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 item $A$ but Bob does not have access to item $B$, 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 item $A$, only if $A$ receives item $B$.
    - If $B$ is honest, then $A$ receives item $B$, only if $B$ receives item $A$. 
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 item_A and desc_B, B has item_B and desc_A
- A sends to B:
  \[
  [A, B, \text{desc}_A, \text{desc}_B, \{\text{item}_A\}_{k_A}, C_A, \text{Sig}_A(\ldots)]
  \]
  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:
  \[
  [B, A, \text{desc}_B, \text{desc}_A, \{\text{item}_B\}_{k_B}, C_B, \text{Sig}_B(\ldots)]
  \]
  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 against the descriptions
Brief analysis

- let us assume that A is honest and B stops after the t-th step
- A has \([B, A, \text{desc}_B, \text{desc}_A, \text{encitem}, C_B, \text{Sig}_B(\ldots)]\) 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

- 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]):

![Diagram showing a protocol between Alice and Bob](image)

- Alice doesn't have item A yet, since otherwise the last message is unnecessary.
- Bob must have item A since he doesn’t receive anymore messages in the protocol.
- If Bob stops before sending the last message, then Alice suffers a disadvantage.

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 → B : A, B, id, C, NRO_0 = sig_A(A, B, id, C).
  - With prob. ε, r_0 = K, and with prob. 1-ε, r_0 is a random number.
  - 3. A → B : A, B, id, 1, r_1, NRO_1 = sig_A(A, B, id, 1, r_1).
  - 4. B → A : A, B, id, NRR_1 = sig_B(A, B, id, 1, r_1).
  - With prob. ε, r_0 = K, and with prob. 1-ε, r_0 is a random number.
  - 2n+1. A → B : A, B, id, n, r_n, NRO_n = sig_A(A, B, id, n, r_n).

- A stops when K is sent or if she does not receive a response from B within some timeout time.
- B stops when he does not receive the next message from A within some timeout time.
  - NRO = NRO_0 + NRO_n; NRR = NRR_0 + NRR_n.

- Important assumption:
  - Encryption 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
- 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

\[
U \rightarrow V: \ m_1 = U, V, tid, val, h(rnd), \ Sig_U(U, V, tid, val, h(rnd)) \\
V \rightarrow U: \ m_2 = srv \\
U \rightarrow V: \ m_3 = rnd
\]

if \( V \) received \( m_1 \) and \( m_3 \):

\[
V \rightarrow B: \ m_4 = m_3, m_5, \ Sig_V(m_5, m_3) \\
B: \ \text{charges} \ U \ \text{with} \ val
\]

if \( V \) received only \( m_1 \):

\[
V \rightarrow B: \ m'_4 = m_3, \ Sig_V(m_1) \\
B: \ \text{charges} \ U \ \text{with} \ val
\]

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
- it performs the same operations as in any credit based payment system

Fair exchange with an on-line TTP

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

- notes:
  - \( E_{TTP}(\ ) \) 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 \textit{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. $A \rightarrow$ TTP : $A$, $B$, $\text{desc}_A$, $\text{desc}_B$, $E_{\text{TTP}}(\text{item}_A)$, $T$
  2. TTP $\rightarrow$ $B$ : $A$, $B$, $\text{desc}_A$, $\text{desc}_B$, $T$
  3. $B \rightarrow$ TTP : $A$, $B$, $\text{desc}_A$, $\text{desc}_B$, $E_{\text{TTP}}(\text{item}_B)$

  if TTP receives msg3 before T:
  4. TTP publishes at T: $A$, $B$, $\text{item}_A$, $\text{item}_B$
  else:
  4'. 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 item$_A$ and item$_B$ with the help of the TTP
- they don’t want the 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 $f: G \rightarrow G$ be a collision resistant one-way function
- let $F: G \times G \rightarrow G$ 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 $q < p-1$
  - $f(x) = g^x \mod p$
  - $F(x, y) = y^x \mod p$
  - $F(x, f(y)) = f(y^x \mod p) = f(xy)$

The protocol

- A and B wants to exchange $K_A$ and $K_B$
- A has $K_A$ and $f(K_B)$, and B has $K_B$ and $f(K_A)$
- 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: $f(K_A), f(K_B), K_A^{-1}, f(y)$
- B sends the following message to TTP: $f(K_B), f(K_A), K_B^{-1}, f(x)$
- TTP receives $a_A, b_A, c_A, d_A$ from A and $a_B, b_B, c_B, d_B$ from B
- TTP verifies that
  - $a_A = b_B = F(c_A, d_B)$ /* $F(K_A^{-1}, f(x)) = f(K_A)$ */
  - $a_B = b_A = F(c_B, d_A)$ /* $F(K_B^{-1}, f(y)) = f(K_B)$ */
- TTP sends $c_B (= K_B^{-1})$ to A and $c_A (= K_A^{-1})$ to B
- A computes $c_B y = K_B^{-1} y = K_B$
- B computes $c_A x = K_A x^{-1} = K_A$
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 $f(c_B y) = f(K_B)$
    - this means that either $c_B = K_B y^{-1}$ and A can compute $c_B y = K_B$, 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 (channels are authentic):
  1. $A \rightarrow B : A, B, \text{desc}_A, \text{desc}_B, E_{\text{TTP}}(\text{item}_A), \text{sig}_A(\ldots)$
  2. $B \rightarrow A : A, B, \text{desc}_A, \text{desc}_B, E_{\text{TTP}}(\text{item}_B), \text{sig}_B(\ldots)$
  3. $A \rightarrow B : \text{item}_A$
  4. $B \rightarrow A : \text{item}_B$

- recovery protocol for $A$ (there’s a similar one for $B$ too):
  1. $A \rightarrow \text{TTP} : (A, B, \text{desc}_A, \text{desc}_B, E_{\text{TTP}}(\text{item}_B), \text{sig}_B(\ldots)), E_{\text{TTP}}(\text{item}_A)$
  2. if not yet RESOLVED or ABORTED, then TTP verifies signature, decrypts $E_{\text{TTP}}(\text{item}_B)$ and $E_{\text{TTP}}(\text{item}_A)$, verifies if $\text{item}_B$ matches $\text{desc}_B$ and if $\text{item}_A$ matches $\text{desc}_A$
  3. if all verifications are successful, then TTP makes available $\text{item}_A$ and $\text{item}_B$ for $A$ and $B$ (RESOLVED); otherwise TTP publishes "ABORT" (ABORTED)
  4. $A$ checks for the result

- abort protocol (only!) for $A$:
  1. $A \rightarrow \text{TTP} : A, B, \text{desc}_A, \text{desc}_B, \text{"abort"}$
  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 $\rightarrow A$ gains access to $\text{item}_B$ or the exchange is aborted in which case $B$ will not have access to $\text{item}_A$
  - if $A$ does not get msg2, then she can run the abort protocol $\rightarrow$ the exchange is aborted and $B$ will not have access to $\text{item}_A$ or $B$ has already called the resolved protocol in which case $A$ gains access to $\text{item}_B$
- 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 $\rightarrow$ 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: $F(x + y) = F(x) \# F(y)$, where + and $\#$ 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 $F(s) = p$, then 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 $r_1, r_2, ..., r_N$ from $G_1$
  - P computes:
    - $c_i = E(PK, \kappa | r_i)$ for all $i = 1, ..., N$
    - $h_i = (c_i, F(r_i))$ for all $i = 1, ..., N$
    - $H = \text{hash}(h_1 | ... | h_N)$
  - P sends $H$ to $V$

- V sends $N$ random bits $b_1, b_2, ..., b_N$ 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 $v_1, v_2, ..., v_N$ 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 $H' = \text{hash}(h_1' | ... | h_N')$
  - if $H' \neq H$ then reject
  - otherwise output $\alpha = \{(c_i, r_i + s) : b_i = 1\}$

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 $H = h(h_1 | ... | h_n)$
- 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 $h_i' = (E(PK, κ|r_i), F(r_i)) = (c_i, F(r_i)) = h_i$
  - 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 $msg$ (items to be exchanged are often signatures anyway)
  - $s = (h(msg))^{d_A} \mod m_A$
- the corresponding public value is $p = F(s)$
Implementation example (cont’d)

- protocol prelude:
  - A sends to B: cert(e\textsubscript{A}, m\textsubscript{A}), msg, F = (e, m), p = F(s)
  - B computes:
    - V = F(h(msg)) and W = p\textsuperscript{\textsubscript{A}} mod m\textsubscript{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))\textsuperscript{e} mod m
    - W = p\textsuperscript{\textsubscript{A}} mod m\textsubscript{A}
    - (s\textsuperscript{e} mod m)\textsuperscript{A} mod m\textsubscript{A}
    - (s\textsuperscript{A} mod m)\textsuperscript{e} mod m\textsubscript{A}
    - ((h(msg))\textsuperscript{d\textsubscript{A} x e} mod m) mod m\textsubscript{A}
    - ((h(msg))\textsuperscript{d\textsubscript{A} \textsuperscript{A} \textsuperscript{e} mod m} mod m\textsubscript{A}
    - (h(msg))\textsuperscript{\textsubscript{A}} 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: