Functionalities in ILC

Cool kids

1 Authenticated Message Transmission

Functionality $\mathcal{F}_{\text{AUTH}}$

1. Upon receiving an input $(\texttt{Send}, S, R, sid, m)$ from ITI S, generate a public delayed output $(\texttt{Sent}, S, sid, m)$ to R and halt.
2. Upon receiving $(\texttt{Corrupt-sender}, sid, m')$ from the adversary, and if the $(\texttt{Sent}, S, sid, m)$ output is not yet delivered to R, then output $(\texttt{Sent}, S, sid, m')$ to R and halt.

```
(* Do not need corrupt sender in static corruptions world *)

nu { p2f, f2p, a2f } .
let ("Send", S, R, sid, m) = rd p2f in
    (wr[pub, delay] (("Sent", S, sid, m), R) -> f2p) &
    (let ("Corrupt-sender", sid, m') = rd a2f in
        wr (("Sent", S, sid, m'), R) -> f2p)
```

2 Secure Message Transmission

Functionality $\mathcal{F}^{l}_{\text{SMT}}$

1. Upon receiving an input $(\texttt{Send}, S, R, sid, m)$ from ITI S, send $(\texttt{Sent}, S, R, sid, l(m))$ to the adversary, generate a private delayed output $(\texttt{Sent}, S, sid, m)$ to R and halt.
2. Upon receiving $(\texttt{Corrupt}, sid, P)$ from the adversary, where $P \in \{S, R\}$, disclose m to the adversary. Next, if the adversary provides a value m', and $P = S$, and no output has been yet written to R, then output $(\texttt{Sent}, S, sid, m')$ to R and halt.

```
(* Need way to parameterize functionality? *)
(* Better way to check if delayed output to R *)

nu { p2f, f2p, f2a, a2f } .
let ("Send", S, R, sid, m) = rd p2f in
    wr ("Sent", S, R, sid, l(m)) -> f2a .
    (wr[priv, delay] (("Sent", S, sid, m), R) -> f2p) &
    (let ("Corrupt", sid, P) = rd a2f in
        wr m -> f2a . let m' = rd a2f in
            if P == S then wr (("Sent", S, sid, m'), R) -> f2p)
```

3 Zero Knowledge Proofs

Functionality $\mathcal{F}_{\mathrm{ZK(R)}}$

1. Upon receiving input $(\texttt{Prove}, x, w, B)$ from A, leak $(A, B, x, R(x, w))$ to S.
2. When S returns *ok*, output $(\texttt{Verified}, A, x, R(x, w))$ to B.

```
(* Cleaner way to supply multiple args to function *)

nu { p2f, f2p } .
let ("Prove", x, w, B) = rd p2f in
    wr ((A, B, x, (R x) w), S) -> f2p .
    if rd p2f == "ok" then wr (("Verified", A, x, (R x) w), B) -> f2p
```

4 Key Exchange

Functionality $\mathcal{F}_{\mathrm{KE}}$

1. Upon receiving input (KE, B) from A, choose a key k and leak (A, B) to S.
2. Upon receiving input (KE, B) from A, leak (A, B) to S.
3. When S returns (ok, P) for $P = \{A, B\}$, output (A, B, k) to P.

```
(* No idea what S or KE are. Will read about it later *)

nu { p2f, f2p } .
let (KE, B) = rd p2f in
    let k = rand in
        wr ((A, B), S) -> f2p .
        rd (KE, B) = rd p2f .
        wr ((A, B), S) -> f2p .
        let ("ok", P) = rd p2f in
            wr ((A, B, k), P) -> f2p
```

5 Common Reference String

From UC commitments [1].

Functionality $\mathcal{F}_{\mathrm{CRS}}$

$\mathcal{F}_{\mathrm{CRS}}$ proceeds as follows, when parameterized by a distribution D.

1. When activated for the first time on input $(\texttt{value}, sid)$, choose a value $d \xleftarrow{R} D$ and send d back to the activating party. In each other activation return the value d to the activating party.

```
(* Functionality parameterized by distribution D *)

```

```
nu {p2f , f2p} .
let (pid , sid) = rd p2f in
  let d = rand in
    wr (pid , d) -> f2p .
    !(let (pid , sid) = rd p2f in
      wr (pid , d) -> f2p);;

(* Test functionality *)
nu {p2f , f2p} .
wr (0 , 0) -> p2f . let (pid , d) = rd f2p in
    show pid ++ " received " ++ show d
| wr (1 , 1) -> p2f . let (pid , d) = rd f2p in
    show pid ++ " received " ++ show d
| wr (2 , 2) -> p2f . let (pid , d) = rd f2p in
    show pid ++ " received " ++ show d
| wr (3 , 3) -> p2f . let (pid , d) = rd f2p in
    show pid ++ " received " ++ show d;;
```

6 Synchronous Communication

Functionality $\mathcal{F}_{\text{SYN}}$

$\mathcal{F}_{\text{SYN}}$ expects its SID to be of the form $sid = (\mathcal{P}, sid')$ where $\mathcal{P}$ is a list of party identities among which sychronization is to be provided. It proceeds as follows.

1. At the first activation, initialize a round counter $r \leftarrow 1$, and send a public delayed output $(\texttt{Init}, sid)$ to all parties $\mathcal{P}$.
2. Upon receiving input $(\texttt{Send}, sid, M)$ from a party $P \in \mathcal{P}$, where $M = \{(m_i, R_i)\}$ is a set of pairs of messages m_i and recipient identities $R_i \in \mathcal{P}$, record (P, M, r) and output (sid, P, M, r) to the adversary.
3. Upon receiving input $(\texttt{Receive}, sid, r')$ from a party $P \in \mathcal{P}$ do:
   (a) If $r' = r$ (i.e., r' is the current round), and all uncorrupted parties in $\mathcal{P}$ have already sent their messages for round r, then:
      i. Interpret N as the list of messages sent by all parties in this round. That is, $N = \{(S_i, R_i, m_i)\}$ where each $S_i, R_i \in \mathcal{P}$, m is a messages, and $S_i, R_i \in \mathcal{P}$. (S_i is taken as the sender of message m_i and R_i is the receiver. Note that either sender or receiver may be corrupted.)
      ii. Prepare for each party $P \in \mathcal{P}$ the list L_P^r of messages that were sent to it in round r.
      iii. Increment the rounder number: $r \leftarrow r + 1$.
      iv. Output $(\texttt{Received}, sid, L_P^{r-1})$ to P. (Let $L_P^0 = \bot$.)

   (b) If $r' < r$ then output $(\texttt{Received}, sid, L_P^{r'})$ to P.
   (c) Else (i.e., $r' > r$ or not all parties in $\mathcal{P}$ have sent their messages for round r), output $(\texttt{Round Incomplete})$ to P.

References

[1] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology—Crypto 2001*, pages 19–40. Springer, 2001.