Non-repudiation protocols

Outline and objective

- outline
  - introduction, definitions
  - classification of non-repudiation protocols
  - example protocols

- the objective is to understand
  - what does non-repudiation mean?
  - what type of non-repudiation protocols do exist?
  - what are the pros and cons of the various approaches?

- useful reading:
  - literature on “fair exchange”
Introduction

- in many applications, it is essential to ensure that participants of a transaction cannot deny having participated in the transaction

- the problem can be traced back to the problem of non-repudiation of message origin and message delivery
  - non-repudiation of message origin
    - sender of the message cannot deny that he sent the message
  - non-repudiation of message delivery (receipt)
    - receiver of a message cannot deny that he received the message

- ingredients of solutions
  - digital signatures, ...
  - protocols that ensure fairness
    - proof of message origin is provided to the receiver only if a proof of message delivery is available to the sender, and vice versa

Fairness

- assume that A wants to send a message m to B

- (strong) fairness:
  at the end of the protocol, either
  - B gets m and a non-repudiation of origin evidence for m, and
  - A gets a non-repudiation of delivery evidence for m
  or none of them get anything useful

- an alternative, more precise definition:
  if both parties are rational, then at the end of the protocol, the following conditions hold
  - if A is honest, then B does not receive anything useful, unless A receives a non-repudiation of delivery evidence for m
  - if B is honest, then A does not receive anything useful, unless B receives m and a non-repudiation of origin for m
More definitions

- **weak fairness:**
  if an honest party does not receive its evidence, while the other party does, then the first party receives a proof of this fact

- **probabilistic fairness:**
  a protocol provides \(\varepsilon\)-fairness, if it guarantees fairness with probability \(\varepsilon\)

- **timeliness:**
  - all honest parties can reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness

- **communication models:**
  - unreliable channel: messages can be lost
  - resilient channel: all messages are eventually delivered (after a finite, but unknown amount of time)
  - reliable (operational) channel: all messages are delivered within a known, constant amount of time (there’s an upper bound on the message delivery delay)

Types of non-repudiation protocols

- **no TTP (Trusted Third Party)**

- **with TTP**
  - on-line TTP
    - the TTP is involved in each run of the protocol
  - off-line TTP
    - the TTP is involved only if something goes wrong (a message is not received due to a communication error or some misbehavior)
    - we may assume that, most of the time, there won’t be any problems, so the protocol can be optimized (in terms of efficiency) for the faultless case (→ also called optimistic protocols)
**A protocol with an on-line TTP**

- **protocol:**
  1. $A \rightarrow \text{TTP} : E_{\text{TTP}}(A, B, m, \text{sig}_{A}(A, B, h(m)))$
  2. $\text{TTP} \rightarrow B : A, B, h(m), \text{sig}_{\text{TTP}}(A, B, h(m))$
  3. $B \rightarrow \text{TTP} : E_{\text{TTP}}(\text{sig}_{B}(A, B, h(m)))$
  4a. $\text{TTP} \rightarrow A : \text{sig}_{B}(A, B, h(m))$
  4b. $\text{TTP} \rightarrow B : m, \text{sig}_{A}(A, B, h(m))$

- **notes:**
  - NRO = $\text{sig}_{A}(A, B, h(m))$, NRR = $\text{sig}_{B}(A, B, h(m))$
  - $E_{\text{TTP}}(\ )$ is used to prevent eavesdropping of $m$ and the evidences
  - TTP is trusted for checking signatures and sending messages 4a and 4b simultaneously
  - fairness is based on this simultaneous transmission of 4a and 4b, but there are problems:
    - if channels are resilient, then it is unclear how long the TTP should wait for B’s response, and thus, how long A should wait for the TTP’s message (timeliness is not guaranteed)
    - the TTP may crash between sending 4a and sending 4b, and leave B in an unfair situation

---

**Fixing the timeliness problem**

- **protocol:**
  1. $A \rightarrow \text{TTP} : E_{\text{TTP}}(A, B, m, T, \text{NRO} = \text{sig}_{A}(A, B, h(m), T))$
  2. $\text{TTP} \rightarrow B : A, B, h(m), T, \text{sig}_{\text{TTP}}(A, B, h(m), T)$
  3. $B \rightarrow \text{TTP} : E_{\text{TTP}}(\text{NRR} = \text{sig}_{B}(A, B, h(m), T))$

  if TTP receives msg 3 before T:
  4. TTP publishes at T: A, B, h(m), T, m, NRO, NRR

  else:
  4’. TTP publishes at T: A, B, h(m), T, “ABORTED”

  5a. after T, A checks for the result of the protocol
  5b. after T, B checks for the result of the protocol

- **notes:**
  - the TTP can publish results by making them available through a server (e.g., through the web)
  - if TTP crashes before step 4, then no result will be available (for some time), but fairness is still preserved
  - in any case, A and B should continue polling the server until they receive some response (their evidences or the abort indication)
  - if channels are resilient, the protocol will end after a finite amount of time
Another variant (Zhou-Gollmann)

protocol:
  \[ C = E_k(m) \text{ where } K \text{ is a random session key} \]
  1. \( A \to B \) : \( A, C, T, NRO_1 = \text{sig}_A(A, B, C, T) \)
  2. \( B \to A \) : \( B, NRR_1 = \text{sig}_B(A, B, C, T) \)
  3. \( A \to \text{TTP} \) : \( E_{\text{TTP}}(A, B, T, K, \text{sig}_A(A, B, T, K)) \)
  4. \( \text{TTP} \) publishes at \( T \) : \( A, B, T, K, NRO_2 = \text{sig}_{\text{TTP}}(A, B, T, K) \)
  5a. after \( T \), \( A \) tries to download \( NRO_2 \)
  5b. after \( T \), \( B \) tries to download \( NRO_2 \)

notes:
- \( NRO = NRO_1 + NRO_2 \)
- \( NRR = NRR_1 + NRO_2 \)
- \( \text{NROR}_2 \) means
  - as part of \( NRO \): \( K \) was sent by \( A \) before \( T \)
  - as part of \( NRR \): \( K \) was made available to \( B \) after \( T \)
- if, in step 5, \( NRO_2 \) is not on the server, then the downloading party can stop the protocol (in order to preserve fairness, the TTP should not ever publish \( NRO_2 \) after \( T \))

A protocol with an off-line TTP

main protocol:
  1. \( A \to B \) : \( A, B, \text{id}, E_k(m), E_{\text{TTP}}(K), NRO_1 = \text{sig}_A(\ldots) \)
  2. \( B \to A \) : \( A, B, \text{id}, NRR = \text{sig}_B(A, B, \text{id}, E_k(m), E_{\text{TTP}}(K)) \)
  3. \( A \to B \) : \( A, B, \text{id}, K, NRO_2 = \text{sig}_A(\ldots) \)
    - if \( B \) timeouts, then call the recovery protocol
  \( NRO = NRO_1 + NRO_2 \)

recovery protocol (only for \( B \)):
  1. \( B \to \text{TTP} \) : \( A, B, \text{id}, E_k(m), E_{\text{TTP}}(K), NRO_1, NRR \)
  2. \( \text{TTP} \to B \) : \( A, B, \text{id}, K, NRO_2' = \text{sig}_{\text{TTP}}(\ldots) \)
  3. \( \text{TTP} \to A \) : \( A, B, \text{id}, NRR \)
  \( NRO' = NRO_1 + NRO_2' \)

notes:
- if \( A \) does not send message 3, then \( B \) can invoke the recovery protocol to re-establish fairness
- \( B \) will then get \( NRO' = NRO \)
- \( B \) may start recovery without sending message 2 (and hence \( NRR \))
- that is why \( B \) must also provide \( NRR \) during recovery, which is then sent to \( A \)
- what if \( A \) sends \( E_{\text{TTP}}(K') \) in message 1?
A timeliness problem again

- A does not know when to stop if message 2 doesn’t arrive
  - if she stops, B may start the recovery protocol and obtain NRO’ (while A will no longer receive NRR)
  - so she should wait, but B may have indeed stopped the protocol, and A will wait forever

- a potential solution
  - an abort protocol that A can call any time to force termination

A protocol with an off-line TTP – revised

- main protocol:
  1. $A \rightarrow B : A, B, id, E_k(m), E_{TTP}(K), NRO1 = \text{sig}_A( A, B, id, E_k(m), E_{TTP}(K) )$
  2. $B \rightarrow A : A, B, id, NRR1 = \text{sig}_B( A, B, id, E_k(m), E_{TTP}(K) )$
     if A timeouts, then call the abort protocol
  3. $A \rightarrow B : A, B, id, K, NRO2 = \text{sig}_A( A, B, id, K )$
     if B timeouts, then call the recovery protocol
  4. $B \rightarrow A : A, B, id, NRR2 = \text{sig}_B( A, B, id, K )$
     if A timeouts, then call the recovery protocol
  
  $NRO = NRO1 + NRO2; \quad NRR = NRR1 + NRR2$

- abort protocol (only for A):
  1. $A \rightarrow TTP : A, B, id, \text{"PLEASE ABORT"}$
     if already aborted or recovered then stop, else aborted = TRUE and ...
  2. $TTP \rightarrow A : A, B, id, \text{"ABORTED"}, \text{sig}_{TTP}( ... )$
  3. $TTP \rightarrow B : A, B, id, \text{"ABORTED"}, \text{sig}_{TTP}( ... )$

- recovery protocol (for X in \{A, B\}):
  1. $X \rightarrow TTP : A, B, id, E_k(m), E_{TTP}(K), NRO1, NRR1$
     if already aborted or recovered then stop, else recovered = TRUE and ...
  2. $TTP \rightarrow A : A, B, id, NRR2' = \text{sig}_{TTP}( ... ); NRR1$
  3. $TTP \rightarrow B : A, B, id, K, NRO2' = \text{sig}_{TTP}( ... )$
Properties

- **fairness:**
  - if B feels something is going wrong, then he can invoke the recovery protocol at any time after receiving message 1 (which is the starting point for B).
  - if A feels something is going wrong, then she can invoke the recovery protocol after receiving message 2.
  - before that, she can cancel the transaction by calling the abort protocol.
  - abort and recovery are mutually exclusive.
  - when A invoked the abort protocol, she shouldn’t continue the main protocol (even if B’s message arrives later).
    - B may misbehave (e.g., doesn’t send message 4), and A cannot call the recovery protocol anymore.
    - an abort evidence is not a proof that the transaction didn’t take place, because the abort protocol can be called after a successful run of the protocol.

- **timeliness**
  - at each point in the protocol both parties can force termination either by calling the recovery protocol or the abort protocol.

- **TTP is not transparent**
  - evidences produced in the protocol when TTP is used are different from those that are produced in the case when no TTP is used.

Protocols with no TTP

- **strong fairness cannot be achieved without a TTP**
  - assume that P is a protocol that
    - does not use a TTP
    - achieves strong fairness
    - and uses minimum number n of messages.
  - assume w.l.o.g. that the last message of P is sent by A to B.
  - before sending this last message A has its evidences, because she does not receive anything else in the protocol.
  - on the other hand, before receiving this last message, B still needs something, otherwise the last message would be useless, and we could have a fair non-repudiation protocol P’ with n-1 messages.
  - therefore, if A does not send the last message, then B will suffer a disadvantage, and hence, P cannot be fair.

- protocols with no TTP try to achieve weaker forms of fairness (e.g., probabilistic fairness).
A protocol with no TTP providing probabilistic fairness

- protocol:
  - $C = E_K(m)$ where $K$ is a random key
  - $1. A \rightarrow B: A, B, id, C, \text{NRO}_0 = \text{sig}_A( A, B, id, C )$
  - $2. B \rightarrow A: A, B, id, \text{NRR}_0 = \text{sig}_B( A, B, id, C )$
  - with prob. $\varepsilon$, $r_1 = K$, and with prob. $1-\varepsilon$, $r_1$ is a random number
  - $3. A \rightarrow B: A, B, id, 1, r_1, \text{NRO}_1 = \text{sig}_A( A, B, id, 1, r_1 )$
  - $4. B \rightarrow A: A, B, id, \text{NRR}_1 = \text{sig}_B( A, B, id, 1, r_1 )$
  - ... with prob. $\varepsilon$, $r_n = K$, and with prob. $1-\varepsilon$, $r_n$ is a random number
  - $2n+1. A \rightarrow B: A, B, id, n, r_n, \text{NRO}_n = \text{sig}_A( A, B, id, n, r_n )$
  - $2n+2. B \rightarrow A: A, B, id, \text{NRR}_n = \text{sig}_B( A, B, id, n, r_n )$

- $\text{NRO} = \text{NRO}_0 + \text{NRO}_n; \text{NRR} = \text{NRR}_0 + \text{NRR}_n$

- important assumption:
  - decryption of $C$ takes longer time than the timeout set by $A$ in each step \( \rightarrow \) if $B$ tries to test $r_i$, then $A$ timeouts and stops the protocol

Non-repudiation protocols

Brief analysis

- fairness for $B$:
  - if $A$ has $\text{NRR}_n$, then $B$ must have $\text{NRO}_n$ (given that $B$ is honest)

- fairness for $A$:
  - in each step of the protocol, $B$ may decide to stop
  - he gets in advantageous situation ($B$ has $\text{NRO}_n$, but $A$ doesn’t have $\text{NRR}_n$) with prob. $\varepsilon$
  - $B$‘s decision is wrong with prob. $1-\varepsilon$, and in this case, fairness is preserved

- timeliness:
  - it is safe to stop for $B$ at any time in the protocol
  - but how long should $A$ wait for $B$’s last message?
    - if $A$ stops prematurely, then she may end up in a disadvantageous state
    - $A$ should wait for $B$’s response, but it may not have been sent by $B$

- overhead problem
  - parameter $\varepsilon$ should be small for better fairness
  - the smaller $\varepsilon$ is, the larger $n$ is \( \rightarrow \) good fairness results in high overhead
Another approach with no TTP – first attempt

- A wants to send message m to B
- \( \text{NRO} = \text{sig}_A(A, B, h(m)), \text{NRR} = \text{sig}_B(A, B, h(m)) \)

protocol:
- A generates a random key \( k_A \) and encrypts m and NRO \( \rightarrow \{m, \text{NRO}\}_{k_A} \)
- A sends \( h(m) \) and \( \{m, \text{NRO}\}_{k_A} \) to B
- B generates a random key \( k_B \) and encrypts NRR \( \rightarrow \{\text{NRR}\}_{k_B} \)
- B sends \( \{\text{NRR}\}_{k_B} \) to A
- A and B exchange \( k_A \) and \( k_B \) bit by bit:
  - in the i-th step A sends \( k_A[i] \) and B sends \( K_B[i] \)
  - at the end, both A and B decrypt the encrypted items and check them

- problem: what if a party sends random bits instead of the real key?
- each party must be able to verify that the other really sends the bits of his/her key!

A useful building block: bit commitment

- a bit commitment protocol ensures that A can commit to a binary value b in such a way that
  - B cannot learn the committed value until A opens the commitment
  - A cannot later change the committed value and claim that she has committed to \( b' \) (instead of b)

- an example based on a collision resistant, one-way hash function \( H \):
  - A wants to commit to a bit b
  - A generates a random number \( r \) (of sufficient length)
  - A computes \( c = H(r \mid b) \)
  - A sends c to B
    - B cannot compute b, because \( H \) is one-way
  - when A wants to open the commitment, she sends \( (r, b) \) to B
  - B verifies that \( H(r \mid b) = c \)
    - in order to cheat, A should be able to find \( r' \) such that \( H(r' \mid b') = H(r \mid b) \)
    - this is not possible, because \( H \) is collision resistant
Second attempt

- A sends to B:
  
  \[ [A, B, h(m), \{m, NRO\}_{k_A}, C_A, \text{Sig}_A(\ldots)] \]

  where \( C_A = ( H(p_1 | k_A[1]), \ldots, H(p_L | k_A[L]) ) \), \( L \) is the bit length of \( k_A \)
  and \( p_i \) are random numbers

- B sends to A:
  
  \[ [B, A, h(m), \{NRR\}_{k_B}, C_B, \text{Sig}_B(\ldots)] \]

  where \( C_B = ( H(q_1 | k_B[1]), \ldots, H(q_L | k_B[L]) ) \), and \( q_i \) are random numbers

- A and B open their commitments one after the other:
  - A sends \( (p_i, k_A[i]) \) and B sends \( (q_i, k_B[i]) \)
  - A and B verify that they received the committed bits

- at the end, both A and B decrypt the encrypted items and check them

Brief analysis

- let us assume that A is honest and B stops after the \( t \)-th step

- A has
  
  \[ [B, A, h(m), \text{enc}, C_B, \text{Sig}_B(\ldots)] \]

  \( (q_1, k_B[1]), \ldots, (q_t, k_B[t]) \)

- if it is infeasible for A to determine the rest of \( k_B \) (\( t \) is too small), then it is
  infeasible for B as well to determine the rest of \( k_A \)

- assume that \( t \) is large enough, and A tries to determine the rest of \( k_B \)
  - she tries to decrypt \( \text{enc} \) with \( k[1..t] | k[t+1..L] \) for all possible values of
    \( k[t+1..L] \)
  - she may succeed
    - B needs almost the same amount of effort to succeed
  - if she doesn’t succeed then she has a proof that B has cheated
    - \( k[1..t] \) are the bits committed by B
    - there’s no \( k[t+1..L] \) such that decrypting \( \text{enc} \) with \( k[1..t] | k[t+1..L] \) results in \( \text{NRR} \)
    - B’s signature proves that B provided false information to A
Some conclusions

- two important requirements for non-repudiation protocols are fairness and timeliness
- there are many subtle details to consider during the design (formal methods?)

- types:
  - with on-line TTP
    - protocols of this kind are conceptually simple, but
    - TTP is a bottleneck and a single point of failure
  - with off-line TTP
    - (full) protocol is complex, but main protocol can be simple
    - less demand on the TTP, efficient in case of no faults
  - with no TTP
    - true fairness cannot be achieved
    - lot of overhead
    - strong assumptions (e.g., equal computing capacity of the parties)

- relation to fair exchange (of general items)
  - can be considered as subclass of fair exchange protocols