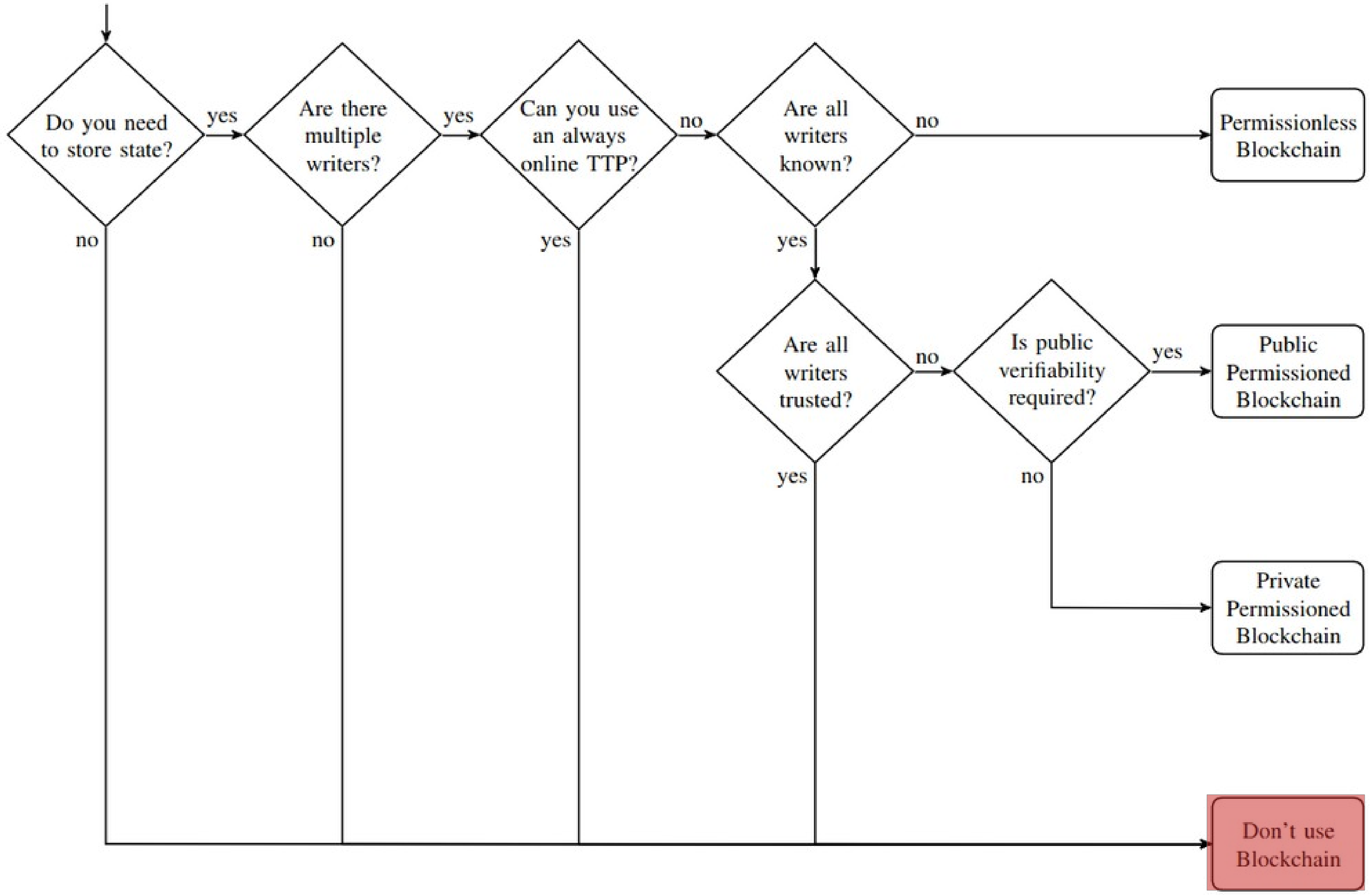
Ok, client side scanning (with or without ML) is tricky, how about some other security whatever thingy?

Blockchain for Social Media / ML Security?


Bitcoin Energy Consumption
Click and drag in the plot area to zoom in
250
200
150
100
50
0
TWh per Year
2018
2019
2020
2021
2022
2023
Zoom
1m
3m
6m
YTD
1y
All
Jan 19, 2017 → Mar 8, 2023
Estimated TWh per Year
Minimum TWh per Year
BitcoinEnergyConsumption.com


Energy Consumption by Country
100
75
50
25
0
TWh per Year
41. Austria
40. Bangladesh
39. Chile
38. Bitcoin
37. Finland
36. Belgium
35. Philippines
BitcoinEnergyConsumption.com

Do you need a Blockchain?


Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

Source: K. Wüst and A. Gervais, "Do you Need a Blockchain?," 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), 2018, pp. 45-54, doi: 10.1109/CVCBT.2018.00011, also online as https://eprint.iacr.org/2017/375.pdf

JOHANNES KEPLER
UNIVERSITY LINZ

Questions?

Web: https://ins.jku.at
Email: rm@ins.jku.at
Signal: Rene.02

~~Twitter: @rene_mobile~~
Mastodon: @rene_mobile@infosec.exchange

DIGiDOW

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenberger Straße 69
4040 Linz, Austria
jku.at