Banking Service Example

Preliminaries

- Principals: $\{Alice, Bob\}$
- Web services:

$$\begin{aligned} w &= \texttt{http://bob.com/BankingService} \\ owner(w) &= Bob \\ class(w) &= \texttt{BankingServiceClass} \\ proxy(w) &= \texttt{BankingServiceProxy} \end{aligned}$$

Source Program

```
class BankingServiceClass
  Id CallerId
  Num Balance (Num account)
    if account=12345 then
      if this.CallerId = Alice
      then 100
      else null
    else null
```

```
class BankingServiceProxy
  Id Id () Bob
  Num Balance (Num account)
    w:Balance(account)
```

Main call by *Alice*:

```
    new BankingServiceProxy().Balance(12345)
```

Formal Semantics

Transitions

- Goal is to evaluate:

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

- Steps:

$$\begin{array}{ll} & \texttt{new BankingServiceProxy().Balance(12345)} \\ \longrightarrow_{Alice} & \texttt{w:Balance(12345)} \\ \longrightarrow_{Alice} & Bob[\texttt{new BankingServiceClass(}\textit{Alice}\texttt{).Balance(12345)}] \end{array}$$

Transitions (ctd)

- New goal is to evaluate:

$$Bob[\texttt{new BankingServiceClass}(\mathit{Alice})\texttt{.Balance(12345)}]$$

- Steps:

```
          new BankingServiceClass(Alice).Balance(12345)]
→_Bob     if 12345=12345 then
            if new BankingServiceClass(Alice).CallerId
               = Alice
            then 100
            else null
          else null
```

$$\longrightarrow^{*}_{Bob}\ 100$$

Translation to the spi-calculus

Global Variables

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

K_{AB}	from *Alice* to *Bob*
K_{BA}	from *Bob* to *Alice*

- For each web service w, a public channel w.

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

BSC_B	method Balance in BankingServiceClass
BSP_I	method Id in BankingServiceProxy
BSP_B	method Balance in BankingServiceProxy

Translation of main method

$$[\![\texttt{new BankingServiceProxy().Balance(12345)}]\!]^{Alice}_{topk}$$

```
= case ⟦new BankingServiceProxy()⟧
    is null(y);stop
    is BankingServiceProxy(y);
       out BSP_B( Alice,
                  ⟦new BankingServiceProxy()⟧,
                  12345,
                  topk)

= out BSP_B( Alice,
             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');
match 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()
```

Translation of the Web Service

```
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

```
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

```
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,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 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 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
```

```
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;
```

Proxy / Service Interaction

```
decrypt
  [req(w,Balance(12345),t,nq)]K_AB
  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;
```

```
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

```
                                                   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
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

```
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
```

```
  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;
```

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 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

```
  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 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

```
  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

```
  out k'' Alice)
| 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

```
out k' 100
```

```
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

```
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 k Balance(100)
| inp k (r);  out k2 (Bob,[res(w,r,t,np)]K_AB;
```

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 [res(w,Balance(100),t,np)]K_AB 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(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.)