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 (reception)
    - receiver of a message cannot deny that he received the message

- ingredients of solutions
  - digital signatures, …
  - fair exchange protocols
The fair exchange problem

- Alice and Bob do not trust each other, they both believe that the other party may try to cheat.
- If Alice has access to itemB but Bob does not have access to itemA, then Bob has a disadvantage, and vice versa.

Fairness

- Informally: an honest (correctly behaving) party shouldn’t suffer any disadvantages.
- More precisely: if both parties are rational, then at the end of the protocol, the following conditions hold:
  - if A is honest, then B receives itemA, only if A receives itemB
  - if B is honest, then A receives itemB, only if B receives itemA
More definitions

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

- probabilistic fairness:
  a protocol provides $\varepsilon$-fairness, if it guarantees fairness with probability $1-\varepsilon$

- 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 message 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)

Instances

- non-repudiation protocols
  - exchange of message and its non-repudiation of origin (NRO) token for non-repudiation of receipt (NRR) token

- certified electronic mail
  - exchange of mail for acknowledgement of receipt

- electronic contract signing
  - exchange of signatures on the contract text

- purchase of network delivered services
  - exchange of electronic payment for services

- mutual disclosure of identities
  - exchange of identity information
Types of fair exchange protocols

- **no TTP (Trusted Third Party)**
  - **main idea:**
    - break the exchange up into small steps
    - if one party stops in the middle of the exchange, both parties need approximately the same amount of effort to finish the exchange (compute the other’s item)
    - doesn’t achieve true fairness, usually needs many messages to exchange → impractical in many applications, but theoretically interesting

- **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)

No TTP – a naïve protocol

- **protocol:**
  - A has item \( A \) and desc\( B \), B has item\( B \) and desc\( A \)
  - A generates a random key \( k_A \) and encrypts item\( A \) → \{item\( A \)\}_{k_A}
  - A sends \{item\( A \)\}_{k_A} to B
  - B generates a random key \( k_B \) and encrypts item\( B \) → \{item\( B \)\}_{k_B}
  - B sends \{item\( B \)\}_{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 against the descriptions

- **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!
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 | 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 | b) = c
      - in order to cheat, A should be able to find r' such that H(r' | b') = H(r | b)
      - this is not possible, because H is collision resistant

No TTP – second attempt

- A has itemA and descB, B has itemB and descA
- A sends to B:
  \[[A, B, desc_A, desc_B, \{item_A\}_{kA}, C_A, Sig_A(\ldots)]\]
  where \(C_A = (H(p_1 | kA[1]), \ldots, H(p_L | kA[L]))\), L is the bit length of \(kA\), and \(p_i\) are random numbers
- B sends to A:
  \[[B, A, desc_B, desc_A, \{item_B\}_{kB}, C_B, Sig_B(\ldots)]\]
  where \(C_B = (H(q_1 | kB[1]), \ldots, H(q_L | kB[L]))\), and \(q_i\) are random numbers
- A and B open their commitments one after the other:
  - A sends \((p_i, kA[i])\) and B sends \((q_i, kB[i])\)
  - A and B verify that they received the committed bits
  - at the end, both A and B decrypt the encrypted items and check them against the descriptions
Brief analysis

- Let us assume that A is honest and B stops after the t-th step.
- A has \( [B, A, \text{desc}_B, \text{desc}_A, \text{encitem}, \text{C}_B, \text{Sig}_B(\ldots)] \) and \( (q_1, k[1]), \ldots, (q_t, k[t]) \).
- 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 \( k_A \).
- Let us assume that it is feasible for A to determine the rest of \( k_B \):
  - She tries to decrypt \( \text{encitem} \) with \( k[1..t] \mid k[t+1..L] \) for all possible values of \( k[t+1..L] \).
  - She may succeed and end up with an item that matches \( \text{desc}_B \).
    - B needs almost the same amount of effort to succeed.
  - If she doesn’t succeed then she has a proof that B has cheated (weak fairness).
    - \( k[1..t] \) are the bits committed by B.
    - There’s no \( k[t+1..L] \) such that decrypting \( \text{encitem} \) with \( k[1..t] \mid k[t+1..L] \) results in an item that matches \( \text{desc}_B \).
    - B’s signature proves that B provided false information to A.

Other properties

- A lot of messages need to be exchanged.
- Interactive, both parties should be on-line simultaneously.
  - Not appropriate for some applications, e.g., e-mail.
- Both parties should have comparable computing power.
  - Not applicable in some cases, e.g., when one of the parties is a mobile phone and the other is a server.
- Does not provide strong fairness (only weak fairness).
- In a strict sense, doesn’t actually provide fairness.
An impossibility result

- true fairness cannot be achieved without a TTP (Even and Yacobi [1980]):

A protocol providing probabilistic fairness

- objective:
  - exchange of a message $m$ and its Non-Repudiation of Origin (NRO) token with a Non-Repudiation of Receipt (NRR) token

- protocol:
  
  $$C = E_K(m)$$ where $K$ is a random key
  1. $A \rightarrow B : A, B, id, C, NRO_0 = \text{sig}_A(A, B, id, C)$
  2. $B \rightarrow A : A, B, id, NRR_0 = \text{sig}_B(A, B, id, C)$
    with prob. $\epsilon$, $r_1 = K$, and with prob. $1-\epsilon$, $r_1$ is a random number
  3. $A \rightarrow B : A, B, id, 1, r_1, NRO_1 = \text{sig}_A(A, B, id, 1, r_1)$
  4. $B \rightarrow A : A, B, id, NRR_1 = \text{sig}_B(A, B, id, 1, r_1)$
    ...
  2n+1. $A \rightarrow B : A, B, id, n, r_n, NRO_n = \text{sig}_A(A, B, id, n, r_n)$
  2n+2. $B \rightarrow A : A, B, id, NRR_n = \text{sig}_B(A, B, id, n, r_n)$

- important assumption:
  - decryption of $C$ takes longer time than the timeout set by $A$ in each step → if $B$ tries to test $r_n$ then $A$ timeouts and stops the protocol
Brief analysis

- fairness for B:
  - if A has NRR_n, then B must have NRO_n (given that B is honest)

- fairness for A:
  - in each step of the protocol, B may decide to stop
  - he gets in an advantageous situation (B has NRO_n, but A doesn’t have NRR_n) with prob. \( \varepsilon \)
  - fairness is preserved with probability \( 1-\varepsilon \)

- timeliness problem:
  - 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 waiting prematurely, then she may end up in a disadvantageous state
    - A should wait for B’s response, but B may not have sent it

- 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

Rational exchange

informal definition

- a misbehaving party cannot gain any advantages
  \( \Rightarrow \) misbehavior is uninteresting and should happen only rarely

- few rational exchange protocols proposed in the literature
- they seem to provide weaker guarantees than fair exchange protocols, but …
- they are usually less complex than fair exchange protocols
  \( \Rightarrow \) trade off between complexity and fairness
  \( \Rightarrow \) interesting solutions to the exchange problem
An example: a rational payment protocol

\[ U \rightarrow V : m_1 = U, V, tid, val, h(rnd), \text{Sig}_U(U, V, tid, val, h(rnd)) \]
\[ V \rightarrow U : m_2 = srv \]
\[ U \rightarrow V : m_3 = \text{rnd} \]

if \( V \) received \( m_1 \) and \( m_3 \):
\[ B : \text{charges } U \text{ with val} \]
\[ \text{credits } V \text{ with val} \]

if \( V \) received only \( m_1 \):
\[ V \rightarrow B : m_4 = m_1, \text{Sig}_V(m_1, m_3) \]

brief informal analysis
- no fairness, but …
- none of the parties gain any financial advantages by cheating (rationality)
- needs a TTP (the bank), but …
- the bank is needed anyway to maintain accounts
- the performs the same operations as in any credit based payment system

Fair exchange with an on-line TTP

- protocol (channels are authentic):
  1. \( A \rightarrow \text{TTP} : A, B, \text{desc}_A, \text{desc}_B, E_{\text{TTP}}(\text{item}_A) \)
  2. \( \text{TTP} \rightarrow B : A, B, \text{desc}_A, \text{desc}_B \)
  3. \( B \rightarrow \text{TTP} : A, B, \text{desc}_A, \text{desc}_B, E_{\text{TTP}}(\text{item}_B) \)
  4a. \( \text{TTP} \rightarrow A : \text{item}_B \)
  4b. \( \text{TTP} \rightarrow B : \text{item}_A \)

- notes:
  - \( E_{\text{TTP}}(\text{item}_A \text{item}_B) \) is used to prevent eavesdropping (by A or B)
  - TTP is trusted for checking if the items match their descriptions 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 → TTP : A, B, descA, descB, E_TTP( itemA ), T
2. TTP → B : A, B, descA, descB, T
3. B → TTP : A, B, descA, descB, E_TTP( itemB )
4. if TTP receives msg3 before T:
   4.1. TTP publishes at T: A, B, itemA, itemB
else:
   4.2. TTP publishes at T: A, B, “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 expected items or the abort indication)
• if channels are resilient, the protocol will end after a finite amount of time

Fair exchange with a semi-trusted on-line TTP

• Alice and Bob wants to exchange itemA and itemB with the help of TTP
• they don’t want TTP to learn the value of their items
• idea:
  • split the item into two pieces using a 2-out-of-2 secret sharing scheme
  • one piece is exchanged directly with the other party, while the other is exchanged through the TTP
• TTP can still help to achieve fairness
• TTP learns only one share of each item out of the two needed to reconstruct the item

main challenge:
• parties need to be able to verify that they received correct shares and not garbage
Building block

- let G be a finite group
- let f: G → G be a collision resistant one-way function
- let F: G × G → G be an efficiently computable function
- f and F satisfy the following property: F(x, f(y)) = f(xy)
- example:
  - G = \mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}, where p is a large prime, and g has order q < p-1
  - f(x) = g^x \mod p
  - F(x, y) = y^x \mod p
  - F(x, f(y)) = (g^x)^y \mod p = g^{xy} \mod p = f(xy)

The protocol

- A and B wants to exchange KA and KB
- A has KA and f(KB), and B has KB and f(KA)
- A generates a random number x and sends it to B
- B generates a random number y and sends it to A
- A sends the following message to TTP: f(KA), f(KB), \text{KA}x^{-1}, f(y)
- B sends the following message to TTP: f(KB), f(KA), \text{KB}y^{-1}, f(x)
- TTP receives \text{a}_A, \text{b}_A, \text{c}_A, \text{d}_A from A and \text{a}_B, \text{b}_B, \text{c}_B, \text{d}_B from B
- TTP verifies that
  - \text{a}_A = \text{b}_B = F(c_A, d_B) /* F(\text{KA}x^{-1}, f(x)) = f(\text{KA}) */
  - \text{a}_B = \text{b}_A = F(c_B, d_A) /* F(\text{KB}y^{-1}, f(y)) = f(\text{KB}) */
- TTP sends \text{c}_B (= \text{KB}y^{-1}) to A and \text{c}_A (= \text{KA}x^{-1}) to B
- A computes \text{c}_B y = \text{KB}y^{-1}y = \text{KB}
- B computes \text{c}_A x = \text{KA}x^{-1}x = \text{KA}
Properties

- if all three parties are honest
  - A learns $K_B$ and B learns $K_A$

- if A and TTP are honest
  - B learns nothing useful unless TTP sends $c_A$ to B
  - but then TTP sends $c_B$ to A such that $f(c_B) = f(K_B)$
  - this means that either $c_B = K_B^{-1}$ and A can compute $c_B = K_B$, or B has found a collision of $f$ at $f(K_B)$
  - the latter is not possible because $f$ is collision resistant

- if B and TTP are honest
  - same as in the previous point due to symmetry

- if A and B are honest
  - TTP learns nothing useful (i.e., it can simulate its view from $f(K_A)$ and $f(K_B)$ alone)
  - the exchange may fail, but A and B can repeat it with another TTP

Fair exchange with an off-line TTP

- motivations:
  - on-line TTP has many disadvantages
    - cost: TTP must be available all the time, and it should be highly reliable (can be very expensive!)
    - congestion: the TTP is a performance bottleneck in the network
    - liability: 100% reliability and availability is impossible → TTP should have some insurance to cover the costs of potential damages caused by its failure
  - optimistic view of the world
    - most of the parties are honest (most of the time) → cheating is rare
    - most of the time, computers and networks work properly → crashes and communication failures are rare

- main idea:
  - use the TTP only in the case when something goes wrong (to re-establish fairness)
  - optimize the protocol for the faultless case (when the TTP is not used) as this is expected to occur most of the times
A possible solution

- main protocol:
  1. A → B : A, B, descA, descB, E_{TTP}(itemA), sigA(…)
  2. B → A : A, B, descA, descB, E_{TTP}(itemB), sigB(…)
  3. A → B : itemA
  4. B → A : itemB

- recovery protocol for A (there’s a similar one for B too):
  1. A → TTP : (A, B, descA, descB, E_{TTP}(itemB), sigB(…)), E_{TTP}(itemA)
  2. if not yet RESOLVED or ABORTED, then TTP verifies signature, decrypts E_{TTP}(itemB) and E_{TTP}(itemA), verifies if itemB matches descB and if itemA matches descA
  3. if all verifications are successful, then TTP makes available itemA and itemB for A and B (RESOLVED), otherwise TTP publishes “ABORT” (ABORTED)
  4. A checks for the result

- abort protocol (only!) for A:
  2. if not yet RESOLVED or ABORTED, then TTP publishes “ABORT” (ABORTED)
  3. A checks for the result

Analysis

- if both parties are honest and there’s no communication failure, then they both get what they want
- if A is honest
  - if A does not get msg4, then she can run the resolve protocol → A gains access to itemB or the exchange is aborted in which case B will not have access to itemA
  - if A does not get msg2, then she can run the abort protocol → the exchange is aborted and B will not have access to itemA or B has already called the resolved protocol in which case A gains access to itemB
- if B is honest
  - if B does not get msg3, then he can run the resolve protocol …
  - if B does not get msg1
    - B does not even know about A’s exchange attempt
    - A does not have msg2, therefore, she can only call the abort protocol → no one gets anything useful
Analysis (cont’d)

- timeliness
  - any party at any time can call either resolve or abort
  - a party that called resolve or abort will check for the result (don’t quit until he/she gets it)
  - resilient channel assumption → party will eventually reach the TTP and doesn’t have to wait forever

- it is crucial that only A can call abort
  - otherwise B could receive msg3 (item$_A$) and then prevent A from gaining access to item$_B$ by calling abort

Verifiable escrow

- motivation
  - it would be nice for any party to be able to verify that the encrypted item received from the other party indeed contains an item that matches its known description

- the primitive we need is “verifiable escrow”
  - a designated party (here the TTP) can decrypt it
  - anybody can verify that the promised item can indeed be efficiently computed from the content of the escrow
Formal definition - preliminaries

- Let \( F : G_1 \rightarrow G_2 \) be a surjective homomorphism, where \( G_1 \) and \( G_2 \) are finite groups and \(|G_1| \geq |G_2|\)
  - Homomorphic property: \( F(x + y) = F(x) \cdot F(y) \), where \(+\) and \(\cdot\) are the group operations
- Let \( p \) in \( G_2 \) be a public group element, and let \( s \) in \( F^{-1}(p) \)
  (which is a subset of \( G_1 \)) be a secret
- We want to escrow \( s \) under the public key of a third party such that
  - It can be publicly verified that when decrypted, a pre-image of \( p \) is obtained, and
  - The verification procedure does not reveal any information that makes it easier to compute an \( s \) in \( F^{-1}(p) \)
- We may also want to bind a condition \( \kappa \) to the encryption that can be used by the third party to determine if the decryption is authorized

Formal definition

- Algorithms:
  - Key generation algorithm
    - Any key pair generation \( \rightarrow \) PK, SK
  - Prover and verifier algorithms
    - The prover and the verifier run an interactive protocol
    - The prover’s inputs are PK, \( p, \kappa, s \)
    - The verifier’s inputs are PK, \( p, \kappa \)
    - At the end of the protocol the verifier outputs \( \alpha \) or “reject”
  - Decryption algorithm
    - \( D(SK, \kappa, \alpha) = s' \) (if condition \( \kappa \) is satisfied)
- Properties
  - Completeness: if \( F(s) = p \), than the verifier accepts and outputs \( \alpha \)
  - Soundness: the probability that the verifier outputs \( \alpha \) and \( F(D(SK, \kappa, \alpha)) \neq p \) is negligible
  - Zero knowledge: no information about \( s \) is leaked to the verifier
**Prover – Verifier protocol**

- P chooses N random elements \( r_1, r_2, \ldots, r_N \) from \( G_1 \)
  - P computes
    - \( c_i = E(PK, \kappa | r_i) \) for all \( i = 1, \ldots, N \)
    - \( h_i = (c_i, F(r_i)) \) for all \( i = 1, \ldots, N \)
    - \( H = \text{hash}(h_1| \ldots| h_N) \)
  - P sends \( H \) to V

- V sends N random bits \( b_1, b_2, \ldots, b_N \) to P

- P computes \( v_i \) as follows:
  - if \( b_i = 0 \), then \( v_i = r_i \)
  - if \( b_i = 1 \), then \( v_i = (c_i, r_i + s) \)
  - P sends \( v_1, v_2, \ldots, v_N \) to V

- V computes \( h'_i \) as follows:
  - if \( b_i = 0 \), then \( h'_i = (E(PK, \kappa | r_i), F(r_i)) = (c_i, F(r_i)) = h_i \)
  - if \( b_i = 1 \), then \( h'_i = (c_i, F(r_i+s) - p) = (c_i, F(r_i) + F(s) - p) = (c_i, F(r_i)) = h_i \)

- V computes \( H' = \text{hash}(h'_1| \ldots| h'_N) \)
  - if \( H' \neq H \) then reject
  - otherwise output \( \alpha = \{ (ci, ri + s) : bi = 1 \} \)

**Decryption algorithm**

- \( \alpha = \{ (c_i, r_i + s) : b_i = 1 \} \) where \( c_i = E(PK, \kappa | r_i) \)

- \( D(SK, \kappa, \alpha) = \)
  - decrypt \( E(PK, \kappa | r_i) \) with \( SK \rightarrow \kappa | r_i \)
  - check condition \( \kappa \)
  - check if \( F((r_i+s)-r_i) = p \)
  - if OK, then output \((r_i+s)-r_i = s\)
Analysis

- standard cut-and-choose technique
- assume that P can predict \( b_i \)
  - if \( b_i = 0 \), then P computes \( h_i = (c_i, F(r_i)) \) (as before)
  - if \( b_i = 1 \), then P computes \( h_i = (c_i, F(r_i) - p) \) (instead of \( (c_i, F(r_i)) \))
- P computes \( H = h(h_1 \| \ldots \| h_N) \)
- later, when V challenges P with the bit \( b_i \), P sends
  - if \( b_i = 0 \), then \( v_i = r_i \) (as before)
  - if \( b_i = 1 \), then \( v_i = (c_i, r_i) \) (instead of \( (c_i, r_i + s) \))
- V computes \( h_i' \) as before:
  - if \( b_i = 0 \), then \( h_i' = (E(PK, \kappa | r_i), F(r_i)) = (c_i, F(r_i)) = h_i \)
  - if \( b_i = 1 \), then \( h_i' = (c_i, F(r_i) - p) = h_i \)
- V computes \( H' = H \) and accepts, yet P sent nothing that depends on \( s \)
- and indeed, decryption yields \( F(r_i) - p - r_i \neq s \)
- however, to succeed, P has to predict all bits \( b_i \), and therefore, his success probability is \( 2^{-N} \)

Implementation example

- \( F(x) = x^e \mod m \)
  - as usual, let us denote the inverse of \( e \mod \phi(m) \) by \( d \)
  - in this case, \( G_1 = G_2 = \{ \mod m \} \) with multiplication \( \mod m \) as the group operation
  - homomorphic property: \( (x^e \mod m)(y^e \mod m) = (xy)^e \mod m \)
- the secret value \( s \) is A’s RSA signature on a message \( \text{msg} \) (items to be exchanged are often signatures anyway)
  - \( s = (h(\text{msg}))^{d_A} \mod m_A \)
- the corresponding public value is \( p = F(s) \)
Implementation example (cont’d)

- protocol prelude:
  - A sends to B: cert(e_A, m_A), msg, F = (e, m), p = F(s)
  - B computes:
    - V = F(h(msg)) and W = p^e mod m_A
    - if V = W, then B is convinced that the pre-image of p is a valid signature of A on message msg
  - proof:
    - V = F(h(msg)) = (h(msg))^e mod m
    - W = p^e mod m_A
    = (s^e mod m)^e mod m_A
    = (s^e mod m)^e mod m_A
    = (h(msg))^e mod m_A
    = (h(msg))^e mod m_A
    = (h(msg))^e mod m
- then A can run the verifiable escrow protocol with B, with A as prover, B as verifier, and the off-line TTP as the designated third party that can decrypt A’s escrow

Some conclusions

- types of fair exchange protocols:
  - 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)
- besides fairness, timeliness is also an important property
- there are many subtle details to consider during the design of a fair exchange protocol (formal methods?)
- useful reading: