HasTEE

Programming Trusted Execution Environments with Haskell

Authors

Abhiroop Sarkar

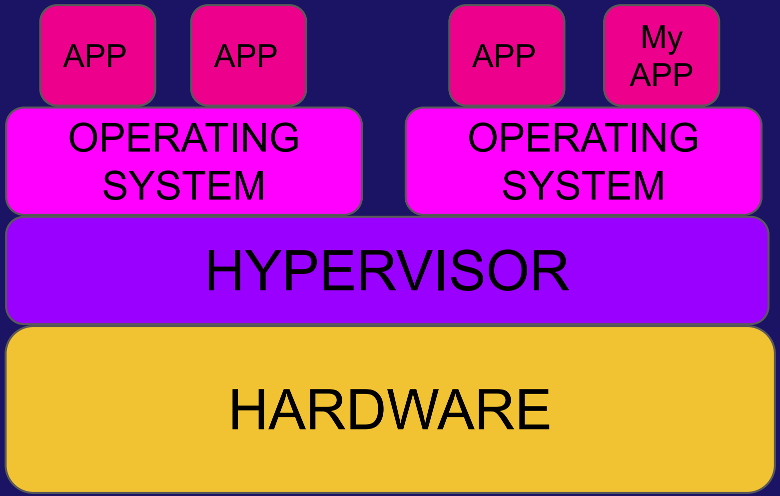
Robert Krook

CHALMERS
UNIVERSITY OF TECHNOLOGY

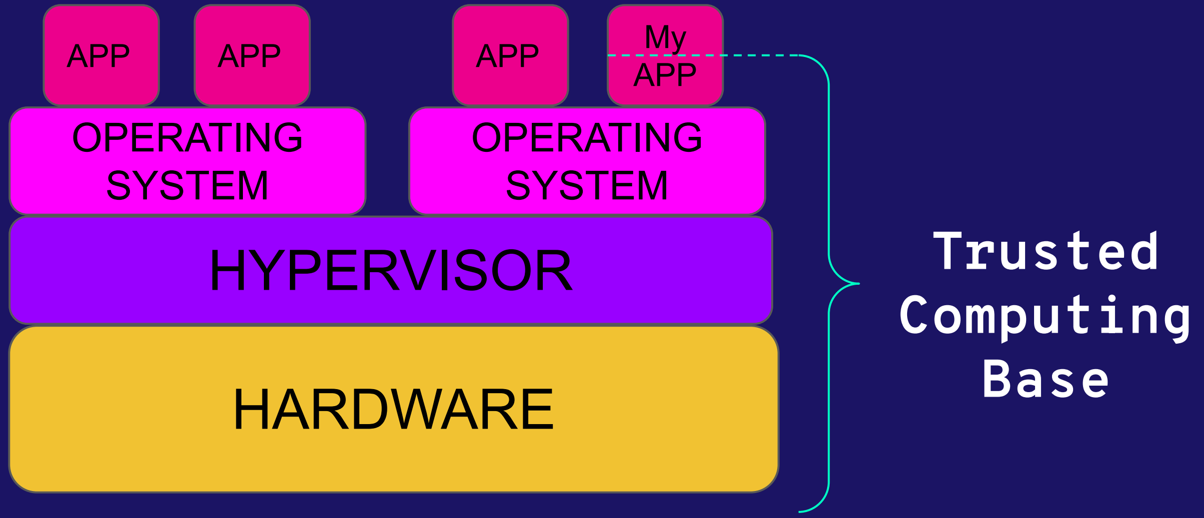
Alejandro Russo

Koen Claessen

Cloud Deployments



Cloud Deployments



OS Vulnerabilities

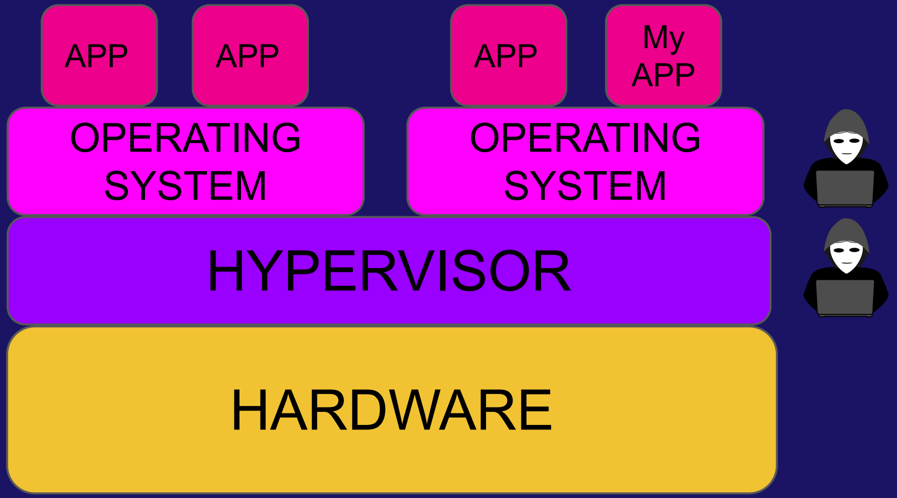
Vulnerability	Total	core	drivers	net	fs	sound
Missing pointer check	8	4	3	1	0	0
Missing permission check	17	3	1	2	11	0
Buffer overflow	15	3	1	5	4	2
Integer overflow	19	4	4	8	2	1
Uninitialized data	29	7	13	5	2	2
Null dereference	20	9	3	7	1	0
Divide by zero	4	2	0	0	1	1
Infinite loop	3	1	1	1	0	0
Data race / deadlock	8	5	1	1	1	0
Memory mismanagement	10	7	1	1	0	1
Miscellaneous	8	2	0	4	2	0
Total	141	47	28	35	24	7

Figure 2: Vulnerabilities (rows) vs. locations (columns).

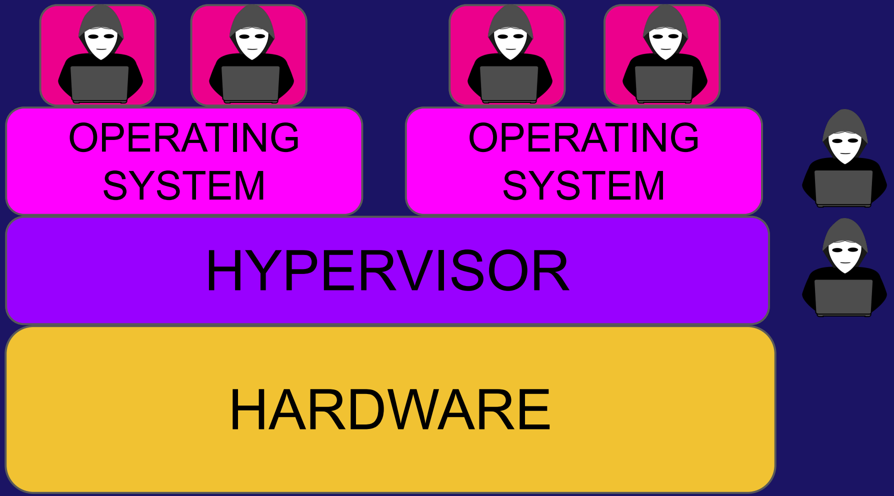
Linux kernel vulnerabilities: State-of-the-art defenses and open problems. Mao et al. In *Proceedings of the Second Asia-Pacific Workshop on Systems* (pp. 1-5).

Characterizing hypervisor vulnerabilities in cloud computing servers. Perez-Botero et al. In *Proceedings of the 2013 international workshop on Security in cloud computing*.

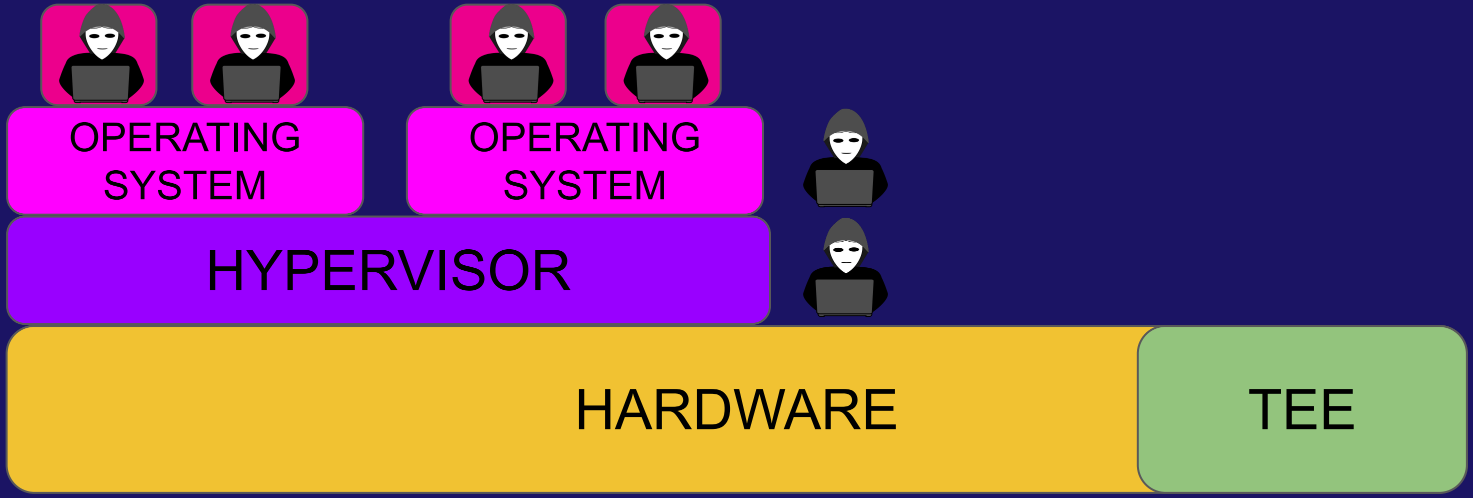
Cloud Deployments



Cloud Deployments



Trusted Execution Environment (TEE)



Trusted Execution Environment (TEE)

intel SGX

arm TRUSTZONE

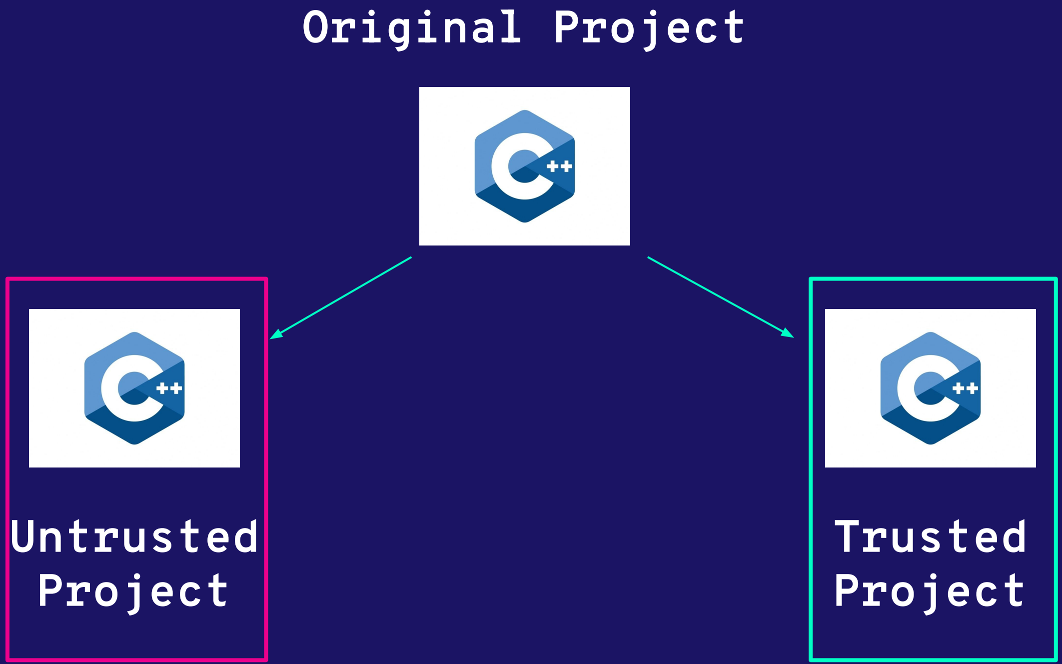
AMD SEV-SNP

RISC-V

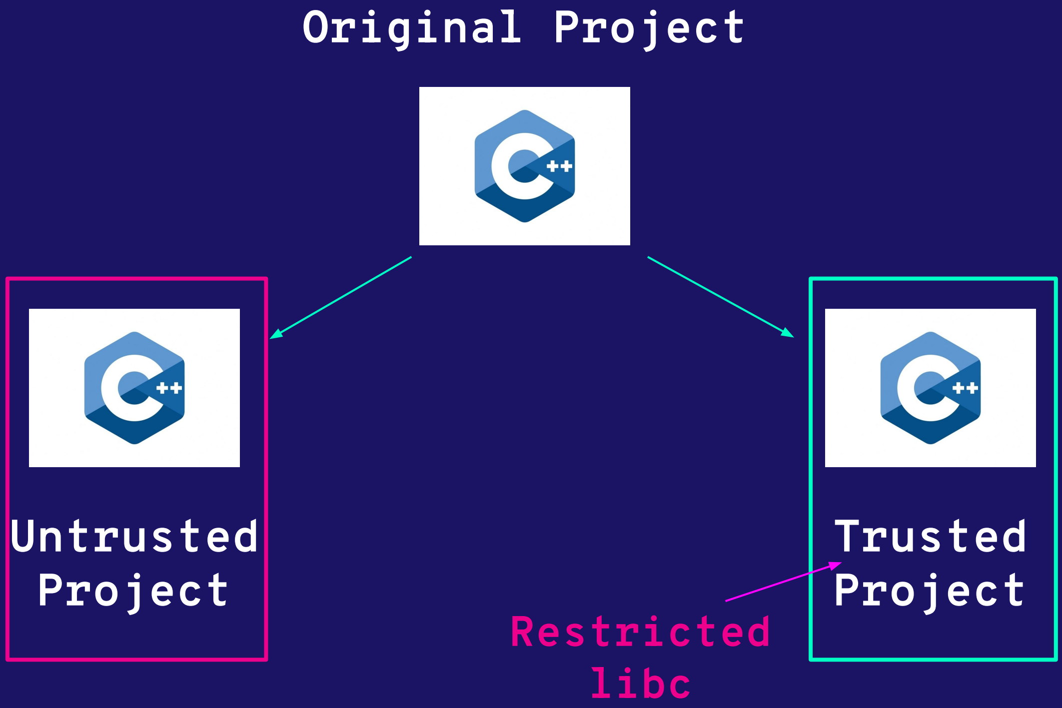
Physical Memory Protection

Programming TEEs

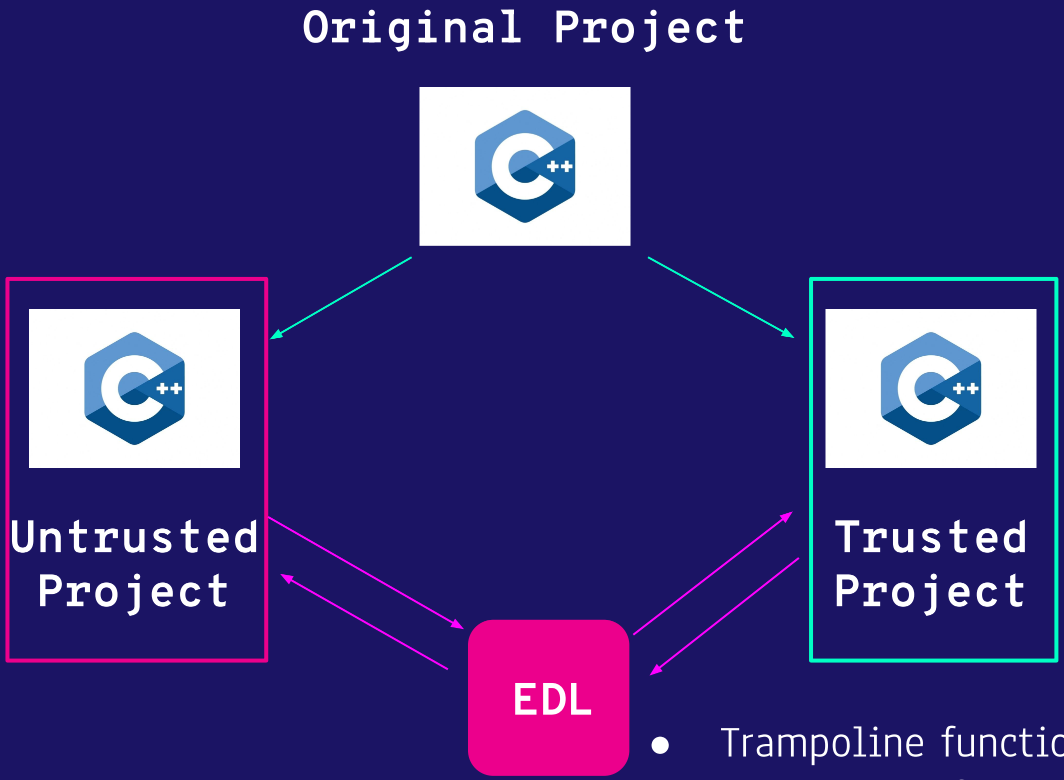
Programming TEEs



Programming TEEs



Programming TEEs



- Trampoline functions
- Arcane Makefiles
- Complex data copying protocol

ALTERNATE APPROACHES

Java

Secured Routines:
Language-based Construction of Trusted Execution Environments

Abstract

Trusted Execution
enclaves, use hard
tegrity of operatio
is available on ma
gramming model a

Golang

Language Support for Secure Software Development with Enclaves

Aditya Oak
TU Darmstadt

Amir M. Ahmadian
KTH Royal Institute of Technology

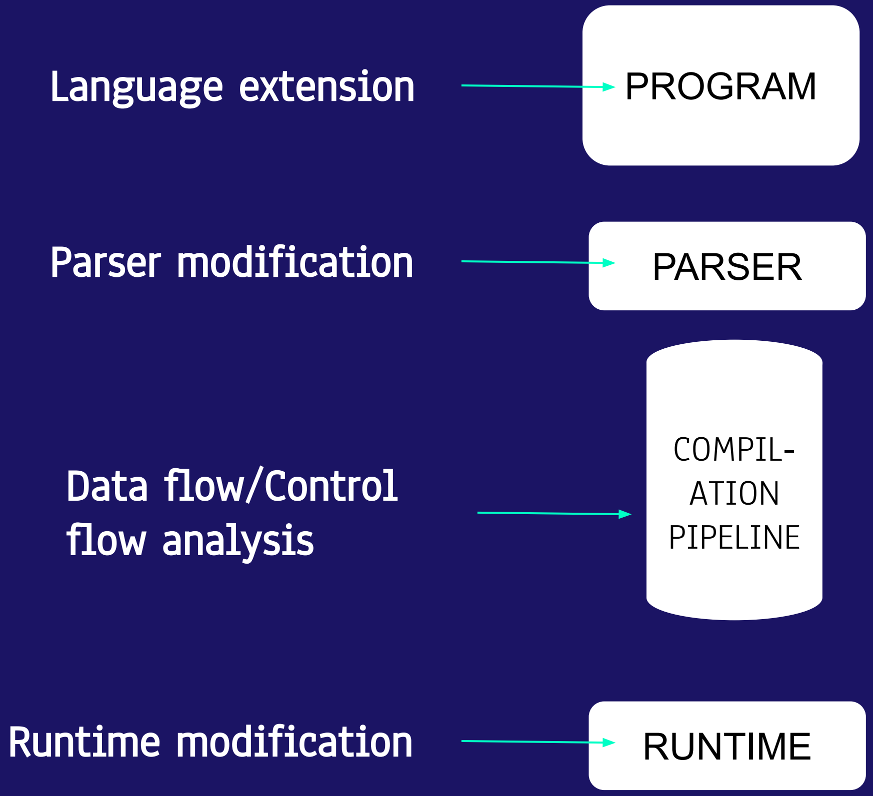
Musard Balliu
KTH Royal Institute of Technology

Guido Salvaneschi
University of St.Gallen

***Abstract*—Confidential computing is a promising technology for securing code and data-in-use on untrusted host machines, e.g., the cloud. Many hardware vendors offer different implementations of Trusted Execution Environments (TEEs). A TEE is a hardware protected execution environment that allows performing confidential computations over sensitive data on untrusted hosts. Despite the appeal of achieving strong security guarantees against low-level attackers, two challenges hinder the adoption of TEEs. First, developing software in high-level managed languages, e.g., Java or Scala, taking advantage of existing TEEs is complex and error-prone. Second, partitioning**

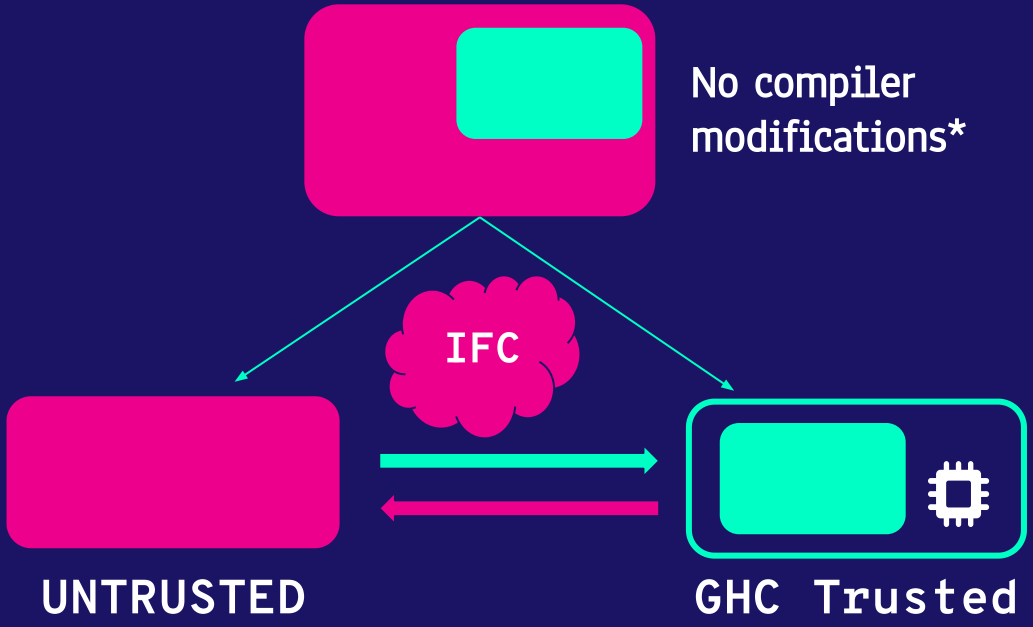
First, **seamless integration** of enclave programming into software applications remains challenging. For example, Intel provides a C/C++ interface to the SGX enclave but no direct support is available for managed languages. As managed languages like Java and Scala are extensively used for developing distributed applications, developers need to either interface their programs with the C++ code executing in the enclave (e.g., using the Java Native Interface [12]) or compile their programs to native code (e.g., using Java Native [13])

Golang & Java APPROACHES



HasTEE

HasTEE



*Ekblad, A. and Claessen, K. A seamless, client-centric programming model for type safe web applications. Haskell Symposium, 2014.

HasTEE Key Contributions

- Automatic Partitioning with no compiler modifications
- Program in a high-level language - Haskell
- Enforce Information Flow Control on data within enclaves

Illustration : Password Checker

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd
```

The Enclave monad

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
```

```
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
```

The App monad

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
```

```haskell
inEnclaveConstant :: a → App (Enclave a)
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
```

```haskell
inEnclave :: (Securable a) => a → App (Secure a)
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
```

The Client monad

```
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
```

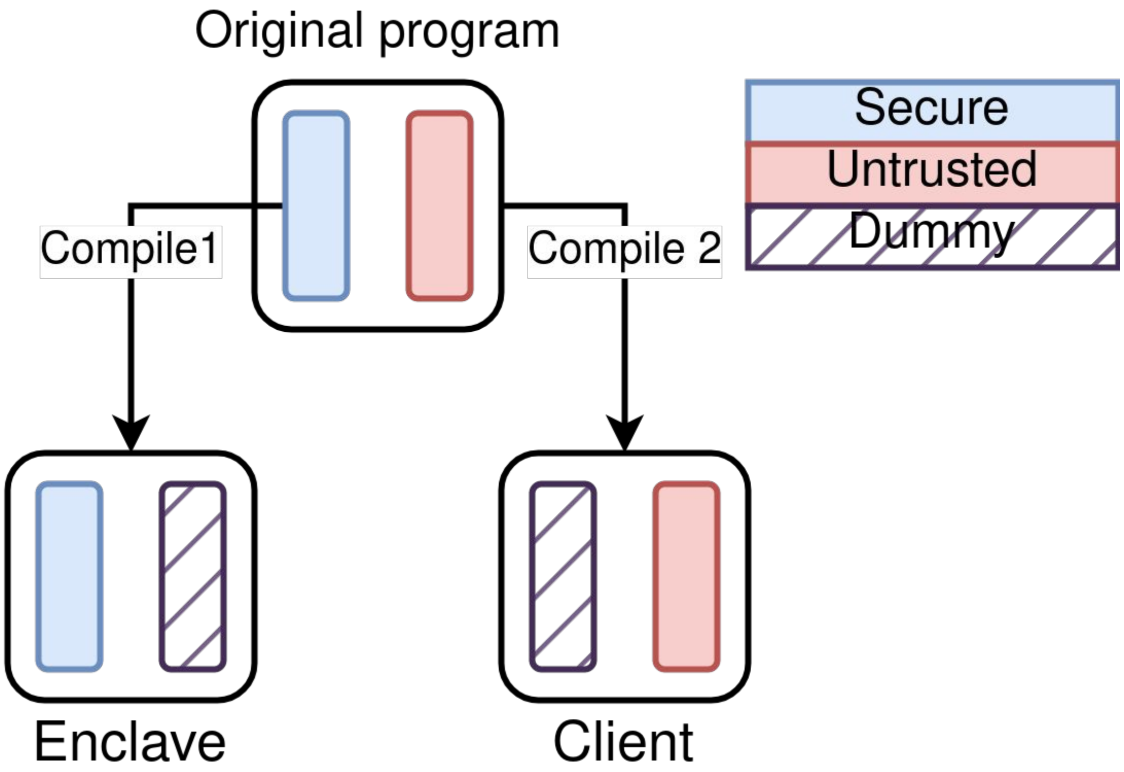
```haskell
(<@>)      :: (Binary a) => Secure (a → b) → a → Secure b
gateway    :: (Binary a) => Secure (Enclave a) → Client a
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)
```

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

main = runApp passwordChecker
```

COMPILED TWICE

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

main = runApp passwordChecker
```


Original program
Secure
Untrusted
Dummy
Compile1
Compile 2
Enclave
Client

```haskell
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

main = runApp passwordChecker
```

Compilation 1

```haskell
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

Compilation 2

```haskell
-- Client
pwdChkr = -- gets optimised away

passwordChecker :: App Done
passwordChecker = do
  passwd <- return Dummy
  efunc <- inEnclave $ -- ignores pwdChkr body
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

-- drives the application
main = runApp passwordChecker
```

Compilation 1

```haskell
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

Compilation 2

```haskell
-- Client
pwdChkr = -- gets optimised away

passwordChecker :: App Done
passwordChecker = do
  passwd <- return Dummy
  efunc <- inEnclave $ -- ignores pwdChkr body
  runClient $ do -- Client code
    liftIO  $ putStrLn "Enter your password"
    userInput <- liftIO getLine
    res       <- gateway (efunc <@> userInput)
    liftIO $ putStrLn ("Login returned " ++ show res)

-- drives the application
main = runApp passwordChecker
```

Compilation 1

```haskell
-- Enclave
pwdChkr :: Enclave String -> String -> Enclave Bool
pwdChkr pwd guess = fmap (== guess) pwd

passwordChecker :: App Done
passwordChecker = do
  passwd <- inEnclaveConstant "secret"
  efunc <- inEnclave $ pwdChkr passwd
  return DONE

-- wait for calls from Client
main = runApp passwordChecker
```

GHC Trusted

INTEL SGX

Information Flow Control

Information Flow Control

Information Flow Control

```
gateway :: (Binary a) => Secure (Enclave a) → Client a
```

Information Flow Control

```
gateway :: (Binary a) => Secure (Enclave a) → Client a
```

Lack of a Binary instance
prevents accidental leaks

Enclave a

Does not instantiate MonadIO but RestrictedIO

```
type RestrictedIO m = (RandomIO m, FileIO m, ..)

class FileIO m where
  readFile :: FilePath -> m String

class RandomIO m ...
```

Information Flow Control

```
gateway :: (Binary a) => Secure (Enclave a) → Client a
```

Enclave monad restricted
using a RestrictedIO typeclass

Non-interference Proposition

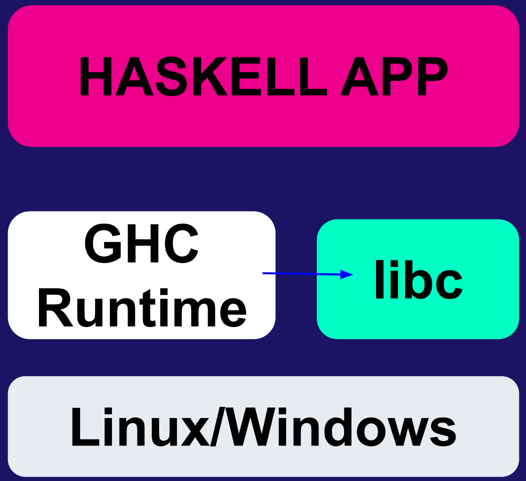
p :: Enclave a -> App Done
p has no `gateway` operation

e1 :: Enclave a

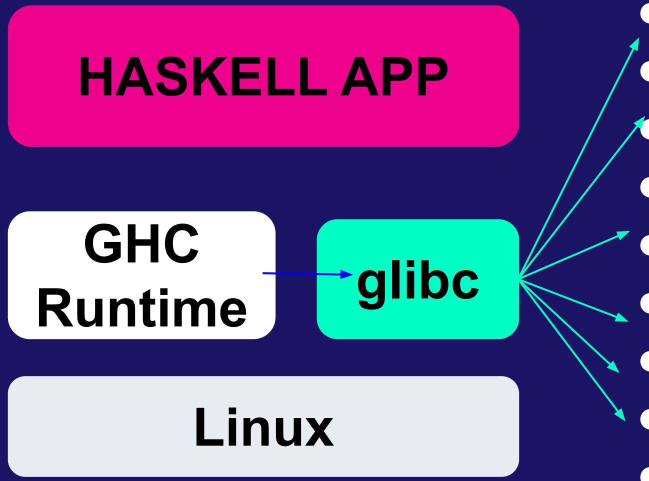
e2 :: Enclave a

p e1 $\underset{\text{side effect}}{\approx}$ **p e2**

IMPLEMENTATION

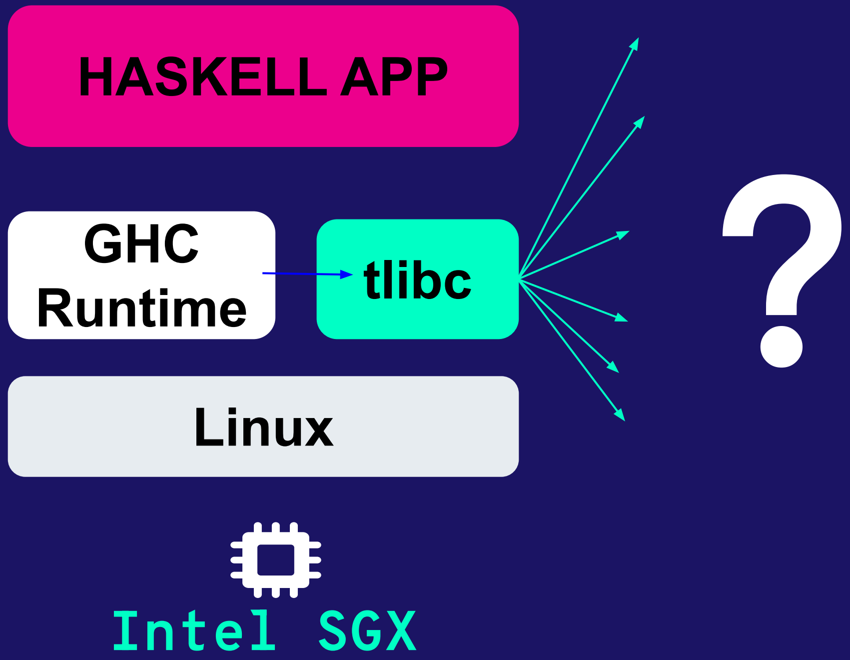


IMPLEMENTATION

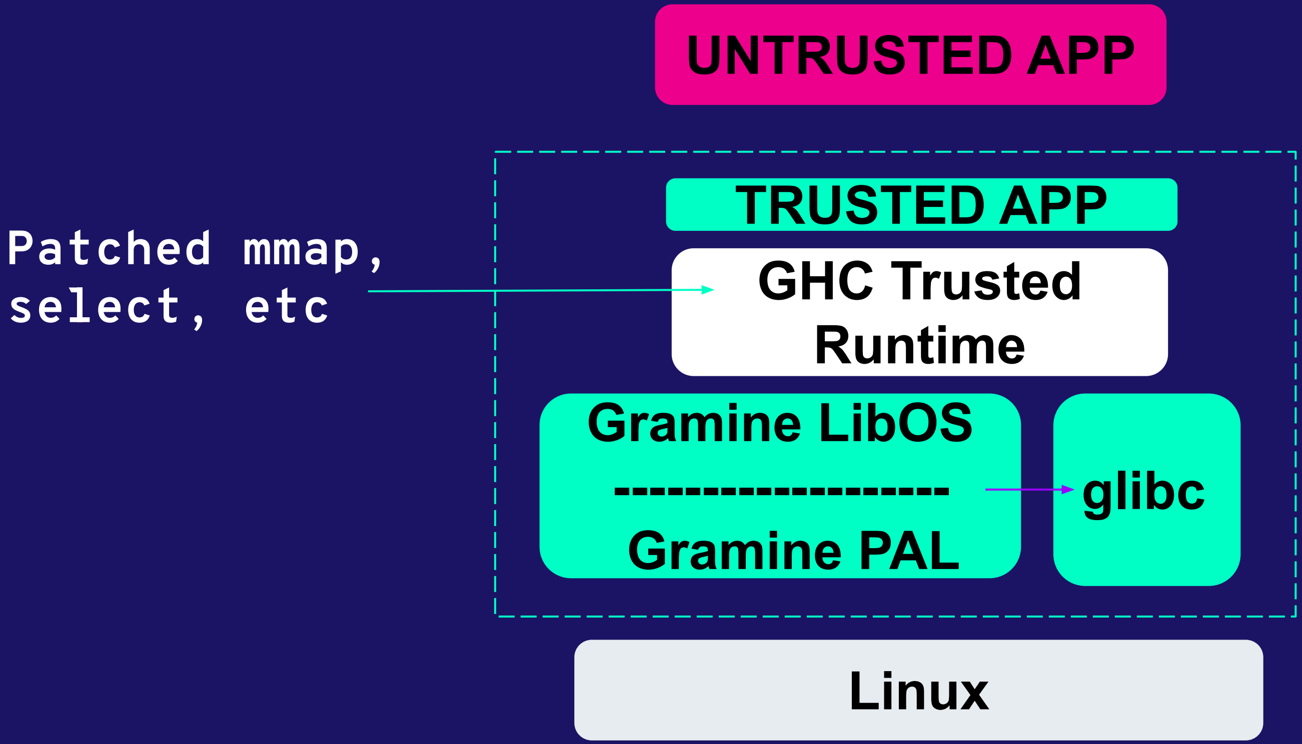


- mmap
- madvise
- getrusage
- poll
- select
- clock apis
- pthread apis
- libm apis
- …

IMPLEMENTATION

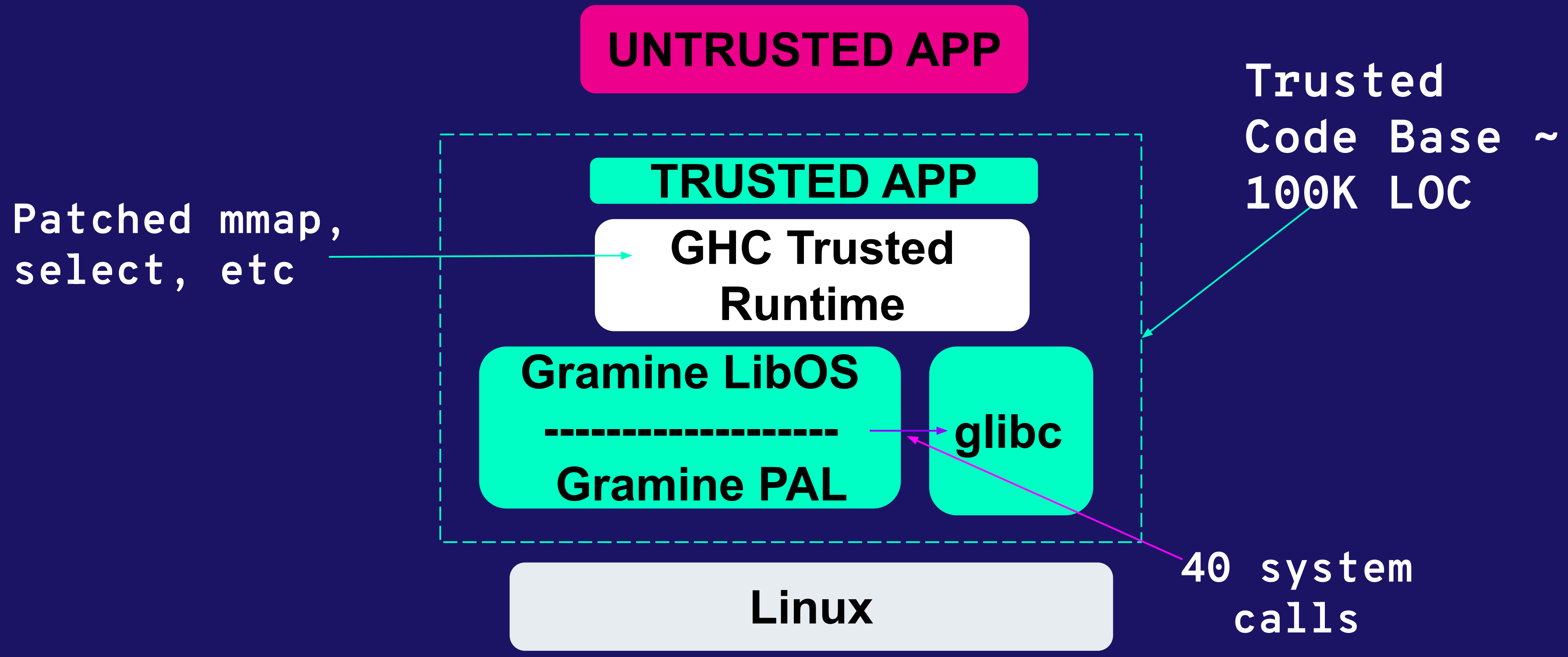


IMPLEMENTATION

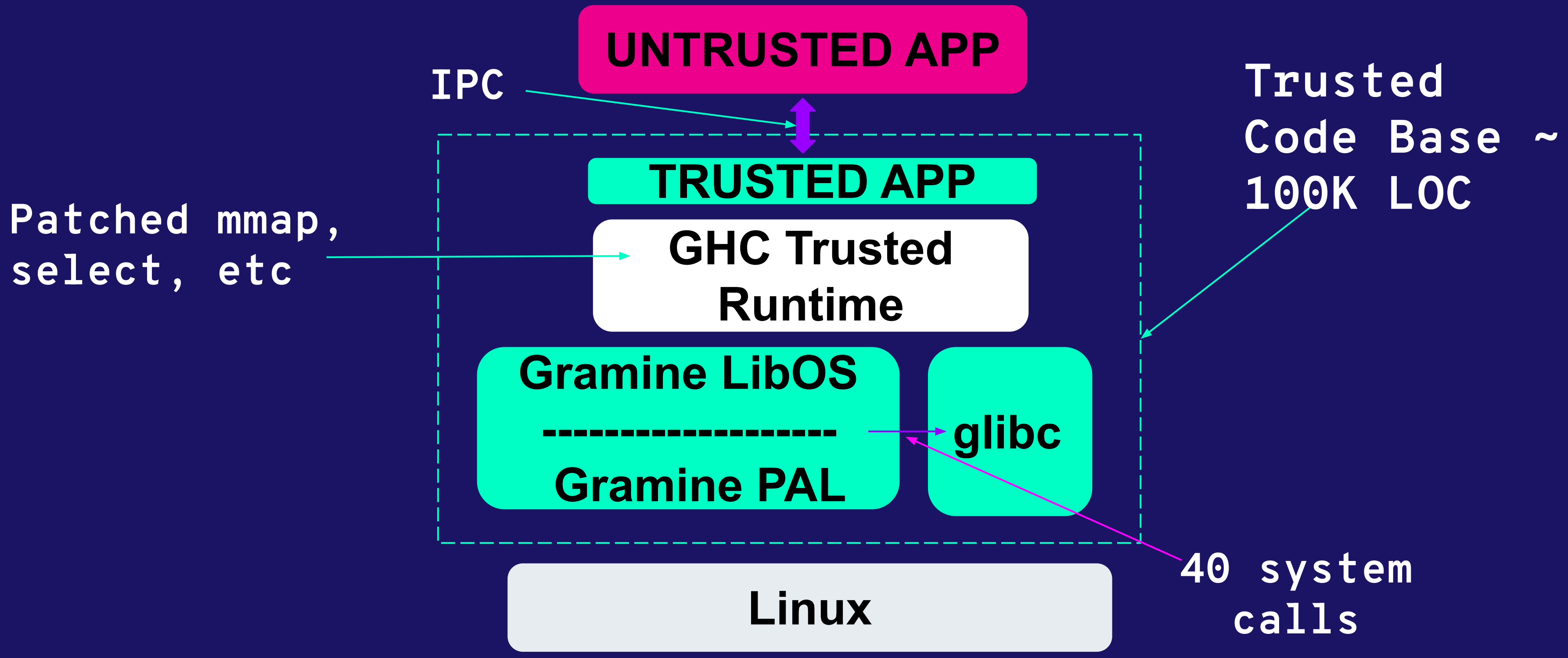


https://gramineproject.io/

IMPLEMENTATION



IMPLEMENTATION



PERFORMANCE

Memory	RSS	Virtual Size	Disk Swap
At rest	19,132 KB	287,920 KB	0 KB
Peak	20,796 KB	290,032KB	0 KB

PERFORMANCE

Memory	RSS	Virtual Size	Disk Swap
At rest	19,132 KB	287,920 KB	0 KB
Peak	20,796 KB	290,032KB	0 KB

Enclave Page Cache
size = 93MB

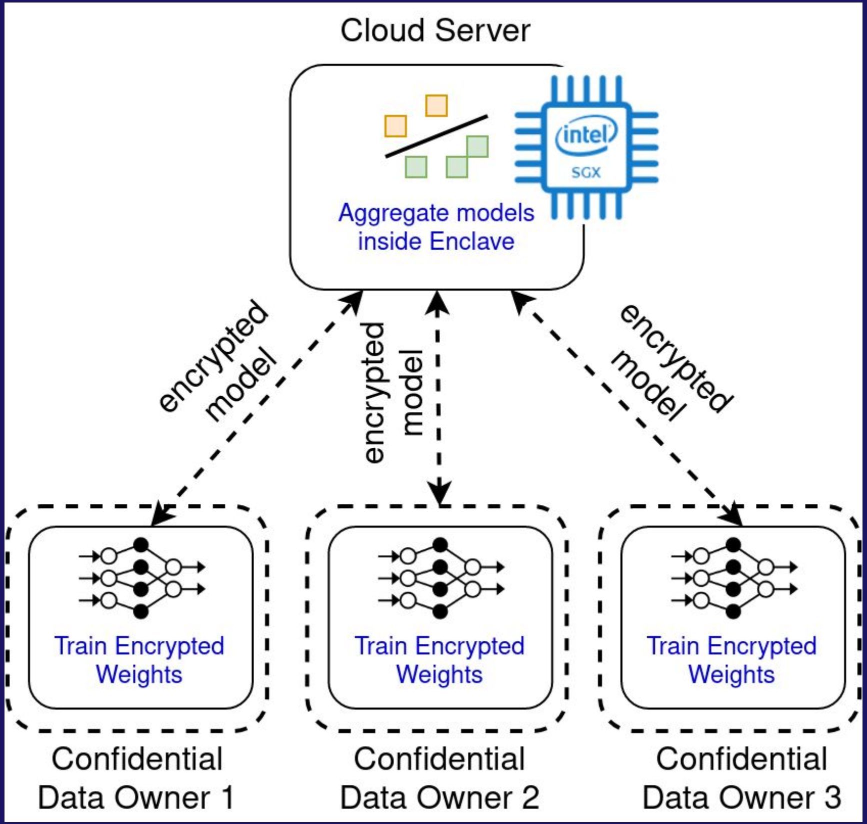
PERFORMANCE

Memory	RSS	Virtual Size	Disk Swap
At rest	19,132 KB	287,920 KB	0 KB
Peak	20,796 KB	290,032KB	0 KB

LATENCY ~ 60 ms
vs
0.6 ms in native SDK

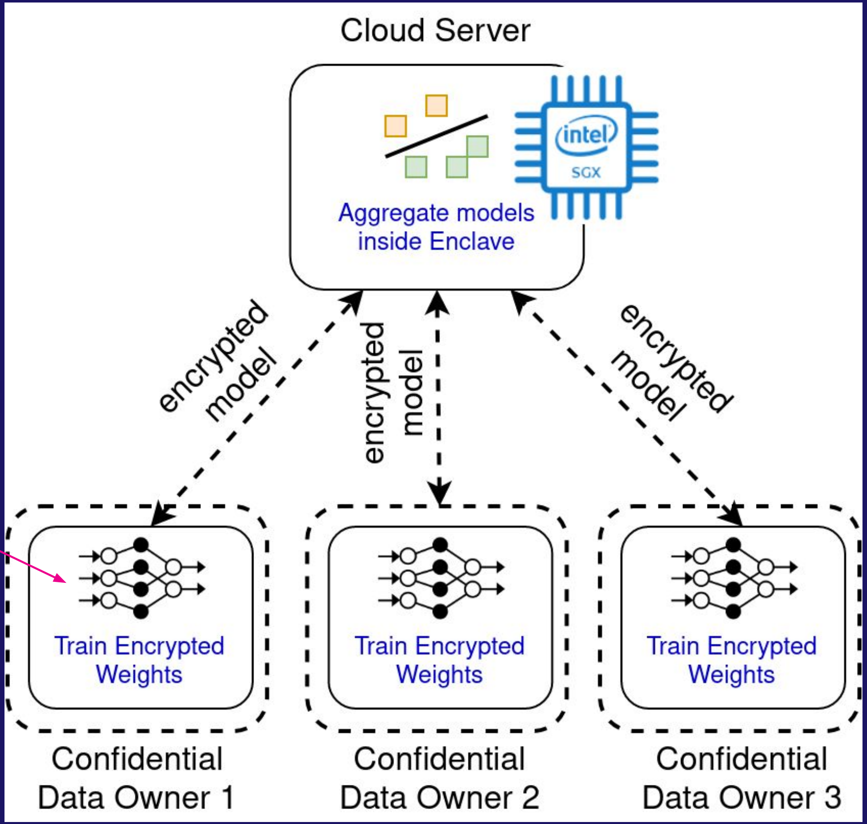
Applications

Zero Trust Federated Learning



Zero Trust Federated Learning

Uses homomorphic encryption for training

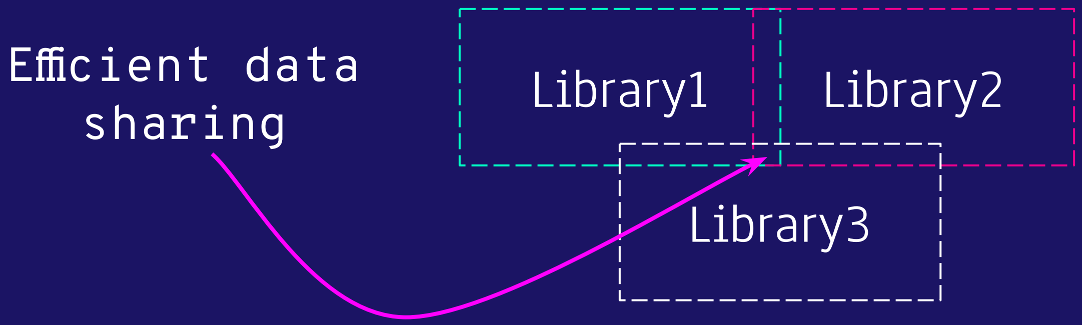


Applications

- Privacy-preserving Federated Learning
- Encrypted Password Wallet
- Data Clean Room with Differential Privacy

FUTURE WORK

FUTURE WORK



CHERI Hardware Compartmentalisation

FUTURE WORK

GHC/Haskell

GHC Runtime

CHERI

Requires substantial overhaul

THANKS!

https://github.com/Abhiroop/EnclaveIFC