

Protocol Π_{Chan}

- 1: Initialisation:
- 2: $State \leftarrow \text{INIT}$

- 3: On TOP UP, CHECK TOP UP by \mathcal{E} , act as $\mathcal{F}_{\text{Chan}}$ (Fig. 3, lines 4-8 and 9-15 respectively)

- 4: On (OPEN, c_F , $pk_{A,out}$, $pk_{B,out}$) by \mathcal{E} :
- 5: ensure $State = \text{TOPPED UP}$
- 6: $State \leftarrow \text{OPENING BASE CHANNEL}$
- 7: do LN (other box)

- 8: On (CHECK FUNDING) by \mathcal{E} :
- 9: ensure $State = \text{WAITING FOR LEDGER}$
- 10: send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ
- 11: ensure $F \in \Sigma$
- 12: $c_A \leftarrow c$; $c_B \leftarrow 0$ // c received in OPEN
- 13: $State \leftarrow \text{OPEN BASE}$
- 14: output (OPEN SUCCESS) to \mathcal{E}

- 15: On (PAY, x) by \mathcal{E} :
- 16: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 17: ensure $c_A \geq x$
- 18: do LN payment (these channels won't be async) (balance change here)
- 19: output (OK) to \mathcal{E}

- 20: On (BALANCE) by \mathcal{E} :
- 21: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 22: output (BALANCE, $(c_A, c_B, \text{locked}_A, \text{locked}_B)$) to \mathcal{E}

- 23: On (CLOSE) by \mathcal{E} :
- 24: **if** $State = \text{OPEN BASE}$ **then**
- 25: prepare C TODO
- 26: send (SUBMIT, C) to $\mathcal{G}_{\text{Ledger}}$
- 27: **else if** $State = \text{OPEN VIRTUAL}$ **then**
- 28: TODO
- 29: **end if**

Fig. 1.

Protocol $\Pi_{\text{Chan}} - \text{virtual}$

- 1: // notification to funder
- 2: // trust that *Alice* has c in her channel
- 3: On (FUND YOU, c , *Bob*) by *Charlie* as input:
- 4: ensure $State = \text{INIT}$
- 5: $State \leftarrow \text{OPENING VIRTUAL CHANNEL}$
- 6: do LN with *Bob* – TODO
- 7: $State \leftarrow \text{OPEN VIRTUAL}$
- 8: output (OK) to *Charlie*

- 9: On (FUND, c , hops, **sub_parties** = (fundee, counterparty), **outer_parties** = (*Alice*, *Dave*), $pk_{VA,out}$, $pk_{VB,out}$) by \mathcal{E} :
- 10: do the same as in $\mathcal{F}_{\text{Chan}}$, Fig. 8, lines 1-11, skipping line 6 // “as *Alice*”
 sender labels are applied anyway, since we *are Alice*
- 11: do VChan() with hops – TODO // $P_{i-1}P_i, P_iP_{i+1}$ and all P_1P_n held by
 BOTH R_{i-1} and L_i . $P_{i-1}P_i$ held only by R_{i-1} , P_iP_{i+1} held only by L_i . This
 (probably) ensures that only relevant parties can close their channels (with the
 exception of honest R_{i-1} wanting to leave channels virtual but corrupted L_i
 demoting them to base, which however doesn’t cost funds to anyone), but that
 they have minimal impact to the decisions of adjacent channels. All $P_{i-1}P_i$
 inputs must be signed by R_{i-1} and all P_iP_{i+1} inputs by L_i .
- 12: do the same as in $\mathcal{F}_{\text{Chan}}$, Fig. 8, lines 19-26
- 13: output (OK) to \mathcal{E}

- 14: // notification to fundee
- 15: On (ALLOW FUND, ...) by *Charlie*, act as $\mathcal{F}_{\text{Chan}}$ (Fig 8, line 9):

Fig. 2.

Functionality $\mathcal{F}_{\text{Chan}}$ – init & top up

- 1: Initialisation: // runs on first activation
- 2: $State \leftarrow \text{INIT}$
- 3: $(\text{locked}_A, \text{locked}_B) \leftarrow (0, 0)$

- 4: On (TOP UP, c_{\min}) by *Alice*:
- 5: ensure $State = \text{INIT}$
- 6: $State \leftarrow \text{SENT KEY}$
- 7: $(sk, pk) \leftarrow \text{KEYGEN}()$
- 8: output (PUBLIC KEY, pk) to *Alice*

- 9: On (CHECK TOP UP) by *Alice*:
- 10: ensure $State = \text{SENT KEY}$
- 11: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and assign output to Σ
- 12: ensure $\exists tx \in \Sigma, c_{\text{on}} : c_{\text{on}} \geq c_{\min} \wedge (c_{\text{on}}, pk) \in tx.\text{outputs}$
- 13: **base_output** $\leftarrow (c_{\text{on}}, pk)$ of tx
- 14: $State \leftarrow (\text{TOPPED UP}, \text{Alice})$
- 15: output (TOPPED UP) to *Alice*

Fig. 3.

Functionality $\mathcal{F}_{\text{Chan}} - \text{base}$

- 1: On (OPEN, c_F , $pk_{A,\text{out}}$, $pk_{B,\text{out}}$) by *Alice*:
- 2: ensure $State = (\text{TOPPED UP}, Alice)$
- 3: ensure $c_F = c_{\text{on}}$
- 4: $(sk_{A,F}, pk_{A,F}) \leftarrow \text{KEYGEN}(); (sk_{B,F}, pk_{B,F}) \leftarrow \text{KEYGEN}()$
- 5: $F \leftarrow \text{TX } \{\text{input: base_output, output: } (c_F, 2/\{pk_{A,F}, pk_{B,F}\})\}$
- 6: $\text{sig}_F \leftarrow \text{SIGN}(F, sk)$
- 7: $State \leftarrow \text{WAITING FOR LEDGER}$
- 8: send (OPEN, c_F , $pk_{A,\text{out}}$, $pk_{B,\text{out}}$, F , sig_F , *Alice*) to \mathcal{A}

- 9: On (CHECK FUNDING) by *Alice*:
- 10: ensure $State = \text{WAITING FOR LEDGER}$
- 11: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and assign output to Σ
- 12: ensure $F \in \Sigma$
- 13: $c_A \leftarrow c$; $c_B \leftarrow 0$
- 14: $State \leftarrow \text{OPEN BASE}$
- 15: output (OPEN SUCCESS) to *Alice*

- 16: On (PAY, x) by *Dave* $\in \{Alice, Bob\}$:
- 17: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 18: ensure $c_D - \text{locked}_D \geq x$
- 19: send (PAY, x , *Dave*) to \mathcal{A} and expect reply (OK)
- 20: $c_D \leftarrow c_D - x$; $c_{\bar{D}} \leftarrow c_{\bar{D}} + x$ // \bar{D} is *Alice* if D is *Bob* and vice-versa
- 21: output (PAY SUCCESS) to *Dave*

- 22: On (BALANCE) by *Dave* $\in \{Alice, Bob\}$:
- 23: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 24: output (BALANCE, $(c_A, c_B, \text{locked}_A, \text{locked}_B)$) to *Dave*

Fig. 4.

Functionality $\mathcal{F}_{\text{Chan}}$ – close Pt. 1

```

1: On (CLOSE) by Alice:
2:   if  $State = \text{OPEN BASE}$  then
3:     if both channel parties are honest then
4:        $C \leftarrow \text{TX}$  {input:  $F.\text{output}$ , outputs:  $(c_A, pk_{A,\text{out}} \wedge t), (c_B, pk_{B,\text{out}})$ }
5:        $\text{sig}_{B,C} \leftarrow \text{SIGN}(C, sk_{B,F})$ 
6:     end if // note that rest is run only if Alice is honest
7:      $\text{sig}_{A,C} \leftarrow \text{SIGN}(C, sk_{A,F})$ 
8:      $State \leftarrow \text{CLOSED}$ 
9:     input (SUBMIT,  $(C, \text{sig}_{A,C}, \text{sig}_{B,C})$ ) to  $\mathcal{G}_{\text{Ledger}}$ 
10:  else if  $State = \text{OPEN VIRTUAL}$  then TODO: think more about closing w/
    one corrupted party
11:     $State \leftarrow \text{CLOSED}$ 
12:    output (CLOSING,  $c_A, c_B$ ) to opener
13:  end if

14: On (CLOSE) by  $\mathcal{A}$ :
15:   ensure that exactly one party in  $\{Alice, Bob\}$  is corrupted
16:   wlog, let Alice corrupted, Bob honest
17:    $C' \leftarrow \text{TX}$  {input:  $F.\text{output}$ , outputs:  $(c_B, pk_{B,\text{out}}), (c_A, pk_{A,\text{out}} \wedge t)$ }
18:    $\text{sig}_{B,C'} \leftarrow \text{SIGN}(C', sk_{B,F})$ 
19:    $State \leftarrow \text{CLOSED}$ 
20:   send  $(C', \text{sig}_{B,C'})$  to  $\mathcal{A}$ 
21:   TODO: asynchronously, if a lot of time passes and  $C'$  not in  $\mathcal{G}_{\text{Ledger}}$ ,
    submit  $C$  to  $\mathcal{G}_{\text{Ledger}}$  for good measure

```

Fig. 5.

Functionality $\mathcal{F}_{\text{Chan}}$ – close Pt. 2

- 1: On (CLOSING, c_{left} , c_{right}) by $\mathcal{F}_{\text{Chan}}$:
- 2: ensure $State \in \{\text{OPEN BASE, OPEN VIRTUAL}\}$
- 3: ensure $((c_L, c_R), \text{hops}, (Charlie, Dave), (Frank, George), \text{id}) \in \text{funded}$ with $Frank \in \{Alice, Bob\}$
- 4: ensure $c_{\text{left}} \leq c_L + c_R$
- 5: remove entry from **funded**
- 6: output (CLOSED VIRTUAL, c_{right} , id) to *Frank*

- 7: On (CLOSED VIRTUAL, c_{right} , id) by $\mathcal{F}_{\text{Chan}}$:
- 8: ensure $State \in \{\text{OPEN BASE, OPEN VIRTUAL}\}$
- 9: ensure $(\text{virtual}, c, \mathcal{F}_{\text{Chan}}, Dave, \text{id}) \in \text{funded}$
- 10: ensure $c_{\text{right}} \leq c$
- 11: send (CLOSED) to virtual and expect reply (YES)
- 12: $c_D \leftarrow c_D + c_{\text{right}}$
- 13: remove entry from **funded**

- 14: On (CLOSED) by P :
- 15: **if** $State = \text{CLOSED}$ **then**
- 16: send (YES) to P
- 17: **else**
- 18: send (NO) to P
- 19: **end if**

Fig. 6.

Functionality $\mathcal{F}_{\text{Chan}} - \text{fund virtual}$

```

1: On (FUND,  $c$ , hops, sub_parties = (fundee, counterparty), outer_parties =
   ( $Alice$ ,  $Dave$ ),  $pk_{VA,out}$ ,  $pk_{VB,out}$ ) by  $Alice$ : // we fund another channel
   TODO: use  $pk_{VA}$  in virtual LN()
2:   ensure  $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$ 
3:   ensure  $c_A - \text{locked}_A \geq c$ 
4:   input (FUND YOU,  $c$ , counterparty) to fundee as  $Alice$ , ensure output is
   (OK)
5:   generate random id
6:   send (FUND  $c$ , hops, sub_parties = (fundee, counterparty), outer_parties
   = ( $Alice$ ,  $Dave$ ), funder =  $Alice$ , id) to  $\mathcal{A}$  and ensure reply is (OK)
7:   ( $L_0, R_0$ )  $\leftarrow$  ( $Alice$ ,  $Bob$ )
8:   for all ( $L_i, R_i$ )  $\in$  hops do //  $i \in \{1, \dots, |\text{hops}|\}$ 
9:     ensure  $R_{i-1} = L_i$ 
10:    send (ALLOW FUND,  $c$ , sub_parties, local_funder  $\leftarrow L_i$ , id,
    $i \stackrel{?}{=} |\text{hops}|$ ) to  $L_i$  as  $Alice$  and ensure reply is (OK)
11:  end for
12:  if both channel parties are honest then
13:    send (IS OPEN SUCCESSFUL, id) to  $\mathcal{A}$  and ensure reply is (OK)
14:  else if only  $Alice$  is honest then
15:    ( $sk_{A,V}, pk_{A,V}$ )  $\leftarrow$  KEYGEN()
16:    send (UPDATE TO VIRTUAL,  $pk_{A,V}$ ) to  $\mathcal{A}$  and assign reply to ( $V = \text{TX}$ 
   {input:  $F$ .output, outputs: ( $c_A + c_B - c$ ,  $2/\{pk_{A,V}, pk_{B,V}\}$ ), ( $c$ ,
    $2/\{pk_{G,V}, pk_{A,V}\}$ ), ( $0$ ,  $|\text{hops}|/\{pk_i\}_i$ ),  $\text{sig}_{B,V}$ ,  $C' = \text{TX}$  {input:
    $V$ .outputs.0, outputs: ( $c_A - \text{locked}_A - c$ ,  $pk_{A,out} \wedge t$ ), ( $c_B - \text{locked}_B$ ,  $pk_{B,out}$ )},
    $\text{sig}_{B,C'}$ ) TODO: think about locked coins
17:    ensure  $\text{VERIFY}(V, \text{sig}_{B,V}, pk_{B,F}) = \text{VERIFY}(C', \text{sig}_{B,C'}, pk_{A,V}) = \text{True}$ 
18:  end if
19:  for all ( $L_i, R_i$ )  $\in$  hops do //  $i \in \{1, \dots, |\text{hops}|\}$ 
20:    send (FUND DONE, id) to  $L_i$  as  $Alice$  and ensure reply is (OK)
21:  end for
22:   $c_A \leftarrow c_A - c$ 
23:  if only  $Alice$  is honest then
24:     $C \leftarrow C'$ ;  $\text{sig}_{B,C} \leftarrow \text{sig}_{B,C'}$ 
25:  end if
26:  add ( $(c, 0)$ , hops, sub_parties, outer_parties, id) to funded
27:  output (OK) to  $Alice$ 

```

Fig. 7.

Functionality $\mathcal{F}_{\text{Chan}} - \text{virtual}$

- 1: On (FUND YOU, c , $Dave$) by *Charlie* as input to *Alice*: // *Alice* is funded by *Charlie*
- 2: ensure $State = \text{INIT}$
- 3: $Bob \leftarrow Dave$
- 4: send (FUND YOU, c , Bob , *Charlie*, *Alice*) to \mathcal{A} and ensure reply is (OK)
- 5: $c_A \leftarrow c$; $c_B \leftarrow 0$
- 6: **opener** \leftarrow *Charlie*
- 7: $State \leftarrow \text{OPEN VIRTUAL}$
- 8: output (OK) to *Charlie*

- 9: On (ALLOW FUND, c , **sub_parties**, $Dave$, **id**, **is_last**) by *Charlie*:
- 10: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 11: ensure $Dave \in \{Alice, Bob\}$
- 12: ensure $c_D - \text{locked}_D \geq c$
- 13: output received message to $Dave$ and ensure reply is (OK)
- 14: $\text{locked}_D \leftarrow \text{locked}_D + c$
- 15: add (**id**, **is_last**, **sub_parties**, c , $Dave$) to **pending**
- 16: send (OK) to *Charlie*

- 17: On (FUND DONE, **id**) by *Charlie*:
- 18: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 19: ensure (**id**, **is_last**, **sub_parties**, c , $Dave$) \in **pending**
- 20: remove (**id**, **is_last**, **sub_parties**, c , $Dave$) from **pending**
- 21: **if is_last then**
- 22: add $((0, c), \perp, \text{sub_parties.reverse()}, (Dave, \perp), \text{id})$ to **funded**
- 23: **end if**
- 24: send (OK) to *Charlie*

Fig. 8.

Functionality $\mathcal{F}_{\text{Chan}} - \text{corruption}$

- 1: On (CORRUPT) by P , addressed to $Alice$:
- 2: ensure $P \in \{\text{opener}, \mathcal{A}\}$
- 3: **for all** $(_, _, (\text{funder}, _), (Alice, _)) \in \text{funded}$ **do**
- 4: send (CORRUPT) to funder and ensure reply is (OK)
- 5: **end for**
- 6: from now on, allow \mathcal{A} to handle all $Alice$'s messages, i.e. act as a relay
- 7: **if** Bob is not corrupted **then**
- 8: from now on, override reactions to messages (OPEN) and (PAY) addressed to Bob with those defined in the current Figure:
- 9: **end if**
- 10: output (OK) to P

- 11: On (OPEN, c_F , $pk_{A,\text{out}}$, $pk_{B,\text{out}}$) by Bob :
- 12: ensure $State = (\text{TOPPED UP}, Bob)$ **TODO: decide what happens when channel funded by corrupted party**
- 13: ensure $c_F = c_{\text{on}}$
- 14: $(sk_{B,F}, pk_{B,F}) \leftarrow \text{KEYGEN}()$
- 15: send (OPEN, c_F , $pk_{A,\text{out}}$, $pk_{B,\text{out}}$, $pk_{B,F}$, Bob) to \mathcal{A} , assign reply to $(pk_{A,F}, C = \text{TX} \{\text{input: } F.\text{output}, \text{outputs: } (c_F, pk_{B,\text{out}} \wedge t), (0, pk_{A,\text{out}})\}, \text{sig}_{A,C})$
- 16: ensure $\text{VERIFY}(C, \text{sig}_{A,C}, pk_{A,F}) = \text{True}$
- 17: $F \leftarrow \text{TX} \{\text{input: base_output}, \text{output: } (c_F, 2/\{pk_{A,F}, pk_{B,F}\})\}$
- 18: $\text{sig}_F \leftarrow \text{SIGN}(F, sk)$
- 19: $State \leftarrow \text{WAITING FOR LEDGER}$
- 20: input (SUBMIT, (F, sig_F)) to $\mathcal{G}_{\text{Ledger}}$

- 21: On (PAY, x) by Bob :
- 22: ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
- 23: ensure $c_B - \text{locked}_B \geq x$
- 24: send (PAY, x , Bob) to \mathcal{A} and assign reply to $(C' = \text{TX} \{\text{input: } F.\text{output}, \text{outputs: } (c_A + x, pk_{A,\text{out}}), (c_B - \text{locked}_B - x, pk_{B,\text{out}} \wedge t)\}, \text{sig}'_{A,C})$ **TODO: think about locked coins again**
- 25: ensure $\text{VERIFY}(C', \text{sig}'_{A,C}, pk_{A,F}) = \text{True}$
- 26: $C \leftarrow C'$; $\text{sig}_{A,C} \leftarrow \text{sig}'_{A,C}$
- 27: $c_B \leftarrow c_B - x$; $c_A \leftarrow c_A + x$
- 28: output (PAY SUCCESS) to Bob

Fig. 9.

Simulator \mathcal{S} – Pt. 1

- 1: On $(\text{OPEN}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F \text{ Alice})$ by $\mathcal{F}_{\text{Chan}}$: // both honest
- 2: simulate *Alice* receiving input $(\text{OPEN}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}})$ by \mathcal{E}
- 3: ensure simulated *Alice* inputs $(\text{SUBMIT}, (F', \text{sig}_{F'}))$ to $\mathcal{G}_{\text{Ledger}}$
- 4: input $(\text{SUBMIT}, (F, \text{sig}_F))$ to $\mathcal{G}_{\text{Ledger}}$

- 5: On $(\text{OPEN}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, pk_{B,F}, \text{Bob})$ by $\mathcal{F}_{\text{Chan}}$: // *Alice* corrupted
- 6: send LN message $(\text{OPEN}, pk_{B,F})$ to *Alice* and relay reply to $\mathcal{F}_{\text{Chan}}$ **TODO:**
 change msg to fit LN, ensure *Alice* doesn't see a difference from real world

- 7: On $(\text{PAY}, x, \text{Dave})$ by $\mathcal{F}_{\text{Chan}}$:
- 8: **if** both channel parties are honest **then**
- 9: simulate *Dave* receiving input (PAY, x) by \mathcal{E}
- 10: ensure simulated *Dave* outputs (OK)
- 11: send (OK) to $\mathcal{F}_{\text{Chan}}$
- 12: **else if** only *Dave's* counterparty is corrupted **then** // else just relay to \mathcal{A}
- 13: simulate *Dave* receiving input (PAY, x) by \mathcal{E}
- 14: ensure simulated *Dave* outputs (OK)
- 15: extract the latest commitment transaction C and its signature by
 Dave's counterparty $\text{sig}_{\bar{D},C}$ from simulated *Dave's* state
- 16: send $(C, \text{sig}_{\bar{D},C})$ to $\mathcal{F}_{\text{Chan}}$
- 17: **end if**

- 18: On $(\text{FUND YOU}, c, \text{Bob}, \text{Charlie}, \text{Alice})$ by $\mathcal{F}_{\text{Chan}}$:
- 19: simulate *Alice* receiving input $(\text{FUND YOU}, c, \text{Bob})$ by *Charlie*
- 20: ensure simulated *Alice* outputs (OK) to *Charlie*
- 21: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 22: On $(\text{FUND } c, \text{hops}, \text{sub_parties} = (\text{fundee}, \text{counterparty}), \text{outer_parties} = (\text{Charlie}, \text{Dave}), \text{funder} = \text{Alice}, \text{id})$ by $\mathcal{F}_{\text{Chan}}$:
- 23: add the message data to **virtual_opening**
- 24: simulate execution of line 7 of Fig. 8 with *Alice* // \mathcal{S} knows *Bob* (*Alice's*
 counterparty) through opening procedure
- 25: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 26: On $(\text{ALLOW FUND}, c, \text{sub_parties}, \text{local_funder} = L_i, \text{id}, i \stackrel{?}{=} |\text{hops}|)$ by
 $\mathcal{F}_{\text{Chan}}$'s *Alice* to *Charlie*:
- 27: simulate receiving message with *Charlie* by *Alice* and all subsequent
 communication
- 28: ensure the simulated *Charlie* sends (OK) to the simulated *Alice*
- 29: intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 10.

Simulator \mathcal{S} – Pt. 2

- 1: On (IS OPEN SUCCESSFUL, id) by $\mathcal{F}_{\text{Chan}}$:
- 2: retrieve and remove from **virtual_opening** the data marked with id
- 3: simulate line 11 of Fig. 2 with *Alice* using this data
- 4: ensure *Alice* completes execution of VChan() successfully
- 5: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 6: On (UPDATE TO VIRTUAL) by $\mathcal{F}_{\text{Chan}}$:
- 7: retrieve and remove from **virtual_opening** the data marked with id
- 8: simulate line 11 of Fig. 2 with *Alice* using this data
- 9: ensure *Alice* completes execution of VChan() successfully
- 10: extract from *Alice*'s state the new virtual funding TX V for pre-existing channel
- 11: extract from *Alice*'s state the new commitment TX C that spends the on-chain funding TX
- 12: send (V, C) to $\mathcal{F}_{\text{Chan}}$

- 13: On (FUND DONE, id) by $\mathcal{F}_{\text{Chan}}$'s *Alice* to *Charlie*:
- 14: simulate receiving message with *Charlie* by *Alice* and all subsequent communication
- 15: ensure the simulated *Charlie* sends (OK) to the simulated *Alice*
- 16: intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 11.

1 Security Proof

When \mathcal{E} sends (FUND, c , hops, (fundee, counterparty), (*Charlie*, *Dave*)) to *Alice* in the real world, lines 1-4 of Fig. 8 are executed and then control is handed over to the “fundee” ITI, which executes lines 3-8 of Fig. 2. This ITI will output (OK) if and only if line 6 of Fig. 2 succeeds.

When \mathcal{E} sends (FUND, c , hops, (fundee, counterparty), (*Charlie*, *Dave*)) to *Alice* in the ideal world, lines 1-4 of Fig. 8 are executed and then control is handed over to the functionality that controls the “fundee”, which executes lines 1-4 of Fig. 8 and then hands control over to \mathcal{S} . The latter in turn simulates lines 3-8 of Fig. 2, thus following the exact same steps as in the real world, therefore it will send (OK) to $\mathcal{F}_{\text{Chan}}$ if and only if the simulated line 6 of Fig. 2 succeeds. From this and the previous paragraph, we see that, up to this point, the two worlds are perfectly indistinguishable.

Moving on, in the ideal world subsequently lines 5-6 of Fig. 8 are executed, which results in \mathcal{S} executing lines 22-25 of Fig. 10. During the latter steps, \mathcal{S} simulates executing line 7 of Fig. 8 with *Alice*.

Similarly in the real world, *Alice* executes lines 5 and 7 of Fig. 8, therefore the two worlds still are perfectly indistinguishable.

The “for” loop of lines 8-11 of Fig. 8 is then executed in both the real and the ideal worlds. The message of line 10 results in the execution of lines 3-8 of Fig. 2 by L_i in both worlds: in the real world directly, in the ideal world simulated by S .

In the ideal world, line 18 in Fig. 8 prompts S to simulate line 11 of Fig. 2 with *Alice*, which is exactly the code that would be directly run by *Alice* in the real world. Therefore the two worlds remain perfectly indistinguishable.

The “for” loop of lines 19-26 of Fig. 8 is also perfectly indistinguishable in the two worlds. With argumentation similar to that of the previous “for” loop, we conclude that the FUND message does not induce any chance of distinguishability between the two worlds.

References