

# Functionalities in ILC

Cool kids

## 1 Authenticated Message Transmission

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

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

```
1  (* Do not need corrupt sender in static corruptions world *)
2
3  nu { p2f, f2p, a2f } .
4  let ("Send", S, R, sid, m) = rd p2f in
5    (wr[pub, delay] (("Sent", S, sid, m), R) -> f2p) &
6    (let ("Corrupt-sender", sid, m') = rd a2f in
7      wr (("Sent", S, sid, m'), R) -> f2p)
```

## 2 Secure Message Transmission

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

1. Upon receiving an input  $(\text{Send}, S, R, \text{sid}, m)$  from  $ITI\ S$ , send  $(\text{Sent}, S, R, \text{sid}, l(m))$  to the adversary, generate a private delayed output  $(\text{Sent}, S, \text{sid}, m)$  to  $R$  and halt.
2. Upon receiving  $(\text{Corrupt}, \text{sid}, P)$  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  $(\text{Sent}, S, \text{sid}, m')$  to  $R$  and halt.

```
1  (* Need way to parameterize functionality? *)
2  (* Better way to check if delayed output to R *)
3
4  nu { p2f, f2p, f2a, a2f } .
5  let ("Send", S, R, sid, m) = rd p2f in
6    wr ("Sent", S, R, sid, l(m)) -> f2a .
7    (wr[priv, delay] (("Sent", S, sid, m), R) -> f2p) &
8    (let ("Corrupt", sid, P) = rd a2f in
9      wr m -> f2a . let m' = rd a2f in
10       if P == S then wr (("Sent", S, sid, m'), R) -> f2p)
```

### 3 Zero Knowledge Proofs

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

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

```

1  (* Cleaner way to supply multiple args to function *)
2
3  nu { p2f, f2p } .
4  let ("Prove", x, w, B) = rd p2f in
5      wr ((A, B, x, (R x) w), S) -> f2p .
6      if rd p2f == "ok" then wr (("Verified", A, x, (R x) w), B) -> f2p

```

### 4 Key Exchange

#### Functionality $\mathcal{F}_{\text{KE}}$

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  $P$ .

```

1  (* No idea what S or KE are. Will read about it later *)
2
3  nu { p2f, f2p } .
4  let (KE, B) = rd p2f in
5      let k = rand in
6          wr ((A, B), S) -> f2p .
7          rd (KE, B) = rd p2f .
8          wr ((A, B), S) -> f2p .
9          let ("ok", P) = rd p2f in
10             wr ((A, B, k), P) -> f2p

```

### 5 Common Reference String

From UC commitments [1].

#### Functionality $\mathcal{F}_{\text{CRS}}$

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

1. When activated for the first time on input  $(\text{value}, \text{sid})$ , 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.

```

1  (* Functionality parameterized by distribution D *)
2

```

```

3 nu {p2f, f2p} .
4 let (pid, sid) = rd p2f in
5   let d = rand in
6     wr (pid, d) -> f2p .
7     !(let (pid, sid) = rd p2f in
8       wr (pid, d) -> f2p);;
9
10 (* Test functionality *)
11 nu {p2f, f2p} .
12 wr (0, 0) -> p2f . let (pid, d) = rd f2p in
13   show pid ++ " received " ++ show d
14 | wr (1, 1) -> p2f . let (pid, d) = rd f2p in
15   show pid ++ " received " ++ show d
16 | wr (2, 2) -> p2f . let (pid, d) = rd f2p in
17   show pid ++ " received " ++ show d
18 | wr (3, 3) -> p2f . let (pid, d) = rd f2p in
19   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 synchronization 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 (**Init**,  $sid$ ) to all parties  $\mathcal{P}$ .
2. Upon receiving input (**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 (**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  $S_i, R_i \in \mathcal{P}$ ,  $m_i$  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 (**Received**,  $sid, L_P^{r-1}$ ) to  $P$ . (Let  $L_P^0 = \perp$ .)
  - (b) If  $r' < r$  then output (**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 (**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.