

Contract Signing Protocols (CSP) *without* a trusted third party (TTP):

A contract signing protocol without a TTP, is protocol used for fair exchange that adheres to the **previously mentioned characteristics of a non-repudiation, fair, and secure exchange** protocol. Such CSPs use no TTP to mediate and resolve any execution error (e.g. cheating) or network failure. These kind of protocols try to achieve their aims using other concepts which tend to be less certain than those with TTP, in a way that at the end of a protocol run, a party can say that the contract is “most probably” valid . The outcome is probabilistic. For instance, the resulting validity of the contract is expressed in a probabilistic ratio.

Eliminating the use of a TTP has two main non-trivial reasons **[no-TTP-1]** . In CSPs with a TTP, the third trusted party acts like a bottleneck for the connection, specially if there is many contracts signed under its supervision or involvement to some extent. That is why we can find many alternatives to a CSP with TTP, as previously illustrated (in section...), some protocols uses inline, where the TTP involves in the execution at the transaction level, others are online (at the session level), and some others uses optimistic TTP, where it only interferes in the protocol run in case of any detected cheating or network failure. All these alternatives main goal is to reduce and control TTP involvement, and thus better efficiency can be attained. More than that, a TTP can rarely be fully trusted, and this a highly desirable property, since its difficult to assess risks introduced by using a TTP[no-TTP-5], which might conflict with some consumers or organizations trust requirements. As mentioned earlier, eliminating the TTP comes at the expense of accepting a probabilistic contract. On the other side, it has been showed by Yacobi and Even [no-TTP-2], there is no non-deterministic protocol exists, where the participation of the TTP is not required. This is true because there is a stage in protocol execution where one party will have his counter party signature without committing himself to the contract. So if this party stops the execution the principle of fairness (no party should commit unless the other is committed)is violated.

If a TTP is used, as a center of cancelation [no-TTP-3], this scenario can easily be solved by simply canceling the contract if it reached such state.

Speaking about fair exchange protocols, its curiously noted that contract signing protocols without a TTP appeared far after those with. This note is logical to some extent, since early trials where

implemented in private business environments, so a TTP would be more trusted and won't be a bottle neck. While in beginning of the 90's [noTTP-4], where the internet was playing an important role in business and commercial world. More open and complex environments were created and thus such transactions needed to be made simpler and more efficient.

History timeline

Introduce like without TTP CSPs appeared after CSPs with TTP

Even (1982): [no-TTP-3]

In 1983, Shimon Even published a paper describing a contract signing protocol that relies on the use of randomization. Such protocols rely on usually on the infeasibility of some number operations, while this protocol rely on public key cryptosystem (PKCS) which is widely believed to be safe.

Even's protocol, as any other protocol assumes certain assumptions. The contract to be signed using this protocol is believed to involve exchanging two equivalent items and is (the contract) of interest to both signing parties. Also, the cost of computation is assumed to be the same for both parties. The availability of a secure conventional cryptosystem is assumed also, this existence is guaranteed by the assumption of a secure PKCS. In a secure PKCS, let E_p be the encryption algorithm and D_p be the decryption algorithm of a party P. E_p is publicly known, while D_p is a secret and known only to P, for a given message m,

$$E_p(D_p(m)) = m$$

Other symmetric cryptosystem can also be used if their security is trusted, e.g. DES. Let the encryption algorithm be F using a key K, while the decryption algorithm is F^{-1} .

Hence,

$$F^{-1}_k(F_k(m)) = m$$

protocol concept:

This protocol uses two concepts, Rabin's signature concept [no-TTP-6], and Merkle's concept of a puzzle [no-TTP-7]. The puzzle is a message pair $(m, F_k(m))$, where the required is to find k . The key, k , must be unique to some very high probability, this can be done by choosing a short key relative to m . The puzzle is solved by exhaustion, this can be by hinting some bits of the key.

Protocol Scheme: Assume the two parties will A and B, with A being the initiator.

1st Step – A prepares $2N$ puzzles,

$F_{k_1}(S), F_{k_2}(S), \dots, F_{k_{2N}}(S)$

where N is a known number in the network (e.g. $N=50$), and $2N$ encoded signed copies of a contract C

$F_{k_1}(D_A(C,1)), F_{k_2}(D_A(C,2)), \dots, F_{k_{2N}}(D_A(C,2N))$

where $(C,i) - 1 \leq i \leq 2N$ - is the i -th copy of C , and formed by appending the bit string representing i to C . Both the $2N$ puzzles and signed copies of C are sent then to B. Then B does the same operation and send his generated puzzles and signed copies to A.

2nd Step – A chooses any $i, 1 \leq i \leq 2N$, and asks B to reveal his k_i , B does so, and A can verify if the received key is right by decrypting the i -th puzzle of B, and then try after that decrypting B's signature of the i -th copy. Then B acts similarly. This step is repeated $N+1$ times, because its conventionally made that B is formally committed to A on contract c , if B has received any $N+1$ of $2N$ signatures sent by A to him.

The $\{ F_{k_i}(S), F_{k_i}(D_A(C,i)) \}$ is called the i -th unit of a message sent from A to B in step 1. A unit received by B will by default considered "bad", if

- The puzzle key size is different from that agreed upon before; i.e. A is making the puzzle harder, thus cheating.
- OR, k_i is not the solution of A's i -th puzzle.

OR, the signature is not verified upon decoding, i.e. $E_A(F_{k_i}^{-1}(M)) \neq (c,1)$, where M_i is the i -th part of the encoded signature sent by A in step (1). Once a bad unit is detected, B will terminate the protocol execution on the assumption that A is cheating.

Protocol Properties:

In a correct and honest execution of this protocol, both parties will have each other signatures on the contract, and both are acknowledged with that. While if the protocol is stopped prematurely

with no reason (cheating, or other problem), the uncooperative party will have a slight advantage on the other in computing the other party signature, this advantage can be kept very small fraction of V (V is the contract value), by choosing an enough large N (for instance, for $N=50$, then this advantage can be expressed as 4% of V). If a party tries sending 'm' bad units purposely, then the probability that he will be exposed during the first k exchanges is:

$$1 - P; \text{ where } P = \frac{2N - m}{2N} \cdot \frac{K}{K}$$

Shimon Even, Oded Golreich, and Abraham Lempel: (1985)

In their paper [no-TTP-8], the authors present a randomized protocol for contract signing that is based on 1-out-of-2 oblivious transfer. 1-out-of-2 oblivious transfer allows one party to transfer exactly only one secret out of two recognizable ones to his counterpart. The receiver receives the secret (whether the first or the second) with a probability half, and the sender doesn't know which secret is received by his counterpart.

According to Even *et al.* a contract signing protocol should have the following properties satisfied:

- i. Viability: After a complete and error-free execution of the protocol, both parties must have each other's signatures.
- ii. Concurrency: No party can obtain the other party signature without yielding his own signature.

--- there-exist an assumption of equal computational power, which might not be suitable for trustCoM, so its better (i think) to include it?? --

Markowitch and Roggeman (1999):

- Goal:

The goal of this protocol is to design a probabilistic non-repudiation protocol. That is accepting a probabilistic fairness at the benefit of avoiding a third trusted party involvement. Markowitch and Roggeman protocol is an iterative one, in a sense that, no party can get any benefit (gain more privileges) except till the last iteration.

- Definitions:

In this protocol, the authors defines used keywords and terms in their protocol. They define non-

repudiation services, based on the “repudiation” definition in [24] (see original paper); it was defined as the “denial by one of the entities involved in a communication of having participated in all or part of the communication”. In transaction protocols, non-repudiation resembles two services: non-repudiation of origin, and non-repudiation of receipt, both defined in the following:

Definition 1: *Non-repudiation of Origin (NRO):* To make sure that the originator of a message can't deny the fact that he sent this message.

Definition 2: *Non-repudiation of Receipt (NRR):* To make sure that the recipient of a message can't deny the fact that he received this message.

The problem of non-repudiation of receipt is similar to the problem of fair exchange of a message, where the originator waits for an acknowledgment that the message was received. So in this protocol, the recipient needs a NRO, while the originator needs NRR, both for ever message exchanged. The similarity of this problem with the fair exchange one necessitates defining what is meant by “Fair” exchange, in addition to other well know properties that must be respected in any fair exchange protocol. These are Fairness, Time-bounded (timeliness), and Viability.

Definition 3: *Fair:* A protocol is said to be fair, if at each step of the execution, either both parties receive the expected items, or none will receive neither expected items nor any information of value concerning the expected items.

Definition 4: *Time-bounded:* A protocol is said to be time-bounded, if at least one party behaves correctly, then the protocol will finish before a finite amount of time.

Definition 5: *Viable:* A protocol is said to be viable, if both parties behave correctly and completed the protocol run normally, then at the end of execution each party will receive his expected item.

– Scheme:

Assuming this protocol is to run between the two parties, Alice (A), and Bob (B). During the protocol, the following evidences will be generated:

- Evidence of Origin of a cipher c : $EOO = \text{Sig}_A(B, l, c)$
- Evidence of Receipt of cipher c : $EOR = \text{Sig}_B(A, l, c)$
- Evidence of Origin of a value v_i : $EOO_i = \text{Sig}_A(B, l, i, v_i)$
- Evidence of Receipt of a value v_i : $EOR_i = \text{Sig}_B(A, l, i, v_i)$

Then the NRO and NRR will be:

$NRO = \{E00, E00\}$

and $NRR = \{EOR, EOR\}$.

Before initiating the protocol, Alice goes through a setup phase where she chooses a random number (n) according to a geometric distribution. This number is kept secretly with Alice during the whole protocol run. (n) will denote the number of iterations of the protocol. After that, $n-1$ random, independent, and equidistributed values r_i are chosen and a key k , both r_i and k are of the same size.

Suppose that Alice want to send a message m to Bob, then she initiates the protocol first by send $c = C_k(m)$, which generates the cipher of the to-be sent message. The cipher is accompanied with an evidence of non-repudiation of origin. Bob acknowledge the received message by sending back the evidence of non-repudiation of receipt. Then Alice send value r_1 along with its NRO_1 , and then Bob acknowledge with NRR_1 , then r_2 , etc...

The process continues; after Alice receives the NRR_{n-1} of r_{n-1} , Alice send the key k , where Bob acknowledge it with NRR_n . Then Bob will wait for the next message from Alice, but there will be none, then after a timeout period, Bob knows that that was the last message, i.e. the key k . Then Bob will be able to decipher c using k .

Its apparent that Bob gets nothing useful before the last step, so he has no interest in terminating the protocol immaturely. The time required by each party to wait for a message must be estimated correctly. Because, we Alice can wait Bob for a long time enough for him to try r_i on c , after this time elapses and Alice didn't receive yet the acknowledgment (NRR), she will consider that Bob is cheating by trying to decipher c using the sent r_i .

Protocol: Markowitch and Roggeman

Step	Message Direction	Message Content
1	A --> B	B, l, c, EOO
2	B --> A	A, l, EOR
3	A --> B	$B, l, 1, r_1, EOO_{k,1}$
4	B --> A	$A, l, EOR_{k,1}$
...
2n-1	A --> B	$B, l, n-1, r_{n-1}, EOO_{k,n-1}$
2n	B --> A	$A, l, EOR_{k,n-1}$
2n+1	A --> B	$B, l, n, k, EOO_{k,n}$
2n+2	B --> A	$A, l, EOR_{k,n}$

Any incorrect message received by any B party will be considered as an error (cheating or network failure), so the recipient will have to stop the protocol. The same case applies for any party sending a message, if transmission didn't start after some (maybe publicly known) deadline, the recipient can consider this delay as a cheating attempt or a network problem, and can thus stop running the protocol.

The number of iteration, n – as said before- is secret, and there is no way that Bob can deduce it from Alice's sent messages, or any other way. But still there is a probability that Bob may guess n , this probability is denoted as δ ; so, $\delta = P(i=n)$. If this is the case, Bob can decipher c and refuse to send back EOR_n . Thus, Bob will have the non-repudiation evidence of origin of m , whereas Alice got no non-repudiation evidence of receipt of m and hence can't create the EOR.

Properties:

It should be noted that this protocol's fairness depends heavily on mainly two things: Timeout, and Cryptosystem used. The timeout period must be enough to get a message sent taking into consideration network's characteristics. Using the deadline (timeout) concept for sending and receiving makes this protocol possible to use in unreliable networks. Also the cryptosystem, the ciphering should be such that its impossible to partially decipher (need less time) the ciphertext and get a meaningful partial plaintext. Cryptosystem to be used must force the deciphering of all message before getting anything of value. Ciphering methods to be chosen may depend on the message size, for instance, an All-or-nothing mode ciphering might not be adequate for short

messages.

At the $(2n+1)^{\text{th}}$ iteration, if the protocol is stopped, no party can gain anything of value (i.e. their EOR and EOO). The probability that Bob will stop accidentally at the $(2n+2)^{\text{th}}$ step is again δ . Therefore, we can say that this protocol is δ -fair.

Mitsianis (2001):

In his paper [no-TTP-8], "A new Approach to enforcing Non-repudiation of receipt", J. Mitsianis proposed a protocol that is similar to [no-TTP-5] by Markowitch and Roggeman. The only difference is that n (the number of iterations or protocol rounds) is static in this protocol, while in Markowitch and Roggeman, its dynamic as its up to the sender to stop or continue the protocol after each completed iteration.

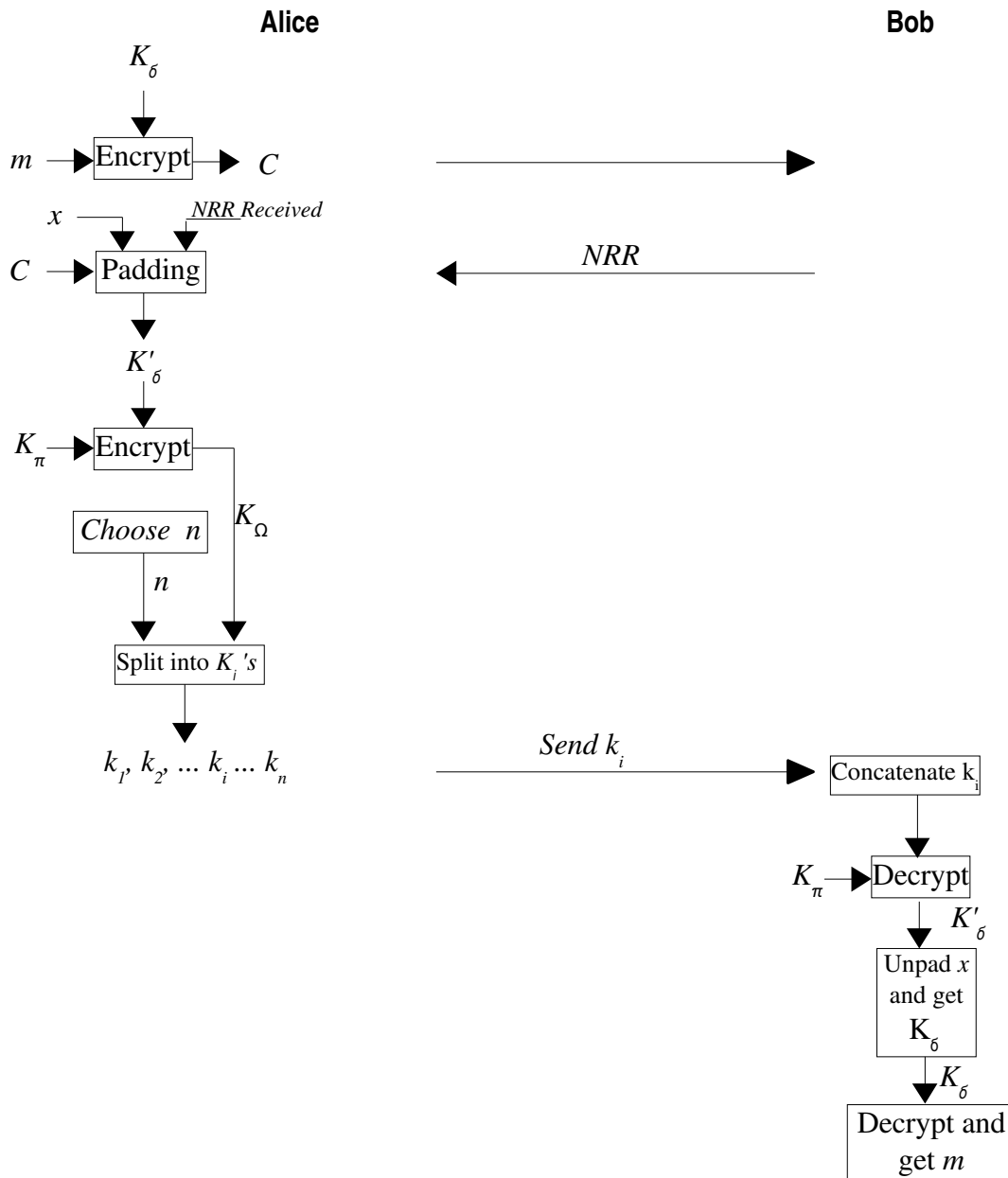
In this protocol, Assuming Alice wants to send Bob a message m , then Alice encrypt m with a secretly chosen session key K_δ , then send the cipher C to Bob. Alice now waits for Bob to send his NRR (see **Definition 2**), after Alice receives the NRR, she appends a padding x to the session key K_δ to have a new key K'_δ . The size of x is secretly chosen by Alice. Then using the shared (between Alice and Bob) session key K_n , Alice encrypts the new padded session key to get a new key, K_Ω , and send it to Bob. The mode of ciphering done at this phase is all-or-nothing mode, i.e. Bob needs to know all the key bits so he can decipher.

After that, Alice chooses randomly and secretly n , the number of protocol rounds. Using this value, Alice splits K_Ω into n K_i parts of different random sizes. The real protocol execution part involves n 2-way transactions (send k_i and receive NRR_i), where Alice sends a K_i and waits for its acknowledgments. At the end Bob concatenates K_i 's to get K_Ω and then decrypt it with K_n . Knowing the size of K_δ , Bob can extract it. Then this key is used to decipher C and get m .

Note that even Bob knows the size of K_δ , he needs to concatenate all K_i 's to get K_δ . This is due to the all-or-nothing mode used in ciphering, in addition to the secrecy of the padding size.

The probability that Bob can read m without sending back the last NRR (NRR_n) is $1/n$. Each message (whether a K_i or an NRR) has a deadline or timeout after which the waiting party can assume a cheating attempt or network error, and hence terminate the execution. In the next figure

the event flow is sketched.



From Alice's side, and after generating the k_i 's // Additional... maybe removed :S

for $i = 1$ to n

 send k_i ;

 if (time-out && !get(NRR _{i}))

 terminate_protocol();

 else

 save NRR _{i}

end for;

References:

[no-TTP-1]

[no-TTP-2] Even, S., and Yacobi, Y., "Relations among Public Key Signature Systems",
Technical Report # 175, Computer Science Dept., Technion, Haifa, Isreal, March
1980.

[no-TTP-3] Even S., "A Protocol for Signing Contracts", Computer Science Dept.,
Technion, Haifa, Isreal, 1983

[no-TTP-4] S. Kermer, O. Markowitch, J. Zhou, "An Intensive Survey of fair Non-
Repudiation Protocols".

[no-TTP-5] O. Markowitch, Y. Roggeman, "Probabilistic Non-repudiation without
Trusted Third Party"

[no-TTP-6] Rabin, "Digitalized Signatures", in Foundation of a secure Computation,
Academic Press, 1978, pp. 155-168.

[no-TTP-7] Merkle, "Secure Communication over Insecure Channel", Communication
of the ACM, Vol.21, April 1978, pp.294-299.

[no-TTP-8] Mitsianis, A New Approach to enforcing non-repudiation of receipt,
manuscript (2001).