Banking Service Example
Preliminaries

- Principals: \{Alice, Bob\}

- Web services:

\[
\begin{align*}
w & = \text{http://bob.com/BankingService} \\
owner(w) & = Bob \\
class(w) & = \text{BankingServiceClass} \\
proxy(w) & = \text{BankingServiceProxy}
\end{align*}
\]
Main call by *Alice*:

```java
new BankingServiceProxy().Balance(12345)
```
Formal Semantics
Transitions

- Goal is to evaluate:

  $Alice[new \text{BankingServiceProxy}().\text{Balance}(12345)]$

- Steps:

  $new \text{BankingServiceProxy}().\text{Balance}(12345)$

  $\rightarrow Alice \ w:\text{Balance}(12345)$

  $\rightarrow Alice \ Bob[new \text{BankingServiceClass}(Alice).\text{Balance}(12345)]$
Transitions (ctd)

• New goal is to evaluate:

\[ \text{Bob}[\text{new BankingServiceClass}(\text{Alice}).\text{Balance}(12345)] \]

• Steps:

\[ \text{new BankingServiceClass}(\text{Alice}).\text{Balance}(12345) \]
\[ \rightarrow_{\text{Bob}} \text{if } 12345 = 12345 \text{ then} \]
\[ \quad \text{if } \text{new BankingServiceClass}(\text{Alice}).\text{CallerId} = \text{Alice} \]
\[ \quad \text{then } 100 \]
\[ \quad \text{else } \text{null} \]
\[ \text{else } \text{null} \]
\[ \rightarrow_{\text{Bob}}^{*} 100 \]
Translation to the spi-calculus
Global Variables

• For each pair of principals, we have a key:

\[ K_{AB} \text{ from } Alice \text{ to } Bob \]
\[ K_{BA} \text{ from } Bob \text{ to } Alice \]

• For each web service \( w \), a public channel \( w \).

• For each class and method we have a public channel:

\[ BSC_B \text{ method } \text{Balance in BankingServiceClass} \]
\[ BSP_I \text{ method } \text{Id in BankingServiceProxy} \]
\[ BSP_B \text{ method } \text{Balance in BankingServiceProxy} \]
Translation of main method

\[
\left[ \text{new } \text{BankingServiceProxy().Balance}(12345) \right]_{topk}^{Alice}
\]

= case \[
\text{null}(y)\]; stop

is \text{BankingServiceProxy}(y);
out \text{BSP}_B(Alice, \[
\text{null}(y),
12345,
\text{null}(y)
\]
, topk)

= out \text{BSP}_B(Alice, \text{BankingServiceProxy}(), 12345, topk)
Translation of method Id in BankingServiceProxy

repeat inp BSP_I (z);
split z is (p, this, k);
out k Bob
Translation of method Balance in BankingServiceProxy

repeat inp BSP_B (z);
split z is (p, this, account, k);
new (k1,k2,t,np);
out w req(getnonce(),k1);
inp k1 res(getnonce(nq));
out w (p, [req(w,Balance(account),t,nq)]K_pB,np,k2);
inp k2 (q’,bdy);
decrypt bdy is [res(plain)]K_pB;
match plain is (w,rest);
split rest is (r,rest’);
macth rest’ is (t,np’);
check np is np’;
case r is Balance(x); out k x
Translation of method Balance in BankingServiceClass

repeat inp BSC_B (z);
split z is (p, this, account, k);
if account=12345 then
    new k’;
    case this
    is null(y); stop
    is BankingServiceClass(y);
    split y is (CallerId);
    out k’ CallerId)
| inp k’ (x);
    if x=Alice
    then out k 100
    else out k null()
else out k null()
repeat inp w (bdy,k1);
case bdy is req(getnonce());
new (nq);
out k1 (res(getnonce(nq)));
inp w (p’,cipher,np,k2);
if p’=Alice then
decrypt cipher is [req(plain)]K_AB;
match plain is (w,rest);
split rest is (a,t,nq’);
check nq is nq’;
new (k);
  case a is Balance(account);
    new (k’);
    out BSC_B (Bob,BankingServiceClass(Alice),
                account,k’)
    | inp k’ (r); out k Balance(r)
    | inp k (r);
    out k2 (Bob,[res(w,r,t,np)]K_AB);
Simulation in the spi-calculus
Main method

out BSP_B( Alice,
            BankingServiceProxy(),
            12345,
            topk)

repeat inp BSP_B (z);
split z is (p, this, account, k);
new (k1,k2,t,np);
out w req(getnonce(),k1);
inp k1 res(getnonce(nq));
out w (p, [req(w,Balance(account),t,nq)]K_pB,np,k2);
inp k2 (q’,bdy);
decrypt bdy is [res(plain)]K_pB;
match plain is (w,rest);
split rest is (r,rest’);
match rest’ is (t,np’);
check np is np’;
case r is Balance(x); out k x
Evaluating in Proxy

\[
\text{out \: w \: req(getnonce(), k1);} \\
\text{inp \: k1 \: res(getnonce(nq));} \\
\text{out \: w \: (Alice, [req(w, Balance(12345), t, nq)]}_{K_{AB}}, np, k2);} \\
\text{inp \: k2 \: (q', bdy);} \\
\text{decrypt \: bdy \: is [res(plain)]}_{K_{AB}}; \\
\text{match \: plain \: is (w, rest);} \\
\text{split \: rest \: is (r, rest');} \\
\text{match \: rest' \: is (t, np');} \\
\text{check \: np \: is np';} \\
\text{case \: r \: is Balance(x); \: out \: topk \: x}
\]
Proxy / Service Interaction

repeat inp w (bdy,k1);
case bdy is req(getnonce());
new (nq);
out k1 (res(getnonce(nq)));
inp w (p',cipher,np,k2);
if p'=Alice then
  decrypt cipher is [req(plain)]K_AB;
match plain is (w,rest);
split rest is (a,t,nq');
check nq is nq';
new (k);
  case a is Balance(account);
    new (k');
    out BSC_B (Bob,BankingServiceClass(Alice),
    account,k')
  | inp k' (r); out k Balance(r)
  | inp k (r);
out k2 (Bob,[res(w,r,t,np)]K_AB);

out w (req(getnonce()),k1);
inp k1 res(getnonce(nq));
out w (Alice,
  [req(w,Balance(12345),t,nq)]K_AB,
  np,
  k2);
inp k2 (q',bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x
Proxy / Service Interaction

out k1 (res(getnonce(nq)));  
inp w (p',cipher,np,k2);  
if p'=Alice then  
decrypt cipher is [req(plain)]K_AB;  
match plain is (w,rest);  
split rest is (a,t,np');  
check np is np';  
new (k);  
case a is Balance(account);  
  new (k');  
    out BSC_B (Bob,BankingServiceClass(Alice),  
    account,k')  
    | inp k' (r); out k Balance(r)  
    | inp k (r);  
    out k2 (Bob,[res(w,r,t,np)]K_AB);  

inp k2 (q’,bdy);  
decrypt bdy is [res(plain)]K_AB;  
match plain is (w,rest);  
split rest is (r,rest’);  
match rest’ is (t,np’);  
check np is np’;  
case r is Balance(x); out topk x
Proxy / Service Interaction

```plaintext
inp w (p',cipher,np,k2);
if p' = Alice then
decrypt cipher is [req(plain)]K_AB;
match plain is (w,rest);
split rest is (a,t,nq');
check nq is nq';
new (k);
  case a is Balance(account);
    new (k');
    out BSC_B (Bob,BankingServiceClass(Alice),
              account,k')
  | inp k' (r); out k Balance(r)
  | inp k (r);
  out k2 (Bob,[res(w,r,t,np)]K_AB);
```

out w (Alice,
       [req(w,Balance(12345),t,np)]K_AB,
       np,
       k2);
in p k2 (q',bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x
```
Proxy / Service Interaction

decrypt
[req(w,Balance(12345),t,nq)]K_AB
match plain is (w,rest);
split rest is (a,t,nq');
check nq is nq';
new (k);
case a is Balance(account);
    new (k');
    out BSC_B (Bob,BankingServiceClass(Alice),
              account,k')
| inp k' (r); out k Balance(r)
| inp k (r);
    out k2 (Bob,[res(w,r,t,np])K_AB;

inp k2 (q',bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x
Proxy / Service Interaction

match (w,Balance(12345),t,nq)
is (w,rest);
split rest is (a,t,nq');
check nq is nq';
new (k);

case a is Balance(account);
    new (k');
    out BSC_B (Bob,BankingServiceClass(Alice),
               account,k')
    | inp k' (r); out k Balance(r)
    | inp k (r);
    out k2 (Bob,[res(w,r,t,np])K_AB;

inp k2 (q’,bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest’);
match rest’ is (t,np’);
check np is np’;
case r is Balance(x); out topk x
Proxy / Service Interaction

```plaintext
split (Balance(12345), t, nq)
  is (a, t, nq');
check nq is nq';
new (k);
  case a is Balance(account);  
      new (k');
      out BSC_B (Bob, BankingServiceClass(Alice), account, k')
      | inp k' (r); out k Balance(r)
      | inp k (r);
      out k2 (Bob, [res(w, r, t, np)] K_AB);
```
Proxy / Service Interaction

```
case Balance(12345) is Balance(account);
  new (k');
  out BSC_B (Bob, BankingServiceClass(Alice),
           account, k')
    | inp k' (r); out k Balance(r)
    | inp k (r);
    out k2 (Bob, [res(w,r,t,np)]K_AB);
```

```plaintext
inp k2 (q', bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w, rest);
split rest is (r, rest');
match rest' is (t, np');
check np is np';
case r is Balance(x); out topk x
```
Proxy / Service Interaction

repeat inp BSC_B (z);
split z is (p, this, account, k);
if account=12345 then
    new k';
    case this
        is null(y); stop
        is BankingServiceClass(y);
        split y is (CallerId);
        out k' CallerId)
    | inp k' (x);
    if x=Alice
        then out k 100
        else out k null()
else out k null()

inp k2 (q’,bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x

out BSC_B (Bob,BankingServiceClass(Alice),12345,k')
| inp k' (r); out k Balance(r)
| inp k (r); out k2 (Bob,[res(w,r,t,np)]K_AB);
Proxy / Service Interaction

if 12345=12345 then
new k’’;
    case BankingServiceClass(Alice)
        is null(y); stop
        is BankingServiceClass(y);
            split y is (CallerId);
            out k’’ CallerId)
            | inp k’’ (x);
            if x=Alice
                then out k’ 100
                else out k’ null()
        else out k’ null()
    | inp k’ (x);
    if x=Alice
        then out k’ 100
        else out k’ null()
else out k’ null()

inp k’ (r); out k Balance(r)
| inp k (r); out k2 (Bob,[res(w,r,t,np)]K_AB);
Proxy / Service Interaction

case BankingServiceClass(Alice)
  is null(y); stop
  is BankingServiceClass(y);
    split y is (CallerId);
    out k'' CallerId)
| inp k'' (x);
  if x=Alice
    then out k' 100
    else out k' null()

inp k' (r); out k Balance(r)
| inp k (r); out k2 (Bob,[res(w,r,t,np)]K_AB;
split Alice is (CallerId);
  out k'' CallerId)
| inp k'' (x);
  if x=Alice
    then out k' 100
  else out k' null()

inp k2 (q',bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x

inp k' (r); out k Balance(r)
| inp k (r); out k2 (Bob,[res(w,r,t,np)]K_AB);
Proxy / Service Interaction

\begin{verbatim}
out k'' Alice)  inp k' (x);
   if x=Alice
      then out k' 100
      else out k' null()

inp k' (r); out k Balance(r)
inp k (r); out k2 (Bob,[res(w,r,t,np])K_AB;
\end{verbatim}
Proxy / Service Interaction

\[
\text{out } k' \ 100
\]

\[
\text{inp } k2 (q',bdy);
\text{decrypt bdy is } \text{[res(plain)]} K_{AB};
\text{match plain is } (w,rest);
\text{split rest is } (r,rest');
\text{match rest' is } (t,np');
\text{check np is } np';
\text{case } r \text{ is Balance}(x); \text{ out } topk x
\]

\[
\text{inp } k' (r); \text{ out } k \text{ Balance}(r)
\]

| \text{inp } k (r); \text{ out } k2 (Bob, [res(w,r,t,np)] K_{AB}); 

Proxy / Service Interaction

\[
\text{inp k2 (q',bdy);} \\
\text{decrypt bdy is } [\text{res(plain)}]K_AB; \\
\text{match plain is } (w,\text{rest}); \\
\text{split rest is } (r,\text{rest'}); \\
\text{match rest' is } (t,\text{np'}); \\
\text{check np is np';} \\
\text{case r is Balance(x); out topk x}
\]
Proxy / Service Interaction

inp k2 (q’,bdy);
decrypt bdy is [res(plain)]K_AB;
match plain is (w,rest);
split rest is (r,rest’);
match rest’ is (t,np’);
check np is np’;
case r is Balance(x); out topk x

out k2 (Bob,[res(w,Balance(100),t,np)]K_AB);
Proxy / Service Interaction

decrypt \([\text{res}(w,\text{Balance}(100),t,np)]\)_{K_{AB}} \text{ is } \text{[res(plain)]}_{K_{AB}};

\text{match } \text{plain is (w,rest)};
\text{split } \text{rest is (r,rest')};
\text{match } \text{rest' is (t,np')};
\text{check } \text{np is np'};
\text{case } r \text{ is Balance(x); out topk x}
Proxy / Service Interaction

match (w,Balance(100),t,np) is (w,rest);
split rest is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x

split (Balance(100),t,np) is (r,rest');
match rest' is (t,np');
check np is np';
case r is Balance(x); out topk x

check np is np;
case Balance(100) is Balance(x); out topk x

out topk 100
Conclusion

• The execution of the example in the spi-calculus corresponds to the specification. The proof shows that every possible execution, even in the presence of attackers, still corresponds to the specification.

• Extend the above to deal with authentication. (Last appendix in the paper.)