A Additional functions

Figure 3 showed the foundational monadic functions parallel, congruently, comm, enclave, naked, fanOut, and fanIn. Figure 9 shows type signatures for the entire library (as exported), in alphabetical order. Some functions are "core" in the sense that they can't be written without reference to MultiChor's internal machinery. Other functions are "helper functions", which are implemented entirely using the exposed core functions.

B Additional Choreographies for the GMW Protocol

Oblivious transfer. Our implementation of oblivious transfer leverages a common strategy for building OT from public-key encryption. First, the receiver generates two public keys and one secret key (line 23); one of the public keys is real, and corresponds to the secret key, while the other is chosen at random from the space of public keys and has no corresponding secret key. The select bit determines the ordering of the public keys. The receiver sends the public keys to the sender (lines 24–25); the sender encrypts both b_1 and b_2 with the corresponding public keys and sends the ciphertexts back to the receiver (lines 26–27). Finally, the receiver decrypts the selected value (lines 28–30).

The sender treats both b_1 and b_2 in the same way, and cannot tell which public key is real and which one is fake—so the sender does not learn which value was selected. The receiver gets both encrypted values, but can only decrypt one of them, since only one of the public keys used has a corresponding secret key (the other public key is totally random). This version of OT is secure only against *honest but curious* or *passive* adversaries, who observe network communications but do not change the behavior of the parties, since an actively malicious receiver could create two actual key pairs in the first step of the protocol and thus be able to decrypt both b_1 and b_2 at the end. More complicated variants of the protocol can defend against this kind of attack.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009


```
1541
           (-~>) :: forall a l ls' m ps. (Show a, Read a, KnownSymbol l, KnownSymbols ls')
1542
                 => (Member 1 ps, m a) -> Subset 1s' ps -> Choreo ps m (Located 1s' a)
1543
           infix 4 -~>
1544
           (~~>) :: forall a l ls' m ps. (Show a, Read a, KnownSymbol l, KnownSymbols ls')
                => (Member 1 ps, Unwrap 1 -> m a) -> Subset 1s' ps -> Choreo ps m (Located 1s' a)
1546
           infix 4 ~~>
1547
1548
           (@@) :: Member x ys -> Subset xs ys -> Subset (x ': xs) ys
1549
       10
1550
       11
1551
       12
           (~>) :: (Show a, Read a, KnownSymbol 1, KnownSymbols ls', CanSend s l a ls ps w)
1552
                ⇒ s -- ^ The message argument can take three forms:
       13
                              `(Member sender census, wrapped owners a)` where
1553
       14
1554
       15
                                  the sender is explicitly listed in owners,
                              '(Member sender owners, Subset owners census, wrapped owners a)', or
1555
                              '(Member sender census, (Member sender owners, wrapped owners a)'.
       17
                -> Subset ls' ps -> Choreo ps m (Located ls' a)
1557
           infix 4 ~>
      19
1559
           allOf :: forall ps. Subset ps ps
      21
1561
           Backend :: Type -> Constraint -- A phonebook for running Network monad expressions.
      23
1562
      24
1563
           broadcast :: forall l a ps ls w m s.
      25
                         (Show a, Read a, KnownSymbol 1, KnownSymbols ps, CanSend s l a ls ps w)
1564
      26
1565
      27
                     => s -> Choreo ps m a
1566
      28
           Choreo :: [LocTy] -> (Type -> Type) -> Type -> Type
1567
           type Choreo ps m = Freer (ChoreoSig ps m)
      30
1568
      31
1569
           comm :: (Show a, Read a, KnownSymbol 1, KnownSymbols 1s', Wrapped w)
      32
1570
                => Member 1 ps -> (Member 1 ls, w ls a) -> Subset ls' ps
      33
1571
                -> Choreo ps m (Located ls' a)
1572
           infix 4 `comm`
       35
1573
1574
       37
           cond :: (KnownSymbols ls)
                => (Subset ls ps, (Subset ls qs, Located qs a)) -> (a -> Choreo ls m b)
1575
       38
                -> Choreo ps m (Located 1s b)
1576
       39
1577
           congruently :: (KnownSymbols ls)
       41
1578
                         => Subset 1s ps -> (Unwraps 1s -> a) -> Choreo ps m (Located 1s a)
           infix 4 `congruently`
       43
1580
       44
1581
           consSet :: Subset xs (x ': xs)
      45
1582
1583
           consSub :: Subset xs ys -> Member x ys -> Subset (x ': xs) ys
       47
1584
1585
           consSuper :: forall xs ys y. Subset xs ys -> Subset xs (y ': ys)
1586
1587
                                              Figure 9. The MultiChor API, part 1/4.
1588
```

```
1651
           enclave :: (KnownSymbols ls)
1652
      51
                   ⇒ Subset ls ps → Choreo ls m a → Choreo ps m (Located ls a)
1653
      52
           infix 4 `enclave`
      53
1654
      54
           enclaveTo :: forall ls a rs ps m. (KnownSymbols ls)
                     => Subset ls ps -> Subset rs ls -> Choreo ls m (Located rs a)
      55
1656
                     -> Choreo ps m (Located rs a)
1657
      57
           infix 4 `enclaveTo`
1658
1659
           enclaveToAll :: forall ls a ps m. (KnownSymbols ls)
1660
       60
                         => Subset ls ps -> Choreo ls m (Located ls a) -> Choreo ps m (Located ls a)
1661
      61
           infix 4 `enclaveToAll`
1662
      62
           epp :: (Monad m) => Choreo ps m a -> LocTm -> Network m a
1663
      63
1664
           ExplicitMember :: forall k. k -> [k] -> Constraint
1665
           explicitMember :: forall k (x :: k) (xs :: [k]). ExplicitMember x xs => Member x xs
      66
      67
1667
           ExplicitSubset :: forall {k}. [k] -> [k] -> Constraint
      68
1668
           explicitSubset :: forall \{k\} (xs :: [k]) (ys :: [k]). ExplicitSubset xs ys \Rightarrow Subset xs ys
      69
1669
1670
      71
           Faceted :: [LocTy] -> Type -> Type
1671
1672
      73
           fanIn :: (KnownSymbols qs, KnownSymbols rs)
1673
      74
                  => Subset as ps -> Subset rs ps
1674
      75
                  -> (forall q. (KnownSymbol q)
1675
       76
                      ⇒ Member q qs → Choreo ps m (Located rs a))
                  -> Choreo ps m (Located rs [a])
1676
      77
1677
           fanOut :: (KnownSymbols qs, Wrapped w)
1678
                  => Subset as ps
      80
1679
                  -> (forall q. (KnownSymbol q) => Member q qs -> Choreo ps m (w '[q] a))
      81
1680
                  -> Choreo ps m (Faceted qs a)
1682
           flatten :: Subset 1s ms -> Subset 1s ns -> Located ms (Located ns a)
1683
                   -> Located ls a
       85
           infix 3 `flatten`
1684
       86
      87
           fracture :: forall ls a. (KnownSymbols ls) => Located ls a -> Faceted ls a
      88
1686
1687
      89
           inSuper :: Subset xs ys -> Member x xs -> Member x ys
      90
1688
      91
           IsMember :: forall k. k -> [k] -> Type
1690
      93
1691
           IsSubset :: forall k. [k] -> [k] -> Type
      94
1692
      95
1693
           KnownSymbols :: [Symbol] -> Constraint -- lift KnownSymbol to type-level lists
      96
1694
1695
           listedFifth :: forall p5 p4 p3 p2 p1 ps. Member p5 (p1 ': p2 ': p3 ': p4 ': p5 ': ps)
1696
           listedFirst :: forall p1 ps. Member p1 (p1 ': ps)
1697
      100
      101
1698
           listedForth :: forall p4 p3 p2 p1 ps. Member p4 (p1 ': p2 ': p3 ': p4 ': ps)
      102
1699
1700
           listedSecond :: forall p2 p1 ps. Member p2 (p1 ': p2 ': ps)
1701
1702
                                               Figure 9. The MultiChor API, part 2/4.
1703
```

```
listedSixth :: forall p6 p5 p4 p3 p2 p1 ps. Member p6 (p1 ': p2 ': p3 ': p4 ': p5 ': p6 ': ps)
1761
1762
1763
      107
           listedThird :: forall p3 p2 p1 ps. Member p3 (p1 ': p2 ': p3 ': ps)
      108
1764
           localize :: (KnownSymbol 1) => Member 1 ls -> Faceted ls a -> Located '[1] a
      109
      110
1766
           locally :: (KnownSymbol (1 :: LocTy))
      111
1767
                    => Member 1 ps -> (Unwrap 1 -> m a) -> Choreo ps m (Located '[1] a)
       12
1768
           infix 4 `locally`
       13
1769
       14
1770
       115
           locally_ :: (KnownSymbol 1) => Member 1 ps -> (Unwrap 1 -> m ()) -> Choreo ps m ()
1771
      116
           infix 4 `locallv `
1772
      117
           _locally :: (KnownSymbol 1) => Member 1 ps -> m a -> Choreo ps m (Located '[1] a)
1773
      118
           infix 4 `_locally`
1774
      119
      120
1775
           _locally_ :: (KnownSymbol 1) => Member 1 ps -> m () -> Choreo ps m ()
      121
1776
           infix 4 `_locally_`
      122
1777
      123
           Located :: [LocTy] -> Type -> Type
      124
1779
       125
1780
           LocTm :: Type
1781
           type LocTm = String
      127
1782
       128
1783
           LocTy :: Type
      129
1784
      130
           type LocTy = Symbol
1785
      131
           Member :: forall {k}. k -> [k] -> Type
1786
      132
           type Member x xs = Proof (IsMember x xs)
1787
      134
1788
           mkLoc :: String -> 0 [Dec] -- Template Haskell
      135
1789
      136
1790
           naked :: Subset ps qs -> Located qs a -> Choreo ps m a
      137
1791
           infix 4 `naked`
       138
1792
1793
           Network :: (Type -> Type) -> Type -> Type
       40
1794
       41
           NetworkSig :: (Type -> Type) -> Type -> Type
       142
1796
      143
           nobody :: Subset '[] ys
1797
      144
      145
1798
           parallel :: (KnownSymbols ls)
1799
                     => Subset ls ps -> (forall 1. (KnownSymbol 1) => Member 1 ls -> Unwrap 1 -> m a)
       147
1800
                     -> Choreo ps m (Faceted 1s a)
       148
1801
1802
           parallel_ :: forall ls ps m. (KnownSymbols ls)
       150
1803
                      => Subset 1s ps -> (forall 1. (KnownSymbol 1) => Member 1 1s -> Unwrap 1 ->m ())
      151
1804
                      -> Choreo ps m ()
       152
1805
       153
1806
       154
           _parallel :: forall ls a ps m. (KnownSymbols ls)
                      ⇒ Subset ls ps → m a → Choreo ps m (Faceted ls a)
      155
1807
1808
           recv :: forall a (m :: Type -> Type). Read a => LocTm -> Network m a
      157
1809
1810
           run :: forall (m :: Type -> Type) a. m a -> Network m a
1811
1812
                                                Figure 9. The MultiChor API, part 3/4.
1813
```

```
1871
           runChoreo :: forall ps b m. Monad m => Choreo ps m b -> m b
1872
      161
           runNetwork :: (Backend c, MonadIO m) => c -> LocTm -> Network m a -> m a
1873
      162
      163
1874
           send :: forall a (m :: Type -> Type). Show a => a -> [LocTm] -> Network m ()
      164
1875
      165
1876
           singleton :: forall p. Member p (p ': '[])
      166
1877
      167
1878
           Subset :: forall {k}. [k] -> [k] -> Type
      168
1879
      169
           type Subset xs ys = Proof (IsSubset xs ys)
1880
      170
1881
           toLocs :: forall (ls :: [LocTy]) (ps :: [LocTy]). KnownSymbols ls => Subset ls ps -> [LocTm]
      171
1882
      172
           toLocTm :: forall (1 :: LocTy) (ps :: [LocTy]). KnownSymbol 1 => Member 1 ps -> LocTm
1883
      173
1884
      174
           Unwrap :: LocTy -> Type
      175
1885
           type Unwrap (1 :: LocTy) = forall ls a w. (Wrapped w) => Member l ls -> w ls a -> a
      176
1886
      177
1887
           Unwraps :: [LocTy] -> Type
      178
1888
           type Unwraps (qs :: [LocTy]) = forall ls a. Subset qs ls -> Located ls a -> a
       79
1889
1890
           Wrapped :: ([Symbol] -> Type -> Type) -> Constraint
1891
1892
                                               Figure 9. The MultiChor API, part 4/4.
1893
```

```
1981
           genKeys :: (CRT.MonadRandom m) => Bool -> m (RSA.PublicKey, RSA.PublicKey, RSA.PrivateKey)
1982
           genKeys s = do -- Generate keys for OT. One key is real, and one is fake - select bit decides
1983
             (pk, sk) <- genKeyPair
            fakePk <- generateFakePK
1984
            return $ if s then (pk, fakePk, sk) else (fakePk, pk, sk)
1986
           encryptS :: (CRT.MonadRandom m) => -- Encryption based on select bit
1987
                        (RSA.PublicKey, RSA.PublicKey) -> Bool -> Bool -> m (ByteString, ByteString)
1988
           encryptS (pk1, pk2) b1 b2 = do c1 <- encryptRSA pk1 b1; c2 <- encryptRSA pk2 b2; return (c1, c2)
1989
       10
1990
       11
           decryptS :: (CRT.MonadRandom m) => -- Decryption based on select bit
1991
                  (RSA.PublicKey, RSA.PublicKey, RSA.PrivateKey) -> Bool -> (ByteString, ByteString) -> m Bool
       12
1992
           decryptS (_, _, sk) s (c1, c2) = if s then decryptRSA sk c1 else decryptRSA sk c2
       13
1993
       14
           -- One out of two OT
1994
       15
           ot2 :: (KnownSymbol sender, KnownSymbol receiver, MonadIO m, CRT.MonadRandom m) =>
1995
            Located '[sender] (Bool, Bool) -> Located '[receiver] Bool
      17
             -> Choreo '[sender, receiver] (CLI m) (Located '[receiver] Bool)
       18
1997
           ot2 bb s = do
       19
            let sender = listedFirst :: Member sender '[sender, receiver]
       20
1999
            let receiver = listedSecond :: Member receiver '[sender, receiver]
      21
       22
2001
            keys <- receiver `locally` \un -> liftIO $ genKeys $ un singleton s
       23
2002
       24
            pks <- (receiver, \un -> let (pk1, pk2, _) = un singleton keys
2003
       25
                                       in return (pk1, pk2)) ~~> sender @@ nobody
            encrypted <- (sender, \un \rightarrow let (b1, b2) = un singleton bb
       26
                                           in liftIO \ encryptS (un singleton pks) b1 b2) ^{\sim\sim}> receiver @@ nobody
2005
       27
            receiver `locally` \un -> liftIO $ decryptS (un singleton keys)
2006
       28
       29
                                                                         (un singleton s)
2007
                                                                         (un singleton encrypted)
2008
2009
```

Figure 10. A choreography for performing 1 out of 2 oblivious transfer (OT) using RSA public-key encryption. The choreography involves exactly two parties, sender and receiver.