Security Level 3 A New CORBA Credentials Model

Polar Humenn

2-212 CST
Syracuse, NY 13244-4100
polar@adiron.com

Agenda

- Security Level 2 Credentials
- The Principal Calculus
- Security Level 3 Credentials
- Relationship to the tokens on the wire

Security Level 2 Credentials API

- Credentials Interface
  - Holder of Security Attributes

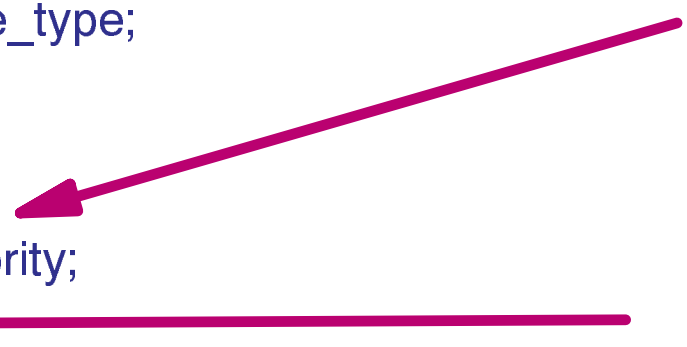
```
interface Credentials {
  AttributeList get_attributes(
    in AttributeTypeList types
  );
};
```

- What is it used for?
  - Access Control
  - Auditing Information

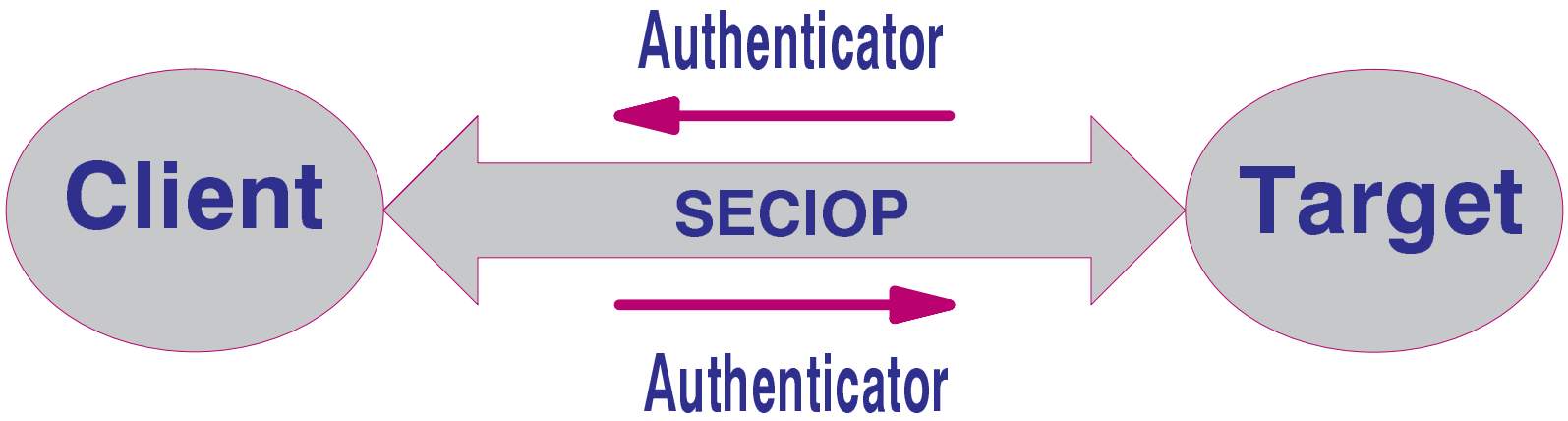
Security Attributes

- Type Value Pairs

```
struct ExtensibleFamily {
    unsigned short     family_definer;
    unsigned short     family;
};

struct AttributeType {
    ExtensibleFamily           attribute_family;
    SecurityAttributeType      attribute_type;
};

struct SecAttribute {
    AttributeType        attribute_type;
    OID                  defining_authority;
    Opaque               value;
};
typedef sequence<SecAttribute> AttributeList;
```



Security Interoperability SECIOP



- Trades Authenticators in Context Establishment
- Authenticators are Mechanism Specific
  - GSS-Kerberos
  - GSS-Sesame
  - SSL, X.509 Certificates

CORBA
Security Service

- Verifies authenticators and presents them to the application as Credentials.

Benefits of the Security Service and Credentials Model

- Security service does the work of authentication and privilege verification.
- Verification results are presented to the application via an API.
- The application need not do its own verification. It believes the security service.

Pitfalls of the Current Model

- Interface and data model is too general
  - Credentials have only single flat list of attributes.
- No comprehensive rules on attributes and the entities they represent.
  - No good way to represent an endorsement or a delegation of identity or privileges.
  - No good way to represent groups of privileges or restrictions on them.

Need A New Model

- Given the pitfalls, we need a comprehensive way to represent authentication, delegation, and privilege information.
- Experience with the Current Model tells me that we need a *FORMAL* model.
- Lets get back to the basics!
- Use Math!

Back To the Basics

- What happened to Principal?
- Principal
  - Abstract Entity that is represented by an identity of a person, a machine, a process, etc.
- Endorsement
  - Act of one Principal giving privilege to another Principal.
- Delegation
  - Act of a Principal giving another Principal **his** identity and/or privileges.

Foundation of the Model is Based on a Formal Calculus

- The Theory of Principals
  - Abadi, et al, A Calculus for Access Control in Distributed Systems, ACM Trans Prog. Lang. and Sys., Oct 1993
  - Polar Humenn, Practical Extensions to the Principal Calculus with Privileges and Resources, *in progress*.

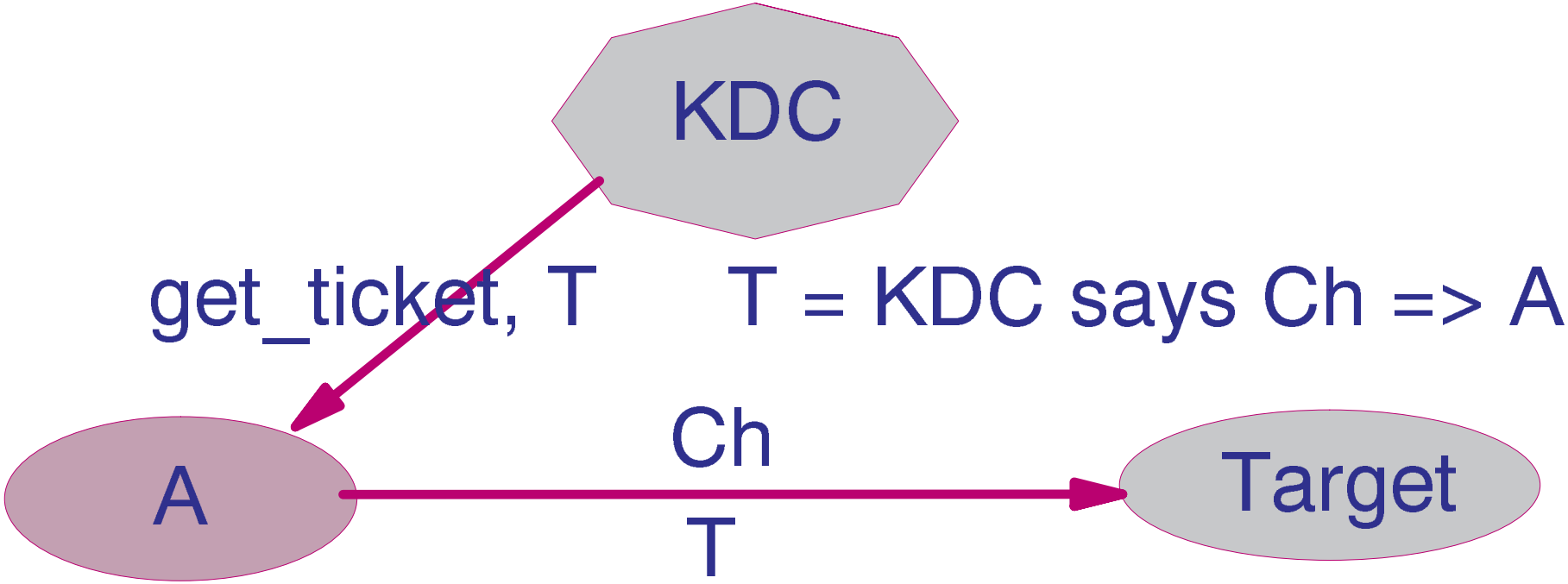
The Principal Calculus

- Principals
  - P
- Quote
  - (P says s)
    where P is a principal and s is a statement
  - A Quote is a statement itself

The Principal Calculus

- Other Statements
  - Speaks For
    - (A => B), A and B are Principals
    - A => B
      if (A says s) then (B says s)
  - Controls
    - (A controls s) A is a Principal, s is a statement
    - A "controls" s
      if (A says s) then s

The Principal Calculus Kerberos



∀ s . Ch controls s
∀ c, x .KDC controls c => x

The Principal Calculus Kerberos

$$\frac{A \textit{ says } req}{Ch \textit{ says } req} \qquad \frac{Ch \Rightarrow A}{Ch \textit{ says } KDC \textit{ says } Ch \Rightarrow A}$$

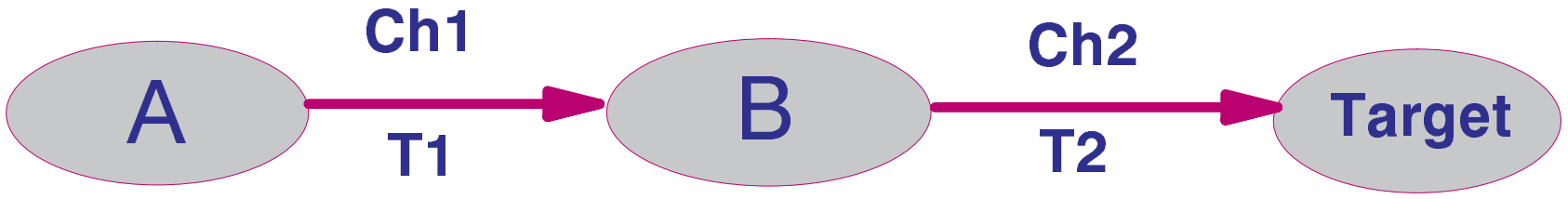
$$\forall\, c, x\ .\ KDC \textit{ controls } c \Rightarrow x$$

$$\forall\, s\ .\ Ch \textit{ controls } s$$

The Principal Calculus Delegation

- Serves Statement
  - (B *serves* A)
    - A and B are Principals
- Quoting Principal
  - B|A, B "quoting" A
    (B|A *says* s) ≡ (B *says* A *says* s)
- For Principal
  - B **for** A, A and B are Principals
  - Combination of Serves and Quoting Principal
  - if (B|A *says* s) /\ (B *serves* A)
    then (B **for** A *says* s)

What about Delegation?



T1 = KDC *says* Ch1 => A
T2 = KDC *says* Ch2 => B
Da = A *says* B *serves* A

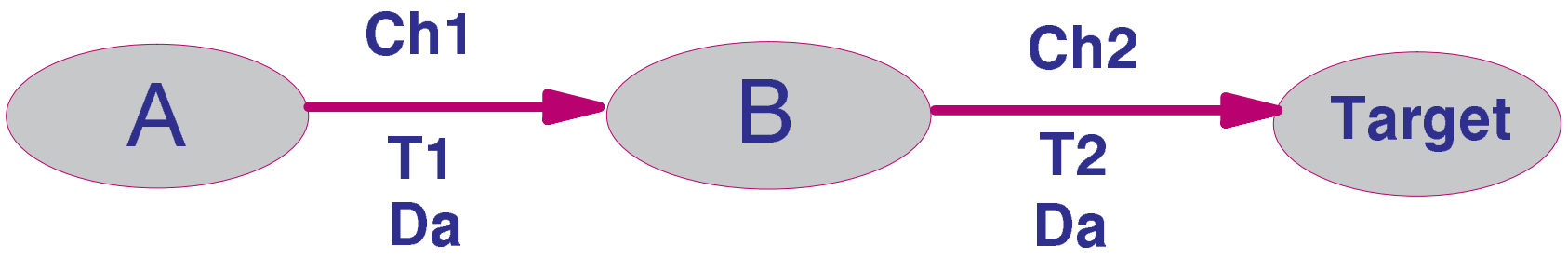
A *says* req

B *says* (A *says* req)
(A says B *serves* A)

Ch2 *says* T2
Ch2 *says* Da
Ch2 *says* A *says* req

∀ c x . KDC *controls* c => x
∀ s . Ch2 *controls* s

What about Delegation?



T1 = KDC *says* Ch1 => A
T2 = KDC *says* Ch2 => B
Da = A *says* B *serves* A

A *says* req

(B for A) *says* s

(B|A) *says* req
A *says* (B *serves* A)

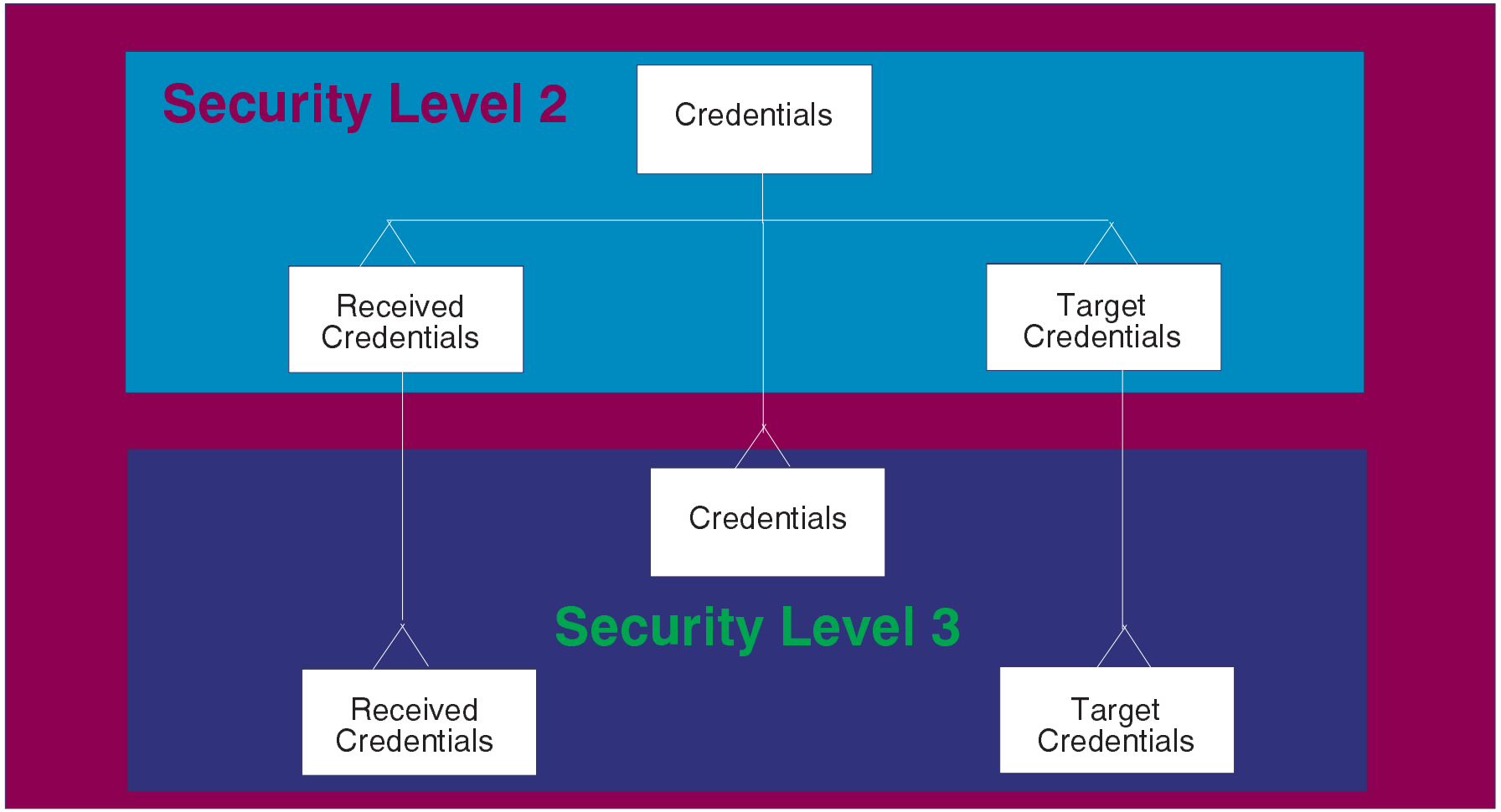
B *says* (A *says* req)
A *says* (B *serves* A)

∀ b . A *controls* (b *serves* A)

The Compound Principal

- The For Principal signifies a delegation
  - B **for** A
    - means B on behalf of A
    - Access control would be performed on A.
    - However, others can write access control policy on B for A as opposed to A
- Delegation Chains
  - C **for** (B **for** A)
    - Access control can still be performed on just A.

Security Level 3 Credentials

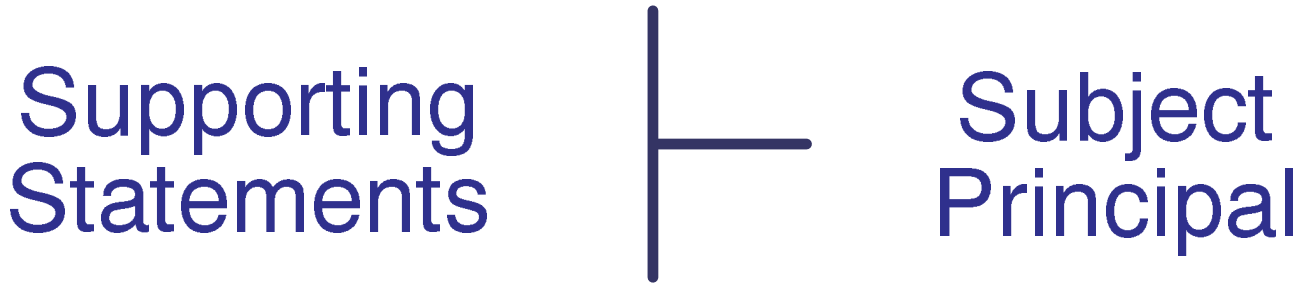


Security Level 3 Credentials

```
interface Credentials :
                SecurityLevel2::Credentials {

   readonly attribute Principal
                      subject;

   readonly attribute StatementList
                      supporting_statements;

};
```

Credentials
Principals

- Principal
  - An entity that is represented as a name that is derived from some verified evidence.
- Statement
  - Representation of the evidence
- Proof



Evidence Statements

- Statements
  - Authentication K *says* Ch => A
  - Identity A *says* req
  - Endorsement T *says* (B *serves* A)
  - Embedded A *says* (s1,...,sn)

Simple Principal

Authen: S *says* Ch => P |- P

```
valuetype Principal {
   PrincipalType         type;
   PrincipalName         name;
   AttributeList         environmental_attributes;
   AttributeList         privileges;
   ReourceNameList  restricted_resources;
};

valuetype SimplePrincipal : Principal {
   PrincipalNameList  alternate_names;
};
```

Delegation
The Compound Principal

KDC *says* Ch => B,
Ch *says* B *says* r, C *says* B *serves* C |- B for C *says* r

```
valuetype ForPrincipal : Principal {
    readonly attribute Principal speaking;
    readonly attribute Principal speaks_for;
};
```

KDC *says* Ch => A, Ch *says* A *says* r,
B *says* A *serves* B
C *says* B *serves* C |- A for (B for C) *says* r

Who is the initiator of P, which is (A for (B for C)?
It is C.

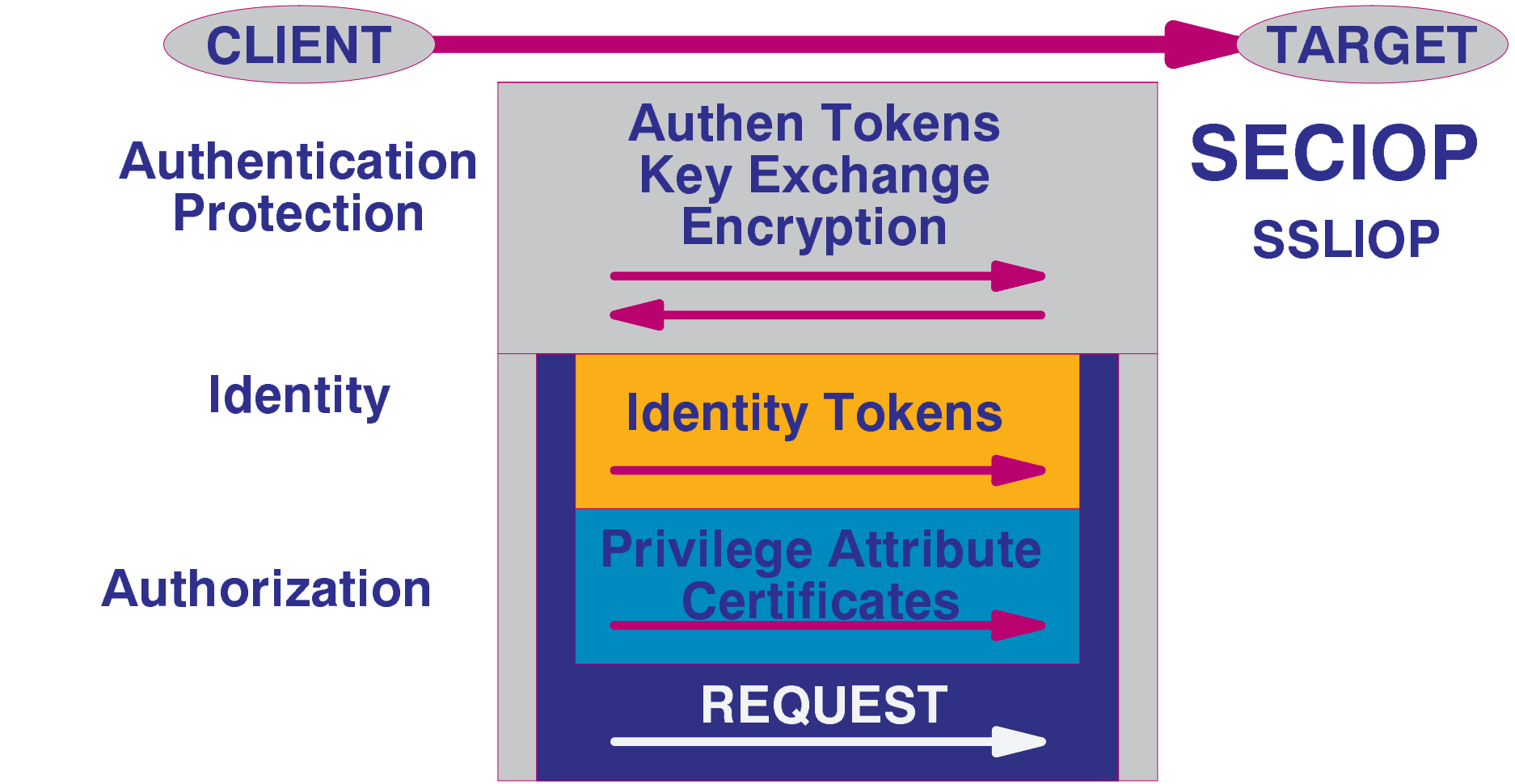
P.name == P.speaks_for.speaks_for.name

Extending The Calculus For Privileges and Restrictions

- Need Statements to Represent Privileges
  - Privilege: S *says* P *has* (p1,...,pn)
  - Restricted Endorsement:

    S *says* ((B *serves* A *on* (r1,...,rn))
    /\ (B *needs* (p1,...,pn))

Common Secure Interoperbility V2



Tokens Delivered on the Wire

- Authentication Tokens used for key exchange and signing.
  - SpeaksFor: (K says Ch => A)
- Identity Tokens
  - Identity: B says r, r is the Request
- Privilege Attribute Certificates
  - Privilege: T says B has p1,...,pn
  - Endorsement: T says A serves B on r1,...,rn

Security Level 3 Credentials Conclusion

- Security Service
  - Does the Proof Work
    - Authentication
    - Privilege Verification
  - Presents the Results to the Application
    - Credentials
      - Statements
      - Principals
- Application
  - From the Credentials interface deliver the Principal Data structure and Statement list to Access Control components for analysis.