What is Trust

Orfeas Stefanos Thyfronitis Litos

University of Edinburgh
o.thyfronitis@ed.ac.uk

Abstract. We will try to define all the abstract properties that we would like "Trust" to have.
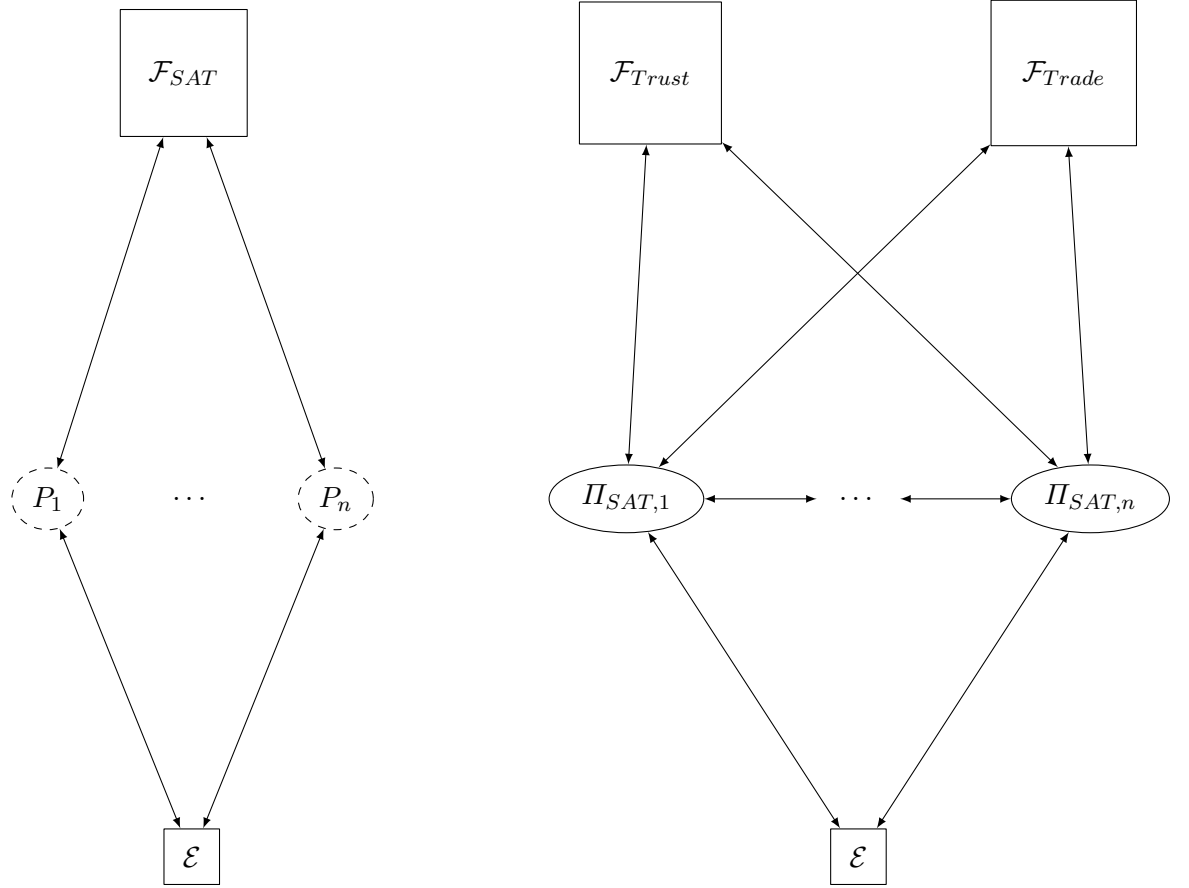
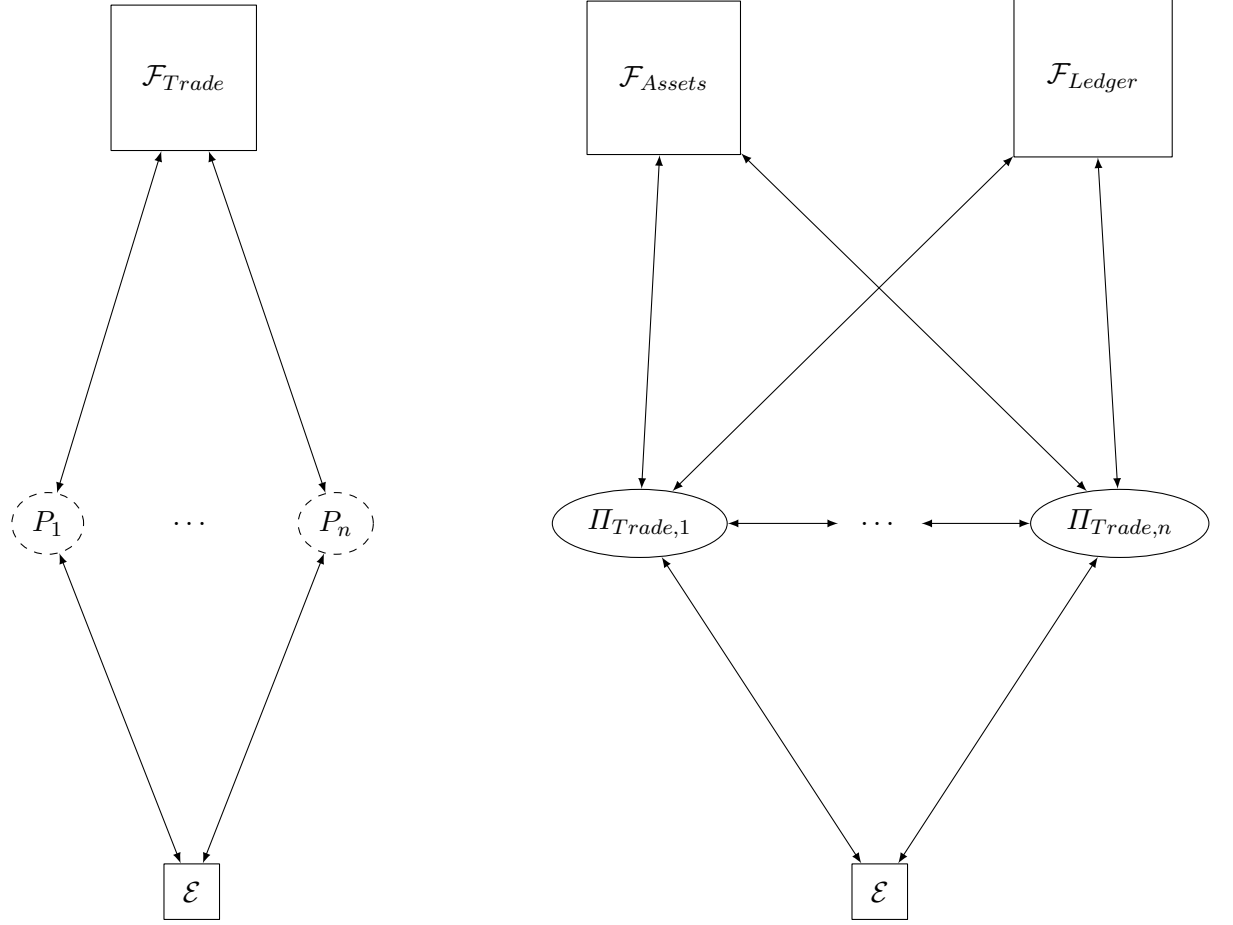# 1 High-level idea

Fig. 1: (Almost) all functionalities

Fig. 2: Trade functionality and protocol

2 Utility Function Properties

Before *Alice* (an ITM that executes Π_{SAT}) can take any action, she must be assigned a utility function, U_{Alice}, and a discount function, λ_{Alice}, by $\mathcal{E}$.

Definition 1 (Utility Function). $U_{Alice} : Money \times multiset\,(Asset) \times Rep \rightarrow Utility$, *where* $Money, Rep \in \mathbb{N}, Utility \in \mathbb{R}$.

The utility function is strictly increasing with respect to the integer *Money* and may do anything with respect to the quantity of any single

Asset or combination of *Asset*s. It is also strictly increasing with respect to the integer *Rep*.

Definition 2 (Reputation function). *Reputation* $: \mathcal{P} \times Time \times 2^{Asset} \rightarrow Rep$, *where* $Time, Rep \in \mathbb{N}$

$$Rep\,(Alice, t, set) > Rep\,(Bob, t, set) \overset{def}{\Longleftrightarrow}$$
$$\Pr\,[Alice \text{ receives message } (\text{canYouSatisfy}, d) \text{ at time t}] >$$
$$\Pr\,[Bob \text{ receives message } (\text{canYouSatisfy}, d) \text{ at time t}]$$
$$\text{where } d \in set$$

Rep is an integer that corresponds to *Alice*'s perceived rank in a partial ordering of all the players where *Alice* assumes that she has a higher rank than *Bob* if the average probability of third players choosing her before *Bob* when they want to buy something amongst a list of *Asset*s that *Alice* is interested in selling is higher than *Bob*'s corresponding probability for the same *Asset*s, as perceived by *Alice*.

Definition 3 (Discount function). $\lambda_{Alice} : Time \rightarrow [0, 1]$, *where* $Time \in \mathbb{N}$ *and* λ *is strictly decreasing.*

Definition 4 (State). *Let* $state_{Alice} : Time \rightarrow Money \times multiset\,(Asset) \times Rep$.

Alice's target is to maximise

$$\sum_{t=1}^{\infty} \lambda_{Alice}\,(t) \left(\sum_{\substack{(m,a,r)\in \\ Money \times mset(Asset) \times Rep}} U_{Alice}\,(m, a, r) \cdot \Pr\,[state_{Alice}\,(t) = (m, a, r)] \right) \quad .$$

3   Protocol

In this section we will describe the real protocol executed by the players in the absence of the $\mathcal{F}_{SAT}$ ideal functionality. The description of this protocol does not need any utility function; all the "important" decisions are taken by the environment.

Consider an environment $\mathcal{E}$, and adversary $\mathcal{A}$ and n players executing copies of the same protocol $\Pi_1, \ldots, \Pi_n$. $\mathcal{E}$ can send the following messages to Π_i:

1. Manage player desires
   (a) Satisfy $d \in \mathcal{D}$ through a player in $L \subseteq [n]$
   (b) Abort attempt to satisfy $d \in \mathcal{D}$
2. Manage offered desires satisfaction
   (a) Gain the ability to satisfy $d \in \mathcal{D}$ for players in $L \subseteq [n]$ for a price $x \in \mathbb{N}$
   (b) Lose the ability to satisfy $d \in \mathcal{D}$ for players in $L \subseteq [n]$ for a price $x \in \mathbb{N}$
3. Satisfy another player's desire
   (a) Satisfy player's $i \in [n]$ desire $d \in \mathcal{D}$ with the corresponding satisfaction string s
   (b) Satisfy player's $i \in [n]$ desire $d \in \mathcal{D}$ with the satisfaction string s' (normally suitable for satisfying $d' \neq d, d' \in \mathcal{D}$)
   (c) Ignore player's $i \in [n]$ desire $d \in \mathcal{D}$
4. Manage direct trusts
   (a) Increase direct trust to player $i \in [n]$ by $x \in \mathbb{N}$
   (b) Decrease direct trust to player $i \in [n]$ by $x \in \mathbb{N}$
   (c) Steal direct trust $x \in \mathbb{N}$ from player $i \in [n]$

Some of these messages (e.g. 1b) are meaningful only when some other messages have been delivered previously (e.g. 1a). $\mathcal{E}$ may send such messages even when they are not meaningful; the protocol should take care to reject/ignore such messages.

Let $i \in [n]$. Π_i can send the following messages to $\mathcal{E}$:

1. No player in L can satisfy my desire $d \in \mathcal{D}$
2. Desire $d \in \mathcal{D}$ made available for satisfaction amongst $L \subseteq [n]$ for price $x \in \mathbb{N}$
3. Desire $d \in \mathcal{D}$ made unavailable for satisfaction amongst $L \subseteq [n]$ for price $x \in \mathbb{N}$
4. Payment $x \in \mathbb{N}$ has been sent to player $j \in [n]$ for the satisfaction of desire $d \in \mathcal{D}$
5. Correct Payment $x \in \mathbb{N}$ from player $j \in [n]$ for the satisfaction of desire $d \in \mathcal{D}$ has been received
6. Wrong Payment $x \in \mathbb{N}$ from player $j \in [n]$ for the satisfaction of desire $d \in \mathcal{D}$ has been received
7. Player $j \in [n]$ has satisfied my desire $d \in \mathcal{D}$ with satisfaction string s
8. Player $j \in [n]$ has partially satisfied my desire $d \in \mathcal{D}$ with satisfaction string s' (normally suitable for satisfying $d' \neq d, d' \in \mathcal{D}$)
9. Player $j \in [n]$ has ignored my desire $d \in \mathcal{D}$
10. Direct trust to player $i \in [n]$ increased by $x \in \mathbb{N}$

11. Direct trust to player $i \in [n]$ decreased by $x \in \mathbb{N}$
12. Stole $x \in \mathbb{N}$ from player's $i \in [n]$ direct trust

Π_i should only send these messages when $\mathcal{E}$ is expecting them.

Let $i, j \in [n]$ The messages that can be sent between Π_i and Π_j are the following:

1. Can you satisfy $d \in \mathcal{D}$?
2. I can satisfy $d \in \mathcal{D}$ for a price $x \in \mathbb{N}$
3. I cannot satisfy $d \in \mathcal{D}$
4. Payment of $x \in \mathbb{N}$ for satisfaction of $d \in \mathcal{D}$ sent
5. Satisfaction string s', response to payment of $x \in \mathbb{N}$ for $d \in \mathcal{D}$

Π_i is supposed to send 4 when it has already paid through $\mathcal{F}_{Ledger}$. Similarly, it is supposed to send 5 when it has verified that the other party has sent the corresponding payment on $\mathcal{F}_{Ledger}$.

Going in more detail, the actual protocol is as follows:

Π_{SAT}

```
1  Initalization:
2    util = ⊥
3
4  Upon receiving (type, t) from ℰ:
5    util = t
6
7  Upon receiving message (satisfy, d, L) from ℰ:
8    If util == ⊥:
9      send message (utilityNotSet) to ℰ
10     go to Idle state
11   aux = L
12   While (aux ≠ ∅):
13     send message (chooseBestSeller, d, aux) to 𝓕_Trust
14     wait for response1 from 𝓕_Trust
15     If response1 == (bestSeller, d, L, Bob):
16       send message (canYouSatisfy, d) to Bob
17       wait for response2 from Bob
18       If response2 == (IcanSatisfy, d, x, s):
19         If util(state ∪ s − x) > util(state) and s ∈ d
20           send message (trade, x, s, Bob) to 𝓕_Trade
21           wait for response3 from 𝓕_Trade
22             If response3 == (traded, x, s, Bob):
23               send message (satisfied, d, x, s, Bob) to 𝓕_Trust
```

```
24              # maybe send only utility difference to F_Trust
25              send message (satisfied, d, L) to E
26              go to Idle state
27            Else If response3 == (cheated, x, s, Bob):
28              send message (cheated, d, x, s, Bob) to F_Trust
29              send message (cheated, d, L) to E
30              go to Idle state
31     Else: # if response1 == ⊥
32       send message (unsatisfied, d, L) to E
33       go to Idle state
34     aux = aux \ {Bob} # only when response2 is not good
35   send message (unsatisfied, d, L) to E

37 Upon receiving message (obtain, s) from E:
38   send message (obtain, s) to F_Trade
39   wait for response from F_Trade
40   If response == (obtained, s):
41     send message (obtained, s) to E
42   Else:
43     send message (notObtained, s) to E

45 Upon receiving message (lose, s) from E:
46   send message (lose, s) to F_Trade
47   wait for response from F_Trade
48   If response == (lost, s):
49     send message (lost, s) to E
50   Else:
51     send message (notLost, s) to E

53 Upon receiving message (canYouSatisfy, d) from Alice:
54   If util == ⊥:
55     ignore request, go to Idle state
56   If (util(state + x + badRep) > util(state) or
57       util(state \ s + x + goodRep) > util(state)) and s ∈ d:
58     send message (IcanSatisfy, d, x, s) to Alice # x is the
           price
59   Else:
60     send message (IcannotSatisfy, d) to Alice

62 Upon receiving message (willWeCheat, x, s, Alice) from F_Trade:
```

```
63    If util == ⊥: # (Unreachable since we've already engaged)
64      ignore request, go to Idle state
65    If util(state \ s + x + goodRep) ≥ util(state + x + badRep):
66      send message (doNotCheat, x, s, Alice) to F_Trade
67    Else:
68      send message (cheat, x, s, Alice) to F_Trade
```

4 Desire Satisfaction Ideal Functionality

Following the UC paradigm, in this section we define the ideal functionality for desire satisfaction, $\mathcal{F}_{SAT}$. In this setting, all the desires that are generated by the environment and are input to the players are immediately forwarded to $\mathcal{F}_{SAT}$; the functionality decides which desires to satisfy. Since the players are dummy and all desires are satisfied by the functionality, no trust semantics amongst the players are necessary.

Nevertheless, given that all desires have a minimum cost, the cost semantics are still necessary, as we show with the following example: Consider a set of desires D with more elements than the total number of tokens all players have. D could never be satisfied by the players because of the high total cost, but a $\mathcal{F}_{SAT}$ with no consideration for cost could in principle satisfy all desires in D.

The functionality can calculate the properties and functions defined in **??**, **??** and **??** for all inputs at any moment in time.

Without knowledge of the utilities the environment is going to give to each satisfied desire, the functionality may fail spectacularly. So knowledge of the utility of each desire, or at least some function of the utility given the desires is needed. We can assume that $\mathcal{F}_{SAT}$ knows U or an approximation of it.

Going into more detail, $\mathcal{F}_{SAT}$ is a stateful process that acts as a market and as a bank for the players. The market does not offer a particular product for the same price to all users; For some users it may be cheaper than for others, reflecting the fact that some players can realize some desires more efficiently than others.

$\mathcal{F}_{SAT}$ stores a number for each player that represents the amount of tokens this player has and a table with the price of each desire for each player. It provides the functions $cost\,(u, d)$ which returns the cost of the desire d for player u with no side effects, $sat\,(u, d)$ that returns the string that satisfies the desire d to u and reduces the amount of the tokens belonging to u by $cost\,(u, d)$. There exists also the function $transfer\,(u_1, u_2, t)$ which reduces the amount of tokens u_1 has by t and

8

increases the tokens of u_2 by t, given that initially the tokens belonging to u_1 were equal or more than t. This function is private to the functionality, thus can be used only internally.

$\mathcal{F}_{SAT}$

```
1   Initialisation:
2     ∀Alice ∈ P,
3       util(Alice) = ⊥
4       assets(Alice) = ⊥
5
6   Upon receiving (type, t) from Alice:
7     util(Alice) = t
8
9   Upon receiving (satisfy, d, L) from Alice:
10    If util(Alice) == ⊥:
11      send message (utilityNotSet) to Alice
12      go to Idle state
13    Find list = {(Bob, x, s) ∈ L × ℝ × Assets :
14      s ∈ assets(Bob) and s ∈ d and x ≥ 0 and
15      Alice has at least x coins available and
16      util(Alice)(state_Alice ∪ s − x) > util(Alice)(state_Alice)
17      and
18      util(Bob)(state_Bob \ s + x + goodRep) > util(Bob)(state_Bob)
19      and
20      util(Bob)(state_Bob \ s + x + goodRep) >
              util(Bob)(state_Bob + x + badRep) }
21    send (chooseBestSeller, list, d, Alice) to A
22    wait for response from A
23    With response as (bestSeller, list, Bob, x, s), Bob ∈ P:
24      If x > 0:
25        Pay x from Alice to Bob
26      assets(Bob) = assets(Bob) \ {s}
27      assets(Alice) = assets(Alice) ∪ {s}
28      send message (satisfied, d, L) to Alice
29    Else If response ≠ (bestSeller, list, Bob, x, s): # e.g.
            Bob = ⊥
30      send message (unsatisfied, d, L) to Alice
31
32  Upon receiving (obtain, s) from Alice:
33    assets(Alice) = assets(Alice) ∪ {s}
34    send message (obtained, s) to Alice
```

```
35
36  Upon receiving (lose, s) from Alice:
37    assets(Alice) = assets(Alice)\{s}
38    send message (lost, s) to Alice


    F_Trade
1   Initialisation:
2     ∀Alice ∈ P,
3       assets(Alice) = ⊥

4
5   Upon receiving (trade, ours, theirs, Bob) from Alice:
6     If not isAvailable(ours, Alice):
7       send message (youDontHave, ours) to Alice
8       go to Idle state
9     If transfer(ours, Alice, Bob) == True:
10      send message (willWeCheat, ours, theirs, Alice) to Bob
11      wait for response from Bob
12      If (response == (complete, x, s, Alice) and
13        not isAvailable(s, Bob)) or
14        response ≠ (complete, x, s, Alice):
15          send message (youDontHave, s) to Bob
16          send message (cheated, ours, theirs, Bob) to Alice
17          go to Idle state
18      Else If (transfer, theirs, Bob, Alice) == True:
19          send message (traded, ours, theirs, Bob) to Alice
20    Else # failed to give (Unreachable for a good F_Ledger)
21      send message (failed, ours, theirs, Bob) to Alice
22
23  isAvailable(object, player):
24    If object is money:
25      send (doIhaveBalance, object) to F_Ledger as player
26      wait for response from F_Ledger
27      return response
28    Else: # object is asset
29      If object ∈ assets(player):
30        return True
31      Else:
32        return False
33
34  transfer(object, sender, receiver):
```

```
35    If isAvailable(object):
36      If object is money:
37        send (pay, object, receiver) to 𝓕_{Ledger} as sender
38        wait for response from 𝓕_{Ledger}
39        Upon receiving (paymentDone, object, receiver):
40          return True
41      Else: # object is asset
42        assets(sender) = assets(sender) \ {object}
43        assets(receiver) = assets(receiver) ∪ {object}
44        return True
45    return False
46
47  Upon receiving (obtain, s) from Alice:
48    assets(Alice) = assets(Alice)∪{s}
49    send message (obtained, s) to Alice
50
51  Upon receiving (lose, s) from Alice:
52    assets(Alice) = assets(Alice)\{s}
53    send message (lost, s) to Alice
```

$\mathcal{F}_{Assets}$

```
1  Initialisation:
2    ∀Alice ∈ 𝓟,
3      assets(Alice) = ⊥
4
5  Upon receiving (add, asset) from Alice:
6    assets(Alice) = assets(Alice)∪{asset}
7    send (added, asset) to Alice
8
9  Upon receiving (remove, asset) from Alice:
10    If asset ∈ assets(Alice):
11      assets(Alice) = assets(Alice)\{asset}
12      send (removed, asset) to Alice
13    Else:
14      send (unableToRemove, asset) to Alice
15
16  Upon receiving (howManyDoIhave, asset) from Alice:
17    send (youHave, assets(Alice).count(asset)) to Alice
18
19  Upon receiving (transfer, asset, Bob) from Alice:
```

```
20    If asset ∈ assets(Alice):
21      assets(Alice) = assets(Alice)\{asset}
22      assets(Alice) = assets(Bob)∪{asset}
23      send (transferred, asset, Bob) to Alice
24    Else:
25      send (unableToTransfer, asset, Bob) to Alice
```

Π_{Trade}

```
1  Upon receiving (trade, ours, theirs, Bob) from E:
2    Send (letsTrade, ours, theirs) to Bob
3    If transfer(ours, Bob) == True:
4      Send (transferred, ours, Bob) to Bob and E
5      Wait for response from Bob
6      If response == (transferred, theirs, Bob):
7        send message (traded, ours, theirs, Bob) to E
8      Else:
9        send message (cheated, ours, theirs, Bob) to E
10
11  Upon receiving (letsTrade, theirs, ours) from Bob:
12    Send (willWeCheat, theirs, ours, Bob) to E
13    Wait for response from E
14    If response is (doNotCheat, theirs, ours, Bob):
15      If (transfer, ours, Bob) == True:
16        Send (transferred, ours, Bob) to Bob and E
17
18  transfer(object, receiver):
19    If isAvailable(object):
20      If object is money:
21        send (pay, object, receiver) to F_{Ledger}
22        wait for response from F_{Ledger}
23        Upon receiving (paymentDone, object, receiver):
24          return True
25      Else: # object is asset
26        send (transfer, object, receiver) to F_{Assets}
27        wait for response from F_{Assets}
28        Upon receiving (transferDone, object, receiver):
29          return True
30    return False
31
32  isAvailable(object):
```

```
33    If object is money:
34        send (doIhaveBalance, object) to ℱ_Ledger
35        wait for response from ℱ_Ledger
36        return response
37    Else: # object is asset
38        Send (doIHave, object) to ℱ_Assets
39        wait for response from ℱ_Assets
40        If response == (youHave, object):
41            return True
42        Else:
43            return False
44
45  Upon receiving message (obtain, s) from 𝓔:
46      send message (add, s) to ℱ_Assets
47      wait for response from ℱ_Assets
48      If response == (added, s)
49          send message (obtained, s) to 𝓔
50      Else
51          send message (notObtained, s) to 𝓔
52
53  Upon receiving message (lose, s) from 𝓔:
54      send message (remove, s) to ℱ_Assets
55      wait for response from ℱ_Assets
56      If response == (removed, s)
57          send message (lost, s) to 𝓔
58      Else
59          send message (notLost, s) to 𝓔
```

5 $\mathcal{F}_{SAT}$ and Π_{SAT} are potentially distinguishable

Consider the hybrid world of Fig. 1 (right) with n ITMs executing Π_{SAT}, where $\mathcal{F}_{Trust}$ is replaced by $\mathcal{F}'_{Trust}$:

$\mathcal{F}'_{Trust}$

```
1  Upon receiving (chooseBestSeller, d, L) from Alice:
```
2      $Bob \xleftarrow{R} L \cup \{\bot\}$
```
3      send message (bestSeller, d, L, Bob) to Alice
```

We will show here that $\mathcal{E}$ can distinguish between $\mathcal{F}_{SAT}$ and a Π_{SAT} that uses $\mathcal{F}'_{Trust}$.

Distinguishability. Consider the following adversary and environment:

$\mathcal{A}$

```
1  Has oracle access to every player's Alice utility function
       util(Alice)
2
3  Upon receiving (chooseBestSeller, list, d, Alice):
4     Bob =  argmax { util(Alice)(state_Alice ∪ s − 1coin) :
              Bob∈list
5     s ∈ d ∧ s ∈ assets(Bob) ∧ Alice has 1 coin ∧
6     (util(Bob)(state_Bob \ s + 1coin + 100Rep) >
7     util(Bob)(state_Bob + 1coin − 100Rep)) ∧
8     util(Bob)(state_Bob \ s + 1coin + 100Rep) > util(Bob)(state_Bob)  }
9     return (bestSeller, list, Bob, 1, s)
```

$\mathcal{E}$ distinguisher

```
1  ∀Alice ∈ P:
2     util(Alice)({assets}, x, r) = |{assets}| + x/2 + 2^r
3     λ(Alice)(t) = 1/t²
4     send message (type, util(Alice), λ(Alice)) to Alice
5  Alice ←R P
6  Bob ←R P \ {Alice}
7  d ←R 2^Asset
8  s ←R d
9  send message (obtain, s) to Bob
10 L = {Bob}
11 send message (satisfy, d, L) to Alice
12
13 Upon receiving message (x, d, L) from Alice:
14    If x == satisfied:
15      return functionality
16    Else: # if x ∈ {cheated, unsatisfied}
17      return protocol
```

Because of the way $\mathcal{E}$ is built, there always exists a seller that can satisfy the desire of the buyer.

In case $\mathcal{E}$ interacts with $\mathcal{F}_{SAT}$, then $\mathcal{A}$ will always match the buyer and the seller because of the way the former is built. Additionally, the price of 1 is cheap enough for the buyer to want to buy the asset. Furthermore, the boost in reputation is enough for the seller to not want to cheat, thus

the exchange will always complete correctly and $\mathcal{E}$ will receive `satisfied` as response. $\mathcal{E}$ will always correctly output `functionality`.

On the other hand, in case $\mathcal{E}$ interacts with Π_{SAT}, then

1. $\mathcal{F}'_{Trust}$ does not modify the players' reputation and
2. $\mathcal{F}'_{Trust}$ does not choose players depending on their reputation.

Thus, if $\mathcal{F}'_{Trust}$ does not respond with $\bot$, it is always in the seller's interest to cheat and $\mathcal{E}$ will always receive `cheated` as response. If $\mathcal{F}'_{Trust}$ responds with $\bot$, the trade will not go through and $\mathcal{E}$ will receive `unsatisfied` as a response. In all cases $\mathcal{E}$ will correctly output `protocol`. $\qquad\square$

References