# Ryoan

Presented by: Ghulam Murtaza

# Threat Model

- Who do users trust?
- Network?
  - Ryoan? DIY?
- Application Provider (Paymaxx or 23andMe)
  - Ryoan? DIY?
- Service Provider (Amazon Lambda, EC2, Google)
  - Ryoan? DIY?
- Attestation server? (Intel SGX attestation server, Amazon Key Management Server)
  - Ryoan? DIY?
- Do users trust Service provider to not collude with Application provider?

# Threat Model

- Other malicious users colluding with malicious service provider

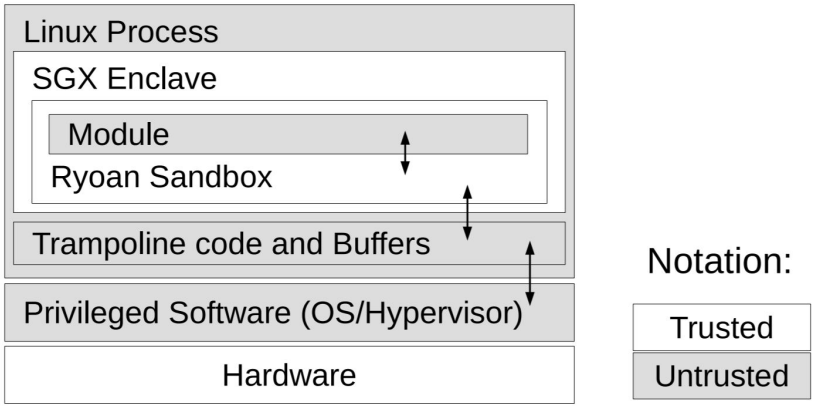
- OS/VMM (Service provider) colluding with Application provider
  - Encoding user's information on system calls (mmap request)
  - Information leak through processing time, output size, network request frequency (I/O etc)
  - OS libc

- Is hardware trusted?
  - Yes, and so is Ryoan.

# Threat Model

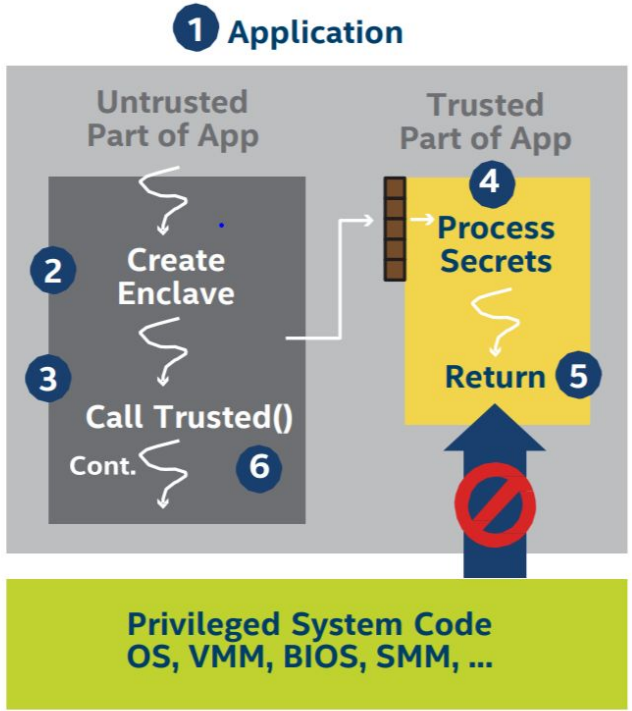
- Who do the service providers trust?
  - Their own code?
  - They face similar threats, because in an abstract sense they are users as well. (If they are hosting their application on some service)

- Figure on the right summarizes the threat model
  - Module is application provider's code (Paymaxx)
  - Hardware and Privileged software might or might not be running on an external service provider



# Intel SGX

- Enclaves
  - Loads the code in a hardware encrypted memory region
  - Provides guarantees that memory is not accessible by a privileged process and won't be tampered with.
- Attestation
  - Hardware signed hash that can be verified remotely or locally to make sure the correct code was loaded in the Enclave.



# Google NaCl

- Provides a sandbox to run native x86, ARM or MIPS code

- Can control how the sandboxed code interacts with OS
  - Runs a code verifier to find unsafe instructions such as System calls and executes them on code's behalf
  - Strong restrictions on the binary that gets executed, so code needs to be recompiled with special compilers

# Programming Model

- What specific type of Applications are supported currently? Why?
  - Request Oriented only!
  - Well defined unit of work, One request —> One Result
  - They define this piece of code as a *module* (and a bunch of other stuff, more on it later)
  - Multiple modules can be chained to build application logic

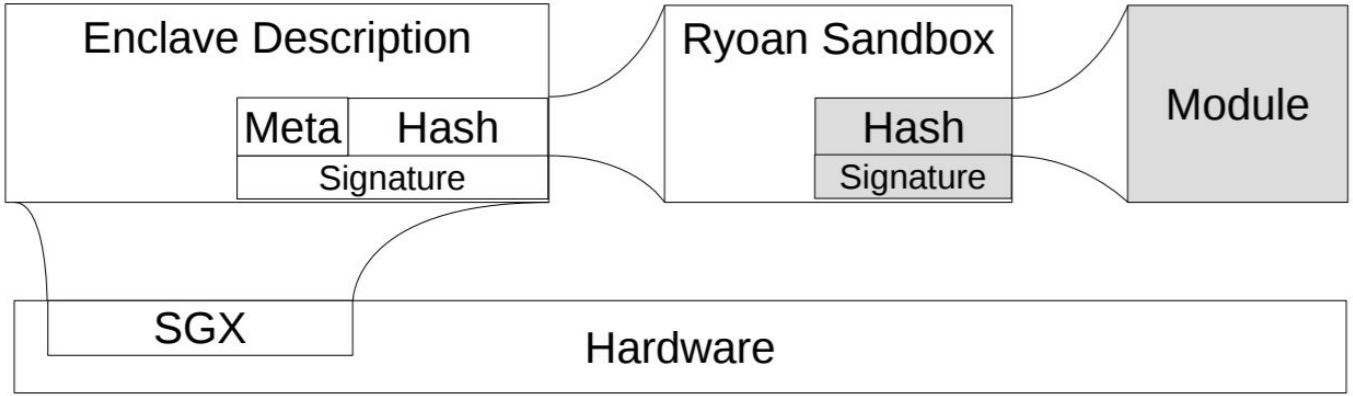
- Similar to DIY system? Discuss

# Ryoan

- Ryoan is a distributed sandbox that executes a DAG (directed acyclic graph) of untrusted modules running on potentially malicious OS.

- Ryoan's main goal is to:
  - Provide user data secrecy without trusting application and the platform
  - Make sure correct code is executed and to prevent modules from leaking sensitive user data.

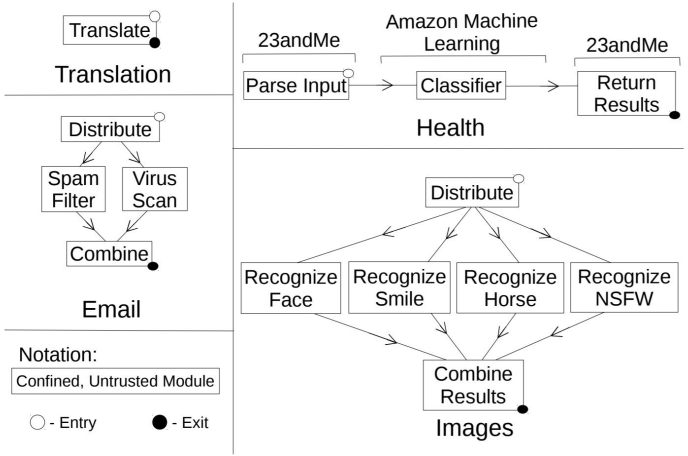
# Ryoan's Chain of Trust



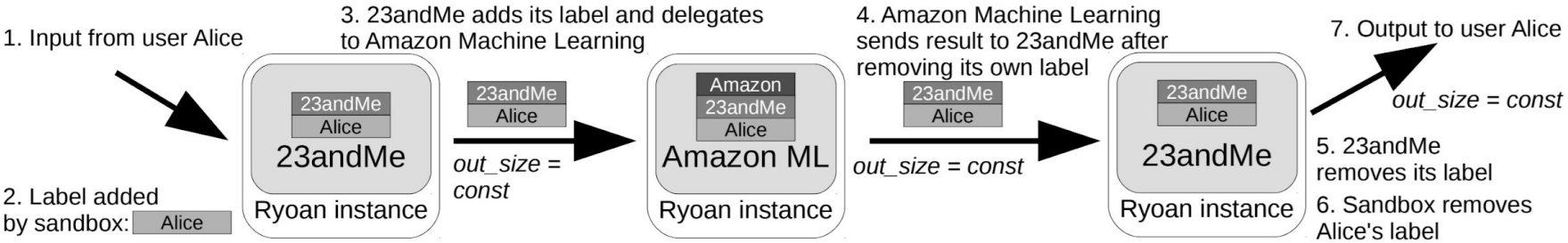

- SGX attests that trustable version of Ryoan sandbox is running on the machine
- Ryoan attests that required module with correct parameters is running in a sandboxed environment

# DAG Generation

1. User can either define or approve a DAG topology of untrusted modules
2. User validates the identity of Primary Ryoan
3. Primary Ryoan requests OS to spawn Ryoan instances with modules in enclave (can be remote or local)
4. Primary uses SGX to attest that Ryoan with correct code is loaded
5. Neighbouring enclaves establish a secure channel over network
6. User validates that correct topology is initialized. Only then she shares her secrets.



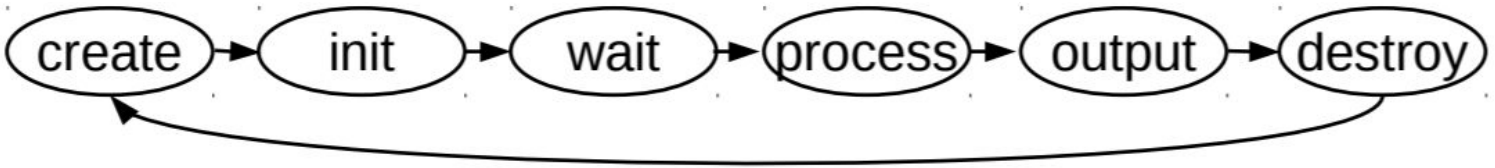
# Workflow




- Any data that has a label that do not belong to that module, will trigger the Ryoan instance to run module in a restrictive environment
- Ryoan instance manages these labels
- What can happen if the 23andMe decides to not put its label?
- Similar to DIFC (Distributed Information Flow Control) in DStar? Discuss

# Module Life Cycle

- For each request module is created, then initialized. And is destroyed when it outputs the data. Flushes out all the state.
  - High overhead?
  - Similar to Lambda Instances?



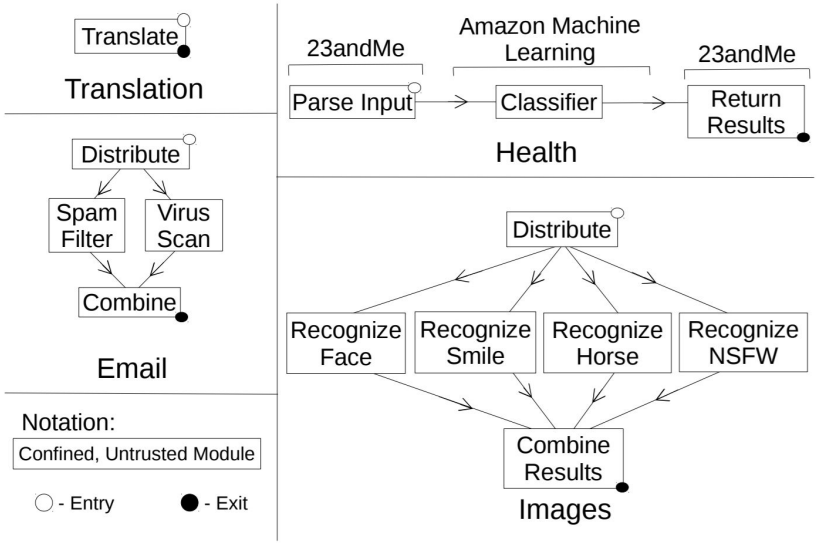
# Some questions

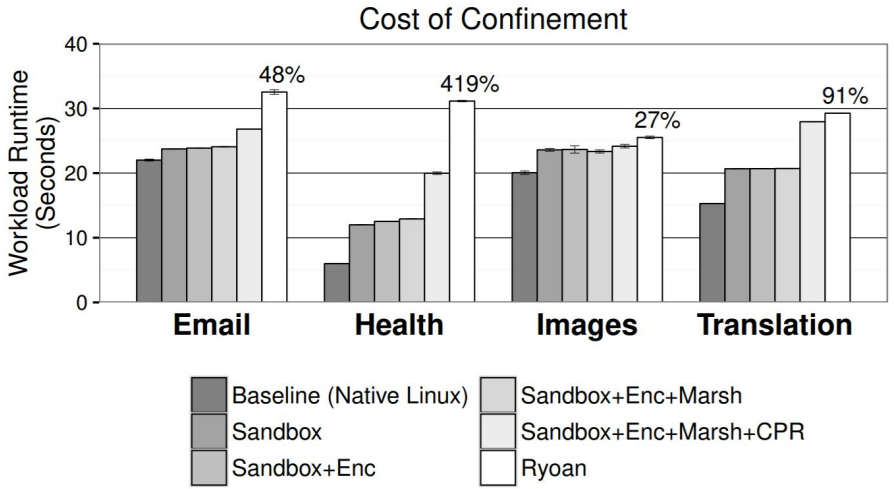
- Who provides guarantee that host OS can not read off secrets from memory?
- What allows the initial code to be verified?
- Module can not write user secrets on arbitrary memory location?
- Module can not modulate information on covert side channels such as system calls?
- Module can not collude with other users to read off secrets of some other user?
- What guarantees do the labels provide?

# Backwards compatibility and Performance

- Ryoan in memory file store, that provides POSIX like API
- Dynamic Memory, mmap call available but it returns a memory chunk from pre-allocated memory
- Module Checkpointing, since module re-initialization is expensive

# Evaluation


Translate
Translation
Distribute
Spam Filter
Virus Scan
Combine
Email
Notation:
Confined, Untrusted Module
- Entry
- Exit
23andMe
Amazon Machine Learning
23andMe
Parse Input
Classifier
Return Results
Health
Distribute
Recognize Face
Recognize Smile
Recognize Horse
Recognize NSFW
Combine Results
Images


Cost of Confinement
Workload Runtime (Seconds)
0
10
20
30
40
48%
419%
27%
91%
Email
Health
Images
Translation
Baseline (Native Linux)
Sandbox
Sandbox+Enc
Sandbox+Enc+Marsh
Sandbox+Enc+Marsh+CPR
Ryoan

# Discussion

- How this model stacks up against other models we have seen so far? In terms of:
  a. Comprehensiveness of the threat model
  b. System complexity (Ryoan's complexity)
  c. Ease of programming a system
  d. Restrictiveness (programming model wise)
  e. Performance overhead
  f. Support for generic web applications
  g. Stateful applications support

- Would you use Ryoan to deploy your applications?