

A Type System for Checking Applet Isolation in Java Card

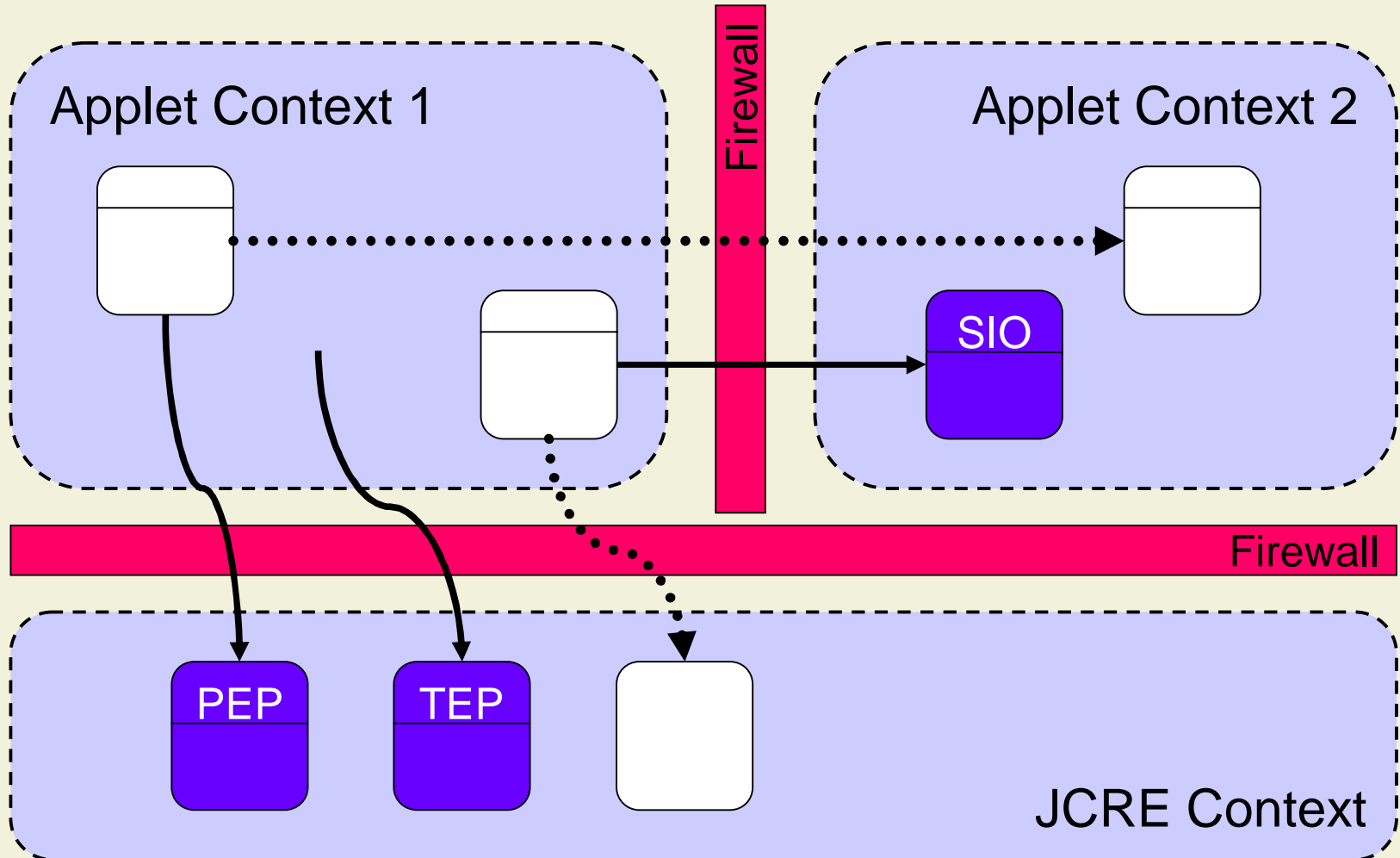
Peter Müller
ETH Zürich

Joint work with Werner Dietl and Arnd Poetzsch-Heffter

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Applet Isolation



Example

```
class Status {  
    ...  
    boolean isSuccess( ) { ... }  
}
```

```
interface Service extends Shareable {  
    Status doService( );  
}
```

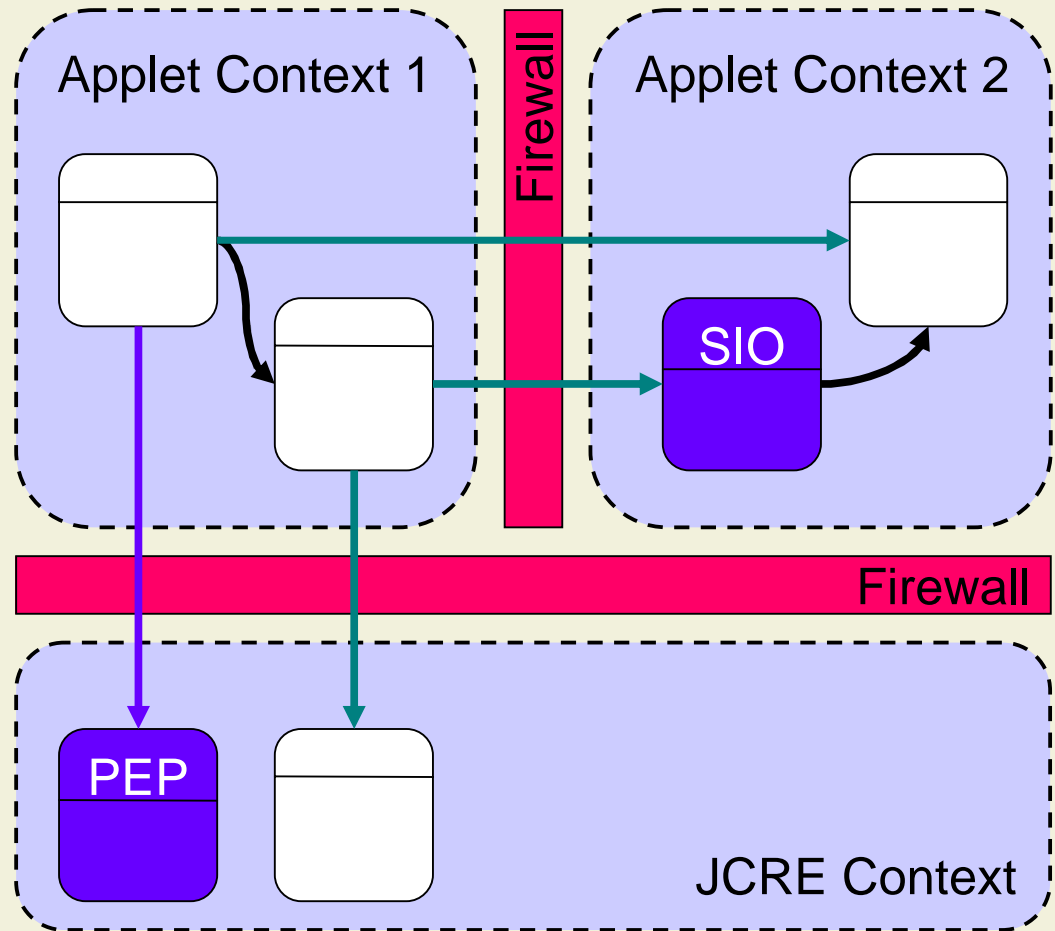
```
class Client extends Applet {  
    ...  
    void process( APDU apdu ) {  
        AID          server = ...;  
        Shareable    s =  
            JCSystem.getAppletShareableInterfaceObject( server, (byte) 0 );  
        Service      service = ( Service ) s;  
        Status       status = service.doService( );  
        if ( status.isSuccess( ) ) { ... } // SecurityException raised  
    } }
```

Motivation

- Formal program verification
 - **Prove absence of SecurityExceptions** for many kinds of expressions
 - Firewall property causes **significant overhead** for specifications and proofs
- Objective
 - **Check** applet isolation **statically**
 - Develop a solution for **source programs**
 - Build on experience with **ownership** and the Universe Type System

Approach

- Use **type system** to classify references to
 - Objects in the same context
 - Objects in any contexts
 - Entry points
- Perform **static checks** to enforce applet isolation



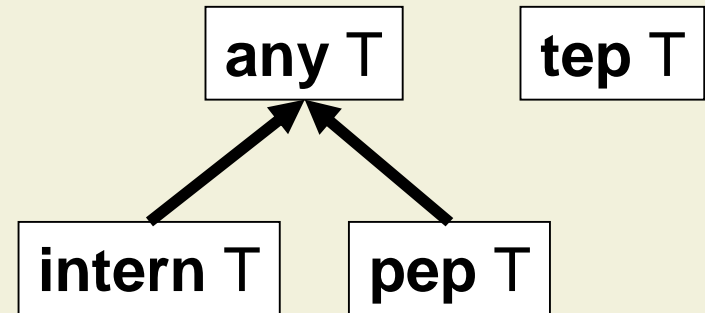
Tagged Types

- Tags
 - **intern**: References within a context
 - **any**: References to any context
 - **pep**: References to permanent entry points
 - **tep**: References to temporary entry points and global arrays

- Tagged types **specify the context** a reference may point into
 - Tagged types are tuples: Tag \times Type, e.g., **intern T**

Type Rules

- **intern** and **tep** types are **subtypes** of the corresponding **any** types



- Type rules for tagged types follow Java's type rules

```

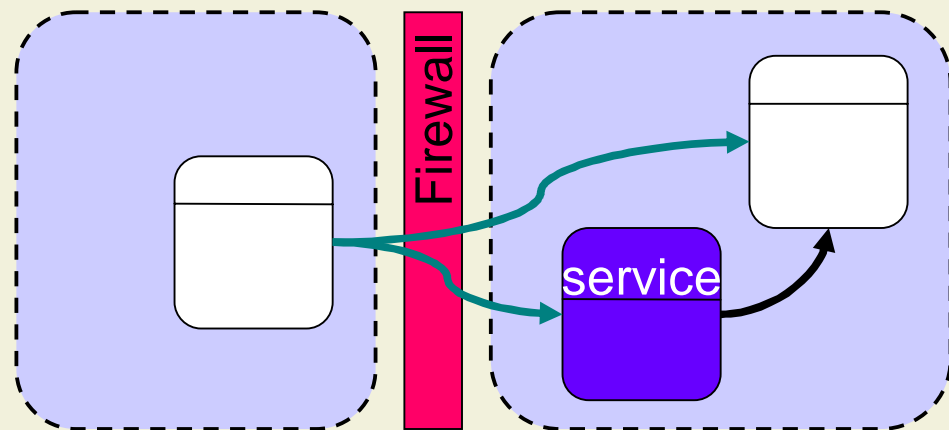
void process( tep APDU apdu ) {
    intern AID      server = ...;
    any Shareable s =
        JCSystem.getAppletShareableInterfaceObject( server, (byte) 0 );
    any Service    service = ( any Service ) s;
    ?? Status      status = service.doService( );
    if ( status.isSuccess( ) ) { ... }
}
  
```

Method Invocations

- Tag intern specifies context **relatively** to the current context
- For method invocations, parameter and result types have to be interpreted **relatively** to the tag of the target

```
interface Service extends Shareable {
    intern Status doService( );
}
```

```
any Service service = ...;
any Status status = service.doService( );
```



Type Combinations

- Type combinator *

$$(H,T)^*(G,S) = \begin{cases} (\mathbf{any},S) & \text{if } H \neq \mathbf{intern} \text{ and } G = \mathbf{intern} \\ (G,S) & \text{otherwise} \end{cases}$$

- Type rule for method invocations

$$\frac{\vdash e1 :: (H,T) , \vdash e2 :: (G,S) , (H,T)^*(G,S) <: (F_P,T_P)}{\vdash e1.m(e2) :: (H,T)^*(F_R,T_R)}$$

Dynamic Type Checks

■ Casts

- Downcasts from **any** types to corresponding **intern** and **pep** types require dynamic checks
- In practice only necessary for static fields (no **intern** tag)
- Casts may throw `SecurityException`

■ Covariant arrays

- **intern** `T[]` and **pep** `T[]` are **not subtypes** of **any** `T[]`
- **Avoid dynamic check** for assignments to array slots

Static Firewall Checks

- Method invocation $e.m(\dots)$
 - (H, T) is the static tagged type of e
 - If H is **any**, T has to be an interface that extends **Shareable**

- Field access $e1.f = e2$
 - Static type of $e1$ must have tag **intern**
 - Static type of $e2$ must not have tag **tep**

Example Revisited

```

class Status {
  ...
  boolean isSuccess( ) { ... }
}

```

```

interface Service extends Shareable {
  intern Status doService( );
}

```

```

class Client extends Applet {
  ...
  void process( tep APDU apdu ) {
    intern AID      server = ...;
    any Shareable  s =
      JCSystem.getAppletShareableInterfaceObject( server, (byte) 0 );
    any Service    service = ( any Service ) s;
    any Status     status = service.doService( );
    if ( status.isSuccess( ) ) { ... }    // Static type error
  } }

```

Results

- Type Safety
 - All references are **correctly tagged**
 - Proof by rule induction based on operational semantics
- Applet Isolation
 - Lemma: Each Java Card program with tagged types that passes the static checks behaves like the corresponding program with dynamic checks
 - Every Java Card program that can be correctly tagged **does not throw SecurityExceptions** (except for casts)
 - Proof by rule induction with two operational semantics (with and without dynamic checks)

Conclusions

- Presented approach supports program verification
 - **Absence of SecurityException** does not have to be shown during verification (except for some casts)
 - Static checking is **modular**

- Security requires
 - Type system on bytecode level
 - Adapted VM / Bytecode verifier
 - Forbidding downcasts from **any** to **intern** or **pep**

Future Work

- Extension of presented work
 - Support for missing language features (exceptions)
 - Annotation of Java Card API

- Formal verification
 - Integration of type system with Universe Type System
 - Implementation in JIVE (Java Interactive Verification Environment)