Optimistic fair exchange using trusted devices

Mohammad Torabi Dashti

Abstract

Efficiency of optimistic fair exchange using trusted devices is studied. Pfitzmann, Schunter and Waidner (PODC 1998) have shown that four messages in the main sub-protocol is optimal when exchanging idempotent items using non-trusted devices. It is straightforward that when using trusted devices for exchanging non-idempotent items this number can be reduced to three. This however comes at the cost of providing trusted devices with an unlimited amount of storage. We prove that exchanging non-idempotent items using trusted devices with a limited storage capacity requires exactly four messages in the main sub-protocol.

1 Introduction

Fair exchange protocols (hereafter called FE) aim at exchanging items in a *fair* manner. Informally, fair means that all involved parties receive a desired item in exchange for their own, or none of them does so. Deterministic fair exchange protocols cannot be constructed with no presumed trust in the system [6]. Therefore, many FE protocols rely on impartial processes which are trusted by all the protocol participants, hence called trusted third parties (TTPs). In the *optimistic* family of FE protocols, normally the participants execute an optimistic (or, main) sub-protocol which does not involve the TTP at all. However, if a failure maliciously or accidentally occurs, the participants are provided with fallback scenarios, which enable them to recover to a fair state with the TTP's help. When failures are infrequent, optimistic protocols are preferred over those which involve the TTP in each exchange.

In this paper, we study optimistic FE between *trusted computing devices* (TDs). TDs, by construction, follow their certified software, and are guaranteed to observe the terms of use and distribution of digital contents. These devices are nowadays becoming prevalent, particularly in entertainment and multimedia industries. A very common application of TDs pertains to protecting digital contents from unauthorized access (e.g. rendering a media file) and illicit distribution.

Using TDs in optimistic FE protocols is hardly new. TDs have previously been used in order to enrich services provided by FE protocols, e.g. for exchanging time-sensitive data [25], for exchanging electronic vouchers [21], and for robust efficient multi-party computation, which is a general form of fair exchange [9]. Moreover, in [7] a class of FE protocols using TDs is developed which also tolerate accidental failures of devices. Note that TDs are not necessarily operated by honest owners, and usually have to communicate over insecure media. Therefore, using TDs does not entirely obliterate the need for security protocols to ensure fairness in exchange. One would however expect that using TDs results in simpler, more efficient, or value-added FE protocols. Our main contribution here is a negative result concerning two-party FE between TDs: Using TDs does not increase the *efficiency* of optimistic fair exchange protocols in *common practical scenarios*. In the following, we describe what is meant by efficiency and common practical scenarios.

The premise of optimistic FE is that failures are infrequent, and consequently fallback sub-protocols are executed rarely. Therefore, a meaningful measure of efficiency in these protocols is the number of messages exchanged in the main sub-protocol. This number will serve as our measure of efficiency for FE protocols as well. As a convention we refer to n-message FE protocols, where n refers to the number of messages exchanged in their optimistic sub-protocol.

Most existing protocols for fair exchange assume that the items subject to exchange are *idempotent*, meaning that receiving (or possessing) an item once is not different from receiving it multiple times [1, 17]. For instance, once Alice gets access to Bob's signature on a contract, receiving it again does not add anything to Alice's knowledge. The idempotency assumption reflects mass reproducibility of digital contents. Certain digital items are however not idempotent. Electronic vouchers [12, 11] are prominent examples of non-idempotent items. Depending on the implementation, right tokens in various digital rights management schemes are as well digital non-idempotent items, e.g. see [3, 14, 22]. As mentioned above, a common approach to secure use and dissemination of non-idempotent items is to limit their distribution to TDs. We focus on practical scenarios in which fair exchange between TDs needs to ensure that non-idempotent items are not cloned arbitrarily.

Contributions. We confine to two-party exchange protocols. Pfitzmann et al. [18] have shown that four messages in the optimistic sub-protocol are sufficient and necessary for secure fair exchange of idempotent items between non-trusted devices. We show that when exchanging non-idempotent items between TDs, the number of messages in the optimistic sub-protocol can be reduced to three. This would however come at a cost which is often

intolerable in practice: The TDs need to keep record of *all* their previous exchanges. If TDs are provided with limited non-volatile storage capacity (hence not being able to store fingerprints of all their previous exchanges), four messages in the main sub-protocol are proved to be necessary. In order to prove our minimality results, we give a knowledge-based model of optimistic FE protocols between TDs. Our formalization mainly follows [2]. Logics of knowledge have proved to be a useful tool in deriving communication lower bounds in various distributed systems, cf. [8].

Related work. In this paper, we investigate to which extent using trusted computing devices can increase the efficiency of optimistic FE protocols. The only papers on the optimal efficiency of FE protocols, to our knowledge, are [18] for two-party protocols, and [15] for protocols with more than two participants. The bounds derived in these work are relevant for non-trusted participants, and their focus is on exchanging idempotent digital signatures over contracts.

Road map. Section 2 gives an informal introduction to optimistic FE protocols, idempotent and non-idempotent items, TTP logic, etc. In section 3 we develop a knowledge-based model for optimistic FE protocols using TDs. The main result of this section is to formally determine the *resolve pattern* of three-message optimistic FE protocols, by proving that a TD p can successfully complete an exchange only if p *knows* that his opponent q can also successfully complete the exchange. Intuitively, a resolve pattern describes alternatives available to protocol participants in case they are waiting for a message from their opponent and the message does not arrive in time, or received messages at that point do not conform with the protocol. Section 4 concerns FE of non-idempotent items between TDs with limited storage capacity. We give a four message protocol which satisfies the desired security requirements. To prove that four messages in the optimistic sub-protocol are necessary, we build upon the result of section 3 and give a generic replay attack on all the three-message protocols between TDs with limited storage which aim at exchanging non-idempotent items. We also show that the mentioned replay attack can be countered if TDs possess an unlimited storage capacity, by presenting a 3-message protocol for this case. Section 5 concludes the paper.

2 An informal introduction to optimistic FE

Non-idempotent items. We consider *electronic vouchers* as a generic model for non-idempotent items. An electronic voucher, according to RFC 3506, is "a digital representation

of the right to claim goods or services" [11]. A voucher v is a tuple $v = (\langle I, P\rangle, H)$, where I is the voucher's issuer, who guarantees the contents of the voucher, H is the voucher's owner, and P is I's promise to the owner of the voucher (i.e. H). Voucher forgery and alteration are assumed infeasible: No one, except I, can create $\langle I, P\rangle$, and once it is created, no one can alter P. This can be realized, e.g., using secure digital signature schemes. Duplicating vouchers is nonetheless possible, and has to be prevented.

Two voucher duplication scenarios are conceivable: (1) local duplication, where H, the owner of $\langle I, P\rangle$, duplicates the voucher for its own (excessive) use, and (2) remote duplication, where H', a device different from H, gets a copy of $\langle I, P\rangle$ and stores it for its own use, without H destroying its copy of the voucher. Using TDs to store and spend vouchers can prevent local duplications. Security protocols designed for distribution, exchange and use of vouchers are in charge of preventing remote duplications, by ensuring that H destroys $(\langle I, P\rangle, H)$ before H' stores the voucher.

Trusted computing devices. Trusted devices are tamper-proof hardware [1] that, though possibly operated by malicious owners, follow only their embedded sealed software. Trusted devices typically contain a secure scratch memory and (a limited amount of) non-volatile storage capacity. Device X maintains a multiset of vouchers V_X. Before adding the voucher $v = (\langle I, P\rangle, Y)$ to V_X, the device transforms v to $(\langle I, P\rangle, X)$.

The owner of a TD can deliberately turn the device off, or permanently destroy it. We assume that TDs are stateful: If a TD is abruptly turned off, it would resume its previous state when turned on later. This can be realized using various logging systems. For TDs, thus, we assume the crash-recovery failure model with no amnesia, e.g. see [13]. For the moment, we ignore the possibility that the owner delays, blocks or tampers with the messages destined to the device, or transmitted by the device. These issues are discussed within the system and communication model below.

The TTP is a trusted device, which is immune to failures, and has access to an unlimited secure persistent database. It is assumed that the identity of the TTP is a common knowledge in the system.

A computing device which is not trusted, called *non-trusted*, can be faulty. In this case, it would follow the Byzantine failure model.

[1] We do not address the problem of detecting, and revoking, tampered trusted devices. An extensive body of work exists on this topic, see e.g. [4].

System and communication model. We assume a fully connected asynchronous message passing network which connects all TDs; computations are asynchronous, and communication delays, although being finite, are not bounded. The communication channels between any two device X and Y are assumed to be *reliable*, meaning:

- (resilience) No messages are lost in transition.
- (authenticity) A message delivered at Y, has been previously sent by X.
- (confidentiality) Messages sent from X to Y are readable only to Y.

We remark that over reliable channels messages *can* be delayed, reordered or replayed. These operations are usually attributed to an omnipresent *adversary* in the system.

Authenticity and confidentiality [2] can be guaranteed using standard secure encryption and digital signature schemes, assuming a deployed secure public key infrastructure. Below, we discuss the prerequisites and implications of the resilience condition.

Assuming resilient channels, as observed in [1], is unavoidable, in order to guarantee termination of FE protocols (cf. Gray's generals paradox). In practice, if two principals fail to properly establish a channel over computer networks, they can ultimately resort to other communication means, such as various postal services. These services, although being much slower, are very reliable and well protected by law.

The resilience assumption may however seem to be unrealistic when TDs are operated by malicious owners, who may block messages destined to the devices. Below, we argue that such communication failures are subsumed in our device failure model and communication model. Assume that X is a TD which expects to receive a message. Device X either (1) has alternative actions to take if the message does not arrive in time, or (2) no such option is available. The effect of blocking the message in case (1) is the same as delaying the message in the communication network, which is allowed in our model. As a convention, when a message is delayed long enough so that the intended recipient device takes an alternative action, we say that the message has been *intercepted*. In case (2), however, the device would not take any steps unless it receives that very message. Preventing the message to ever arrive, thus, corresponds to turning the device off; this is indeed allowed in our device failure model (see above).

[2] Note that confidentiality is an attribute for channels, but not an attribute for messages transmitted over channels. For example, either the sender or the receiver of a message transmitted over a confidential channel may deliberately share the message with other participants.

Optimistic fair exchange. Below, we briefly introduce optimistic FE. For an extensive exposition of FE in general see [1]. In the following, we opt for a high level description that underlines the exchange patterns, and for the moment we do not focus on exchanges between TDs. Exact message contents are abstracted away, and all messages are assumed to contain enough information for protocol participants to distinguish different protocol instantiations, and different roles in protocols. Detailed specification of these issues is orthogonal to our current purpose.

Optimistic protocols typically consist of three sub-protocols: *main* or *optimistic* sub-protocol, *abort* sub-protocol and *recovery* sub-protocol. Figure 1 depicts a generic main sub-protocol between A and B. The regions in which the other two sub-protocols are alternative possibilities are numbered (1–4) in the figure. In the main sub-protocol, that does not involve the TTP, the agents first *commit* to release their items and then they actually release them. The items subject to exchange, and commitments are respectively denoted by i_A, i_B and $c_A(i_A), c_B(i_B)$. In figure 1 we have $m_1 = c_A(i_A)$, $m_2 = c_B(i_B)$, $m_3 = i_A$ and $m_4 = i_B$. If no failures occur, the participants exchange their items successfully using the main sub-protocol.

If an expected message does not arrive in time, or the arrived message does not conform to the protocol, then the participant expecting that massage can resort to the TTP using abort or recovery sub-protocols. Here we introduce the notion of *resolve patterns*. This notion helps us in reasoning about optimistic FE protocols. Consider again the generic four message protocol shown in figure 1. A resolve pattern characterizes the alternative sub-protocols which are available to participants when they are waiting for a message from their opponent in the main sub-protocol; namely, the alternative sub-protocols envisaged for points 1, 2, 3 and 4 in figure 1.

Four different symbols can be assigned to a point in the resolve pattern: *abort* (a), *recovery* (r), *quit* (q), and *none* ($-$). Intuitively, a (r) means that the device can initiate an abort (resolve) sub-protocol, and q means that in case the expected message does not arrive in time, the participant can safely quit the exchange. Naturally, if no message has been exchanged, the participant quits the protocol, e.g. B is figure 1 quits the exchange, if it does not receive the first message in time. A 'none' option ($-$) indicates that the participant has no alternatives but following the optimistic protocol. It will be proved later (in theorem 3) that 'none' options undermine termination of optimistic FE protocol. This is intuitively because TDs may crash and never send the message their opponent is waiting for. When communicating with the TTP (using resolve sub-protocols), however, TDs know that the message they send to and expect to receive from the TTP will be delivered in a finite time. This is due to resilience of the channels, and the fact that the TTP is immune to failures (see TTP assumptions,

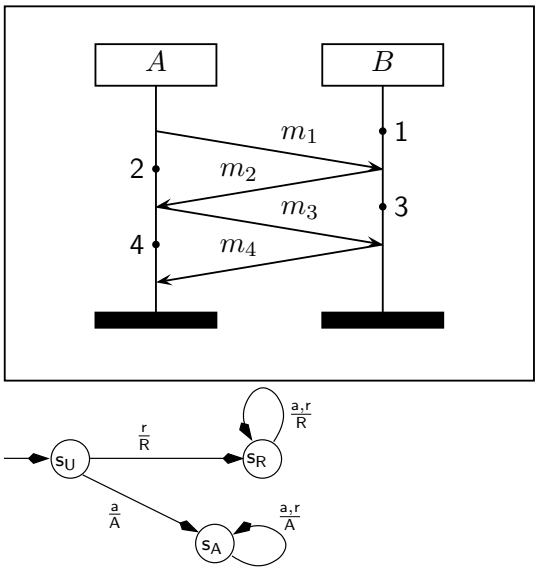

Figure 1: Generic four message protocol (top); Abstract Mealy machine of TTP (down)

above). We use tuples for representing resolve patterns. For instance, a resolve pattern for the protocol of figure 1 can be $\pi = (\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$; then we write $\pi_1 = \mathsf{q}$, $\pi_2 = \mathsf{a}$, etc.

The resolve sub-protocols (abort/recovery) involve the TTP. Without loss of generality we assume that the participant sends its message history (all messages sent and received up to now by the participant in the current execution of the protocol) to the TTP, and based on these the TTP either returns an *abort token* A, or a *recovery token* R. Token A often has no intrinsic value; it merely indicates that the TTP will never send an R token in the context of the current exchange. Token R should however help a participant to recover to a fair state. Although it is impossible for B alone to derive i_A from $c_A(i_A)$ (and vice versa), it is often assumed that the TTP can generate i_A from $c_A(i_A)$, and i_B from $c_B(i_B)$, and that R contains i_A and i_B. In case the TTP cannot do so, usually an affidavit from the TTP is deemed adequate, cf. *weak fairness* [17].

The TTP logic matching the resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$ for the protocol of figure 1 is also shown in figure 1. For each exchanged item, the finite state (Mealy) machine of the TTP is initially in the *undisputed* state $\mathsf{s_U}$. If the TTP receives a valid abort request (from A) while being at state $\mathsf{s_U}$, then it sends back an abort token, and moves to *aborted* state $\mathsf{s_A}$. Similarly, if the TTP is in state $\mathsf{s_U}$, and receives a valid resolve request (from either A or B), then it sends back R, containing i_A and i_B, and moves to *recovered* state $\mathsf{s_R}$. When the TTP is in

either of $\mathsf{s_A}$ or $\mathsf{s_R}$ states, no matter it receives an abort or a recovery request on this exchange, it consistently replies with A or R, respectively.

Security requirements. A process is *correct* if it does not deviate from the terms of the protocol; otherwise it is *faulty*. In particular, a TD is correct if it does not crash. Due to our assumptions, TTP is always correct. A fair exchange protocol is *secure* iff it satisfies the following conditions in presence of any number of faulty processes [1]: [3]

- Timeliness: Any correct process can terminate the protocol in a finite amount of time, with no help from its opponent.
- Fairness: When the exchange terminates, if A owns i_B (or R) and B does not own i_A, then we say the protocol is *unfair* for B. A protocol is *fair* iff it is not unfair for any correct process.
- Functionality: If A and B are correct, and communication delays are negligible, then the A gets i_B and B gets i_A, with no contact to the TTP.

Furthermore, any secure protocol for exchanging non-idempotent items (between TDs) has to guarantee the following requirement [11]:

- No-duplication: The total number of instances of any non-idempotent item v is never increased in the system (i.e. in the V_X sets collectively maintained by TDs).

The scenario in which issuer I injects new vouchers to the system is here considered to occur out-of-band. We remark that assigning unique (serial) numbers to non-idempotent items does not in general address the problem of ensuring no-duplication.

We recall the following theorem from [18].

Theorem 1. *Four messages in the main sub-protocol is sufficient and necessary for secure fair exchange of idempotent items, using non-trusted computing devices.*

Remark 1. The four-message FE protocol that is given in [18] as a witness has the resolve pattern $\pi^{in} = (\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$. It can easily be verified that π^{in} is the *only* resolve pattern suitable for secure fair exchange of idempotent items using non-trusted devices.

[3] Fairness is a safety trace property, timeliness is a liveness trace property, while functionality is not a trace property: It concerns existence of particular traces in the system.

3 A formal model for optimistic FE using TDs

We introduce a minimal formal system for reasoning about FE protocols. The formalization mostly follows the knowledge-based approach of [2], see also [8].

A formal model for protocols. For finite set Σ we write Σ^* for the set of all finite sequences of elements of Σ, containing the empty sequence ϵ. Concatenation of sequences x and y is denoted xy. For two sequences $x, y \in \Sigma^*$, we write $x \leq y$ iff x is a prefix of y, i.e. $\exists z \in \Sigma^*.\ xz = y$; we write $y - x$ for z. We write $x < y$ if $x \leq y$ and $x \neq y$. The *prefix closure* of set $Y \subseteq \Sigma^*$ is defined as $\underline{Y} = \{x \in \Sigma^* \mid \exists y \in Y.\ x \leq y\}$. Set Y is *prefix closed* iff $\underline{Y} = Y$.

We define the set of actions as $Act = S \cup \bar{S} \cup I$, where S, $\bar{S}$ and I are pairwise disjoint, and respectively contain the set of send, receive and internal actions. We assume there is a bijective function $\bar{\cdot} : S \to \bar{S}$ such that $\forall s \in S.\ \bar{a} \in \bar{S}$. Intuitively, $a \in S$ denotes the event of sending a message, and $\bar{a} \in \bar{S}$ stands for the corresponding receive event. A *process* is a prefix closed subset of Act^*. A *protocol* $\mathcal{P}$ is a finite number of processes. We assume the set of actions appearing in different processes are disjoint. This makes it possible to associate a unique process to each action. Let $x \in Act^*$, and write x_p for the sequence of actions that results from x after erasing all the actions *not* performed by process p. We say x is a *computation* of protocol $\mathcal{P}$ iff (1) for all $p \in \mathcal{P}$, x_p belongs to process p, and (2) any $\bar{a} \in \bar{S}$ which appear in x is preceded by a in x. It follows that computations of protocols are prefix closed. That is, any protocol can be seen as a process in itself. We write $a \in x$, with $a \in Act, x \in Act^*$, if a appears in x.

Let us fix a finite nonempty set of propositions Φ, and an *interpretation* function $\mathcal{I} : Act^* \to \Phi \to \{\mathsf{tt}, \mathsf{ff}\}$ which assigns truth values to the elements of Φ, given a computation. We augment the set of propositions with the standard negation and disjunction operators, and also a knowledge-based operator $\mathcal{E}$, in order to define the syntax of our knowledge-based logic *EL*: (1) Every element of Φ is an *EL* formula; (2) If e is an *EL* formula and p a process, then $\mathcal{E}_p(e)$ is an *EL* formula; (3) If e and e' are *EL* formulas, then so are $\neg e$ and $e \vee e'$. Read $\mathcal{E}_p(e)$ as "p knows e".

In the following we introduce the notion of *isomorphism*: Two computation x and y are isomorphic w.r.t. process p iff $x_p = y_p$. Clearly the isomorphism relation is an equivalence relation on the set of all computations. Isomorphism is the core of *EL*'s semantics: Processes have only local views, and cannot therefore distinguish computations which are isomorphic in their view [2]. Models of *EL* formulas are computations. [4] For computation x and *EL*

[4] Executions of a protocol yield a Kripke structure, and we could have defined the semantics of *EL* based

formula e, the relation $x \models e$ is inductively defined as:

- $x \models e$ with $e \in \Phi$ iff $\mathcal{I}(x)e = \mathsf{tt}$.
- $x \models \mathcal{E}_p(e)$ iff $y \models e$ for all computations y that are isomorphic to x w.r.t. p.
- $x \models \neg e$ iff $\neg(x \models e)$.
- $x \models e \vee e'$ iff $x \models e$ or $x \models e'$.

A formal model for fair exchange protocols using TDs. Below, TDs are referred to as processes. We assume that any process p which finishes the optimistic sub-protocol successfully executes an internal action $\top(p)$ and terminates: (Recall that the set of computations of p is prefix closed.)

$$\forall x \in p.\ \top(p) \in x \implies \neg\exists y \in p.\ x < y$$

Since communications with the TTP are over reliable channels, and the TTP is indeed always correct, we choose to model these communications as internal actions for processes. We write $\mathsf{R}(p)$ and $\mathsf{A}(p)$, with $p \in \mathcal{P}$, for receiving the recovery and abort tokens by p. Notation $\mathsf{Q}(p)$ denotes p quiting the exchange. Immediately after executing any of these actions, the process terminates:

$$\begin{aligned}\forall x \in p.\ \mathsf{A}(p) \in x \vee \mathsf{R}(p) \in x \vee \mathsf{Q}(p) \in x \\ \implies \neg\exists y \in p.\ x < y\end{aligned}$$

This condition in particular implies that only one of $\mathsf{A}(p)$, $\mathsf{R}(p)$ or $\mathsf{Q}(p)$ can appear in any execution x.

When a process p crashes it simply executes $\bot(p)$ and terminates:

$$\forall x \in p.\ \bot(p) \in x \implies \neg\exists y \in p.\ x < y$$

Since any process p (except TTP) may crash at any moment (before terminating) we assume:

$$\begin{aligned}\forall x \in p.\ \top(p) \notin x \wedge \bot(p) \notin x \wedge \mathsf{A}(p) \notin x \\ \wedge \mathsf{R}(p) \notin x \wedge \mathsf{Q}(p) \notin x \implies x\bot(p) \in p\end{aligned}$$

The consistent behavior of the TTP is captured via considering only *TTP-consistent* computations, as defined below. Computation x of an FE protocol is TTP-consistent iff

on these structures. This is however unnecessary for our analyzes. We have thus opted for the simpler setting of computations as models of *EL* formulas.

- If $\mathsf{R}(p)$ appears in x for process p, then $\mathsf{A}(q)$ does not appear in x for any $q \in \mathcal{P}$.

We also need to assert that if a correct process has the choice to, e.g., contact the TTP in computations x, then the computations x also allows (or, covers) this possibility. That is, we confine only to *maximal* computations. For a computation of protocol $\mathcal{P}$ like x, we say x is maximal iff

- For any $p \in \mathcal{P}$, for all $y \in p$ such that $x_p < y$ and $\bot(p) \notin y - x_p$, we have $x(y - x_p)$ is not a computation in $\mathcal{P}$.

This constraint, intuitively, implies that computation x is considered maximal only if no process p can progress further in computation x except by crashing. Note that since processes are prefix closed, if p can progress further than x_p, then there exists a $y \in p$ such that $y - x_p$ contains only one action.

TTP-consistence and maximality can be seen as *fairness constraints* on protocol computations, cf. [10]. From this point on, by a computation we mean a TTP-consistent and maximal computation, unless otherwise stated.

Fair exchange security requirements. A protocol $\mathcal{P}$ is a secure fair exchange protocol iff it satisfies the following properties. Here, $x : \mathcal{P}$ stands for "computation x belongs to protocol $\mathcal{P}$".

- Functionality:
$$\exists x : Prot. \forall p \in \mathcal{P}.\ \top(p) \in x$$
- Timeliness:
$$\begin{aligned}\forall x : \mathcal{P}. \forall p \in \mathcal{P}.\ & \mathsf{Q}(p) \in x \vee \mathsf{A}(p) \in x \\ & \vee \mathsf{R}(p) \in x \vee \top(p) \in x \vee \bot(p) \in x\end{aligned}$$
- Fairness:
$$\begin{aligned}\forall x : \mathcal{P}. \forall p, q \in \mathcal{P}.\ & \\ & (\mathsf{R}(p) \in x \vee \top(p) \in x) \implies \\ & (\mathsf{R}(q) \in x \vee \top(q) \in x \vee \bot(q) \in x)\end{aligned}$$

The following theorem is the main technical result of our formalization that relates fairness to knowledge. For a computation x and $p \in \mathcal{P}$, we write $\mathcal{I}(x)\mathbb{G}(p) = \mathsf{tt}$ iff $\top(p) \in x \vee \mathsf{R}(p) \in x$.

Theorem 2. *For any computation x in a protocol between p and q that satisfies fairness, $x \models \mathbb{G}(p)$ only if $x \models \mathcal{E}_p(\mathbb{G}(q) \vee \bot(q))$.*

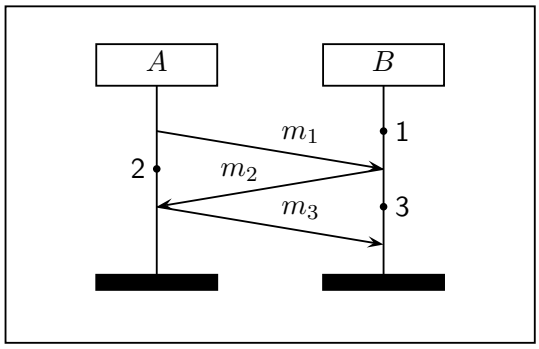

Figure 2: Generic three message protocol.

Proof. Let $x \models \mathbb{G}(p)$, and assume y is any computation in the protocol such that $x_p = y_p$. We need to show that $y \models \mathbb{G}(q) \vee \bot(q)$. From $x_p = y_p$ we conclude $y \models \mathbb{G}(p)$. As the protocol is assumed to satisfy fairness we get $y \models \mathbb{G}(q) \vee \bot(q)$. □

Intuitively, the theorem states that p can add an item i_q to V_p iff p *knows* that a correct q would add i_q to V_q.

3.1 Three-message protocols for FE using TDs

In this section we determine the resolve pattern of any three-message FE protocol that satisfies functionality, timeliness and fairness. Figure 2 shows a generic three message protocol. Our focus in this section is on *optimistic non-redundant* protocols, as defined below. A protocol is optimistic iff it satisfies functionality and $\pi_1 \notin \{\mathsf{r}, \mathsf{a}\}$. The intuition behind this definition is that by functionality the protocol has at least one successful computation without contacting the TTP, and the condition $\pi_1 \notin \{\mathsf{r}, \mathsf{a}\}$ implies that in case the receiver of the first message, say Bob, in the protocol does not receive this message he can either wait, or quite the exchange, but may not contact the TTP. In other words, if no messages are exchanged in Bob's view, then he does not contact the TTP.

An optimistic protocol with ℓ messages is non-redundant iff the protocol has no computation of length less that $2\ell + 2$ which contains both $\top(p)$ and $\top(q)$. Here, 2ℓ counts all the m_i and $\bar{m}_i$ for $1 \leq i \leq \ell$, and then two actions $\top(p)$ and $\top(q)$ are added to the result. Intuitively, this implies that all the messages of the protocol are required to be exchanged in order to successfully complete the protocol, with no TTP interventions.

Below, in order to capture the intuitive meaning of resolve patterns we require that given

resolve pattern $\pi = (\pi_1, \pi_2, \pi_3)$ and any computation x in which only, say p, contacts the TTP at a point corresponding to π_i, the TTP answer with R if $\pi_i = \mathsf{r}$ and the TTP will answers with A if $\pi_i = \mathsf{a}$. This is in accordance with the description of TTP logic in section 2. The proof of the following theorem relies on theorem 2, and can be found in appendix A.

Theorem 3. *The resolve pattern of any three-message optimistic FE protocol between p and q that satisfies fairness, timeliness and functionality, is necessarily $\pi = (\mathsf{q}, \mathsf{a}, \mathsf{r})$.*

The resolve pattern determined in theorem 3 is a necessary, but not generally sufficient, condition for satisfying fairness, timeliness and functionality. In case processes are not trusted, there is no three-message protocol for secure FE, cf. theorem 1.

4 Optimistic FE of non-idempotent items between TDs

Below, we focus on exchanging non-idempotent items between TDs. In section 4.1 we give a four-message FE protocol for exchanging non-idempotent items between TDs with limited storage capacity. It is worth mentioning that the resolve pattern of the protocol of section 4.1 is different from π^{in} (see remark 1); using π^{in} would require TDs with unlimited storage capacity. In section 4.2 we show that four messages in the optimistic sub-protocol are necessary by giving a generic replay attack on any three-message FE protocol between TDs with limited storage capacity. Protocols with one or two messages in the optimistic sub-protocol are not discussed, due to their trivial inadequacy. In section 4.3 we give a three-message FE protocol for exchanging non-idempotent items between TDs with unlimited storage capacity.

4.1 A four-message FE protocol between TDs with limited storage

It can be easily verified that the resolve pattern π^{in} is not suitable for exchanging non-idempotent items. Namely, there exists a generic replay attack on protocols with resolve pattern π^{in} which can be countered only if the TDs have access to an unlimited amount of secure storage. The attack is due to the no-duplication requirement. The proof of the following proposition is given in appendix A.

Proposition 1. *The resolve pattern $\pi^{in} = (\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$ is not secure for fair exchange of non-idempotent items, using TDs with limited storage capacity.*

Next, we present a protocol with resolve pattern $(\mathsf{q}, \mathsf{q}, \mathsf{a}, \mathsf{r})$ for exchanging non-idempotent items using trusted devices with limited storage capacity. This protocol is inspired by a

protocol of Terada et al. [21]. [5]

In order to give a detailed description of the protocol, we relax the integrity, authenticity and confidentiality assumptions on communication channels (cf. section 2) for proving theorem 4, and also theorem 6. Below, $[M]_X$ denotes message M signed by participant X; and M can be extracted from $[M]_X$. We write $h(M)$ for the hash value of M, where h is a one-way secure hash function. A secure PKI infrastructure is also assumed to be in place. The cryptographic apparatus are assumed to be *ideal*, as in [5].

Theorem 4. *Resolve pattern* $\pi = (\mathsf{q}, \mathsf{q}, \mathsf{a}, \mathsf{r})$ *can be used for secure fair exchange of non-idempotent items, using TDs with limited storage capacity.*

Proof. Consider an instantiation of the 4-message protocol and the TTP logic of figure 1, with resolve pattern π. Initially $v \in V_A$, $v' \in V_B$, and A and B want to exchange v for v'. We assume that

1. A temporarily removes v from V_A when starting the exchange. If A receives token A, it puts v back into V_A. Upon a successful exchange with B, or receiving token R, A adds v' to V_A and destroys v. A similar assumption is made for B.

2. A and B are programmed such that once they start the resolve sub-protocols, they will ignore all the messages from the main sub-protocol.

These assumptions are tenable, since A and B are trusted devices.

The specifications for initiator A and responder B are given in algorithm 1 and algorithm 2, respectively. We assume that confidentiality of the vouchers v and v' is not a concern (otherwise, communications between A, B and the TTP have to be encrypted in the following). The message contents for the protocol (referred to in algorithms 1 and 2) are described below. We recall that messages are communicated over resilient channels, and A and B are TDs.

- $m_1 := [v, v', B, n]_A$, where n is a fresh nonce generated by A.
- $m_2 := [h(v, v', A, B, n), h(n')]_B$, where n' is a fresh nonce generated by B.
- $m_3 := [h(n')]_A$
- $m_4 := n'$

[5] Terada et al. first presented a similar, but flawed, protocol in [20]. We have spotted the flaw, and upon contacting the authors, realized the protocol has been patched in [21].

Algorithm 1 Specification of initiator A in theorem 4

$V_A := V_A \setminus \{v\}$
send m_1 to B
recv m_2 from B
if recv times out **then**
  quit
send m_3 to B
recv m_4 from B
if recv times out **then**
  send recovery request r to TTP
  if recv abort token A from TTP **then**
    $V_A := V_A \cup \{v\}$
  else if recv recovery token R from TTP **then**
    $V_A := V_A \cup \{v'\}$
else
  $V_A := V_A \cup \{v'\}$

Algorithm 2 Specification of responder B in theorem 4

recv m_1 from A
if recv times out **then**
  quit
$V_B := V_B \setminus \{v'\}$
send m_2 to A
recv m_3 from A
if recv times out **then**
  send abort request a to TTP
  if recv abort token A from TTP **then**
    $V_B := V_B \cup \{v'\}$
  else if recv recovery token R from TTP **then**
    $V_B := V_B \cup \{v\}$
else
  send m_4 to A
  $V_B := V_B \cup \{v\}$

We assume upon receiving a message, the TDs check the integrity of the message, and its conformance to the protocol. A bogus message is destroyed, and considered as not having been received. For contacting the TTP, the following messages are used:

- $\mathsf{a} := [f_1, A, B, v, v', n, h(n')]_B$
- $\mathsf{r} := [f_2, A, B, v, v', n, h(n')]_A$
- $\mathsf{A} := [ack(f_1), A, B, v, v', n, h(n')]_{TTP}$
- $\mathsf{R} := [ack(f_2), A, B, v, v', n, h(n')]_{TTP}$

Here f_1, f_2 and $ack(f_1)$ and $ack(f_2)$ are unique flags respectively denoting an abort request, a resolve request, an abort token, and a recovery token. Notice that in this protocol, the TTP can readily extract v and v' from m_1 and m_2. In fact, to recover to a fair state, the participants do not require the contents of their opponent's item, but rather the permission to add the item to their local voucher set.

A complete security analysis of the protocol is omitted due to space constraints. We however note that assumption (1) in the beginning of this proof, and fairness imply that during exchanges no items are duplicated. A subtlety here is to ensure that replay attacks are not possible. Note that abort option for B (that is $\pi_3 = \mathsf{a}$) thwarts the replay attack described in proposition 1. □

4.2 Four messages are necessary for FE between TDs with limited storage

Theorem 3 maps out the resolve pattern of any three-message FE protocol that satisfies fairness, timeliness and functionality; namely, $\pi = (\mathsf{q}, \mathsf{a}, \mathsf{r})$. Below, we use this result to show that any three-message protocol that is used for exchanging non-idempotent items between TDs with limited storage capacity is susceptible to a generic replay attack.

Theorem 5. *There is no three-message protocol for secure exchange of non-idempotent items between TDs with limited storage capacity.*

Proof. Assume there exists a three-message FE protocol for secure exchange of non-idempotent items. The resolve pattern of the protocol needs to be $(\mathsf{q}, \mathsf{a}, \mathsf{r})$, due to theorem 3. Now, assume the protocol is repeatedly executed between processes p and q. Let computation x be one of these computations which has completed successfully without resorting to the TTP. Such a computation exists due to the functionality property. Let

$$x = m_1 \bar{m}_1 m_2 \bar{m}_2 m_3 \bar{m}_3 \top(p) \top(q)$$

Since the processes have limited storage capacity, there exists a point in time θ, when all the information about computation x is erased from the storage of p and q. Note that at time θ, the adversary can replay m_1. Now the computation $y = m_1\bar{m}_1 m_2 \mathsf{R}(q)$ is a valid computation: It is maximal, and indeed TTP-consistent. Note that p has no actions to perform in this computation, and is in fact not even "aware" that the exchange is happening. Clearly y violates the no-duplication property of non-idempotent items. It is worth mentioning that fairness is not violated in computation y. □

Remark that simply assigning sequence numbers to different transactions between TDs A and B *does* prevent the replay attack described in theorem 5. However, such sequence numbers must be of an unbounded length, in order to prevent repetition. That is, TDs require an unbounded storage capacity to store the sequence numbers in general.

4.3 A three-message FE protocol between TDs with unlimited storage

In this section, we give a three-message FE protocol with resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r})$ for exchanging non-idempotent items between TDs with unlimited storage capacity. The main idea of the protocol is that TDs can use their unlimited storage to counter the replay attack described in theorem 5.

Theorem 6. *Resolve pattern $\pi = (\mathsf{q}, \mathsf{a}, \mathsf{r})$ can be used for secure fair exchange of non-idempotent items, using TDs with unlimited storage capacity.*

Proof. Consider the 3-message protocol of figure 2 with the TTP logic of figure 1 and resolve pattern π. Initially $v \in V_A$, $v' \in V_B$, and A and B want to exchange v for v'. We assume

1. A temporarily removes v from V_A when starting the exchange. If A receives token A, it puts v back into V_A. Upon a successful exchange with B, or receiving token R, A adds v' to V_A and destroys v. A similar assumption is made for B.

2. A and B are programmed such that once they start the resolve sub-protocols, they will ignore all the messages from the main sub-protocol.

These assumptions are tenable, since A and B are trusted devices.

The specifications for initiator A and responder B are given in algorithm 3 and algorithm 4, respectively. We assume that confidentiality of the vouchers v and v' is not a concern (otherwise, communications between A, B and the TTP have to be encrypted in the following). The message contents for the protocol (referred to in algorithms 3 and 4) are described below. We recall that messages are communicated over resilient channels, and A and B are TDs.

Algorithm 3 Specification of initiator A in lemma 6

$V_A := V_A \setminus \{v\}$
send m_1 to B
recv m_2 from B
if recv times out **then**
  send abort request a to TTP
  if recv abort token A from TTP **then**
    $V_A := V_A \cup \{v\}$
  else if recv recovery token R from TTP **then**
    $V_A := V_A \cup \{v'\}$
else
  send m_3 to B
  $V_A := V_A \cup \{v'\}$

Algorithm 4 Specification of responder B in lemma 6

REQUIRES: History of previous exchanges of B, called $\mathcal{H}_B$

recv m_1 from A
if $m_1 \in \mathcal{H}_B$ or recv times out **then**
  quit
$V_B := V_B \setminus \{v'\}$
$\mathcal{H}_B := \mathcal{H}_B \cup \{m_1\}$
send m_2 to A
recv m_3 from A
if recv times out **then**
  send recovery request r to TTP
  if recv abort token A from TTP **then**
    $V_B := V_B \cup \{v'\}$
  else if recv recovery token R from TTP **then**
    $V_B := V_B \cup \{v\}$
else
  $V_B := V_B \cup \{v\}$

- $m_1 := [v, v', B, h(n)]_A$, where n is a fresh nonce generated by A.
- $m_2 := [h(v, v', A, B, h(n))]_B$
- $m_3 := [n]_A$

We assume upon receiving a message, the TDs check the integrity of the message, and its conformance to the protocol. A bogus message is destroyed, and considered as not having been received. For contacting the TTP, the following messages are used:

- $\mathsf{a} := [f_1, A, B, v, v', h(n)]_A$, where f_1 is a unique flag denoting an abort request.
- $\mathsf{r} := [f_2, A, B, v, v', h(n)]_B$, where f_2 is a unique flag denoting a recovery request.
- $\mathsf{A} := [ack(f_1), A, B, v, v', h(n)]_{TTP}$
- $\mathsf{R} := [ack(f_2), A, B, v, v', h(n)]_{TTP}$

Here $ack(f_1)$ and $ack(f_2)$ are unique flags respectively denoting an abort and a recovery token. A complete security analysis of the protocol is omitted due to space constraints. We however note that assumption (1) in the beginning of this proof, and fairness imply that during exchanges no items are duplicated. A subtlety here is to ensure that replay attacks are not possible. Note that the *history set* $\mathcal{H}_B$ in device B is used prevent the replay attack described in theorem 5. □

Remark 2. Micali has proposed [16] a three-message protocol for fair exchange of idempotent items between non-trusted devices, which has the resolve pattern $(\mathsf{q}, -, \mathsf{r})$. That is, A cannot run the abort sub-protocol (A's access to abort jeopardizes fairness if A is non-trusted). As A is not provided with any means to contact the TTP in Micali's protocol, in case A does not receive m_2, timeliness is violated (as observed in [1]), since A can terminate the protocol only when B takes actions. The resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r})$ of theorem 6 has also been used in [19] and [24] for secure fair exchange of the so-called *revocable* digital contents; intuitively the TTP's testimony is necessary for the validity of (some of) the exchanged items in these protocols.

5 Concluding remarks

We analyze the efficiency of optimistic protocols for fair exchange of non-idempotent items using trusted devices. Four messages in the main sub-protocol is proved to be necessary,

given that the trusted devices have access to a limited amount of storage. With an unlimited non-volatile storage, this number can however be reduced to three.

If (some) non-idempotent items are only of temporary value, it is possible to identify and eliminate those fingerprints of the previous exchanges which are obsolete, hence irrelevant, for the security of the protocol. such "garbage collection" procedures can reduce the amount of required secure storage on trusted devices. Investigating this area is left for future work.

Also, it must be interesting to explore (the efficiency of) the fair exchange protocols which guarantee *atomicity* for non-idempotent items, that is no-duplication and also no-destruction. Atomicity is a typical requirement for financial transactions [23].

Acknowledgement

This research has been carried out under the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures".

References

[1] N. Asokan. *Fairness in electronic commerce*. PhD thesis, Uni. Waterloo, 1998.

[2] K. Chandy and J. Misra. How processes learn. *Distrib. Comput.*, 1(1):40–52, 1986.

[3] C. Chong, S. Iacob, P. Koster, J. Montaner, and R. van Buuren. License transfer in OMA-DRM. In *ESORICS '06*, volume 4189 of *LNCS*, pages 81–96. Springer, 2006.

[4] B. Chor, A. Fiat, M. Naor, and B. Pinkas. Tracing traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000.

[5] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.

[6] S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Computer Science Dept., Technion, Haifa, Isreal, March 1980.

[7] P. D. Ezhilchelvan and S. K. Shrivastava. A family of trusted third party based fair-exchange protocols. *IEEE Trans. Dependable Secur. Comput.*, 2(4):273–286, 2005.

[8] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT, 2003.

[9] M. Fort, F. Freiling, L. Draque Penso, Z. Benenson, and D. Kesdogan. TrustedPals: Secure multiparty computation implemented with smart cards. In *ESORICS '06*, volume 4189 of *LNCS*, pages 34–48. Springer, 2006.

[10] N. Francez. *Fairness*. Springer, 1986.

[11] K. Fujimura and D. Eastlake. Requirements and Design for Voucher Trading System (VTS). RFC 3506, March 2003.

[12] K. Fujimura, H. Kuno, M. Terada, K. Matsuyama, Y. Mizuno, and J. Sekine. Digital-ticket-controlled digital ticket circulation. In *USENIX Security '99*, pages 229–240, 1999.

[13] R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer, 2006.

[14] N. Kuntze and A. Schmidt. Trusted ticket systems and applications. In *IFIP SEC '07*, volume 232 of *IFIP*, pages 49–60. Springer, 2007.

[15] S. Mauw, S. Radomirovic, and M. Torabi Dashti. Minimal message complexity of asynchronous multi-party contract signing. In *CSF '09*, pages 13–25. IEEE CS, 2009.

[16] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC '03*, pages 12–19. ACM Press, 2003.

[17] H. Pagnia, H. Vogt, and F. C. Gärtner. Fair exchange. *Computer Journal*, 46(1):55–7, 2003.

[18] B. Pfitzmann, M. Schunter, and M. Waidner. Optimal efficiency of optimistic contract signing. In *PODC '98*, pages 113–122. ACM Press, 1998.

[19] M. Schunter. *Optimistic fair exchange*. PhD thesis, Universität des Saarlandes, 2000.

[20] M. Terada, M. Iguchi, M. Hanadate, and K. Fujimura. An optimistic fair exchange protocol for trading electronic rights. In *CARDIS '04*, pages 255–270. IFIP, 2004.

[21] M. Terada, K. Mori, K. Ishii, S. Hongo, T. Usaka, N. Koshizuka, and K. Sakamura. A framework for distributed inter-smartcard communication. *IPSJ Digital Courier*, 2:120–132, 2006.

[22] M. Torabi Dashti, S. K. Nair, and H. Jonker. Nuovo DRM Paradiso: Designing a secure, verified, fair exchange DRM scheme. *Fundam. Inform.*, 89(4):393–417, 2008.

[23] J. Tygar. Atomicity in electronic commerce. In *PODC '96*, pages 8–26. ACM press, 1996.

[24] H. Vogt. Asynchronous optimistic fair exchange based on revocable items. In *Financial Cryptography*, volume 2742 of *LNCS*, pages 208–222. Springer, 2003.

[25] H. Vogt, H. Pagnia, and F. C. Gärtner. Using smart cards for fair exchange. In *Electronic Commerce '01*, volume 2232 of *LNCS*, pages 101–113. Springer, 2001.

A Proofs omitted in the text

A.1 Proof of theorem 3

The results below concern optimistic non-redundant protocols. We continue with a few auxiliary lemmas. By abusing the notation, in the following we may write $\pi_i(p)$ instead of TTP's answer to request π_i sent by process p.

Lemma 1. *In any three-message protocol with resolve pattern (π_1, π_2, π_3) between p and q that satisfies timeliness, we have $\pi_i \neq -$, for $i = 1, 2, 3$.*

Proof. Let $\pi_i = -$ for some i. Consider the computation x in which the process who has to send m_i crashes: $x = x_1 \perp(p)$ for some $x_1 \in Act^*$. Note that x is maximal since $q \neq p$ cannot progress (due to $\pi_i = -$) and p has already crashed. The computation x is a counterexample to timeliness if the protocol is non-redundant. □

Lemma 2. *In any three-message protocol with resolve pattern (π_1, π_2, π_3) between p and q that satisfies fairness, timeliness and functionality, $\pi_3 = \mathsf{r}$.*

Proof. By lemma 1, $\pi_3 \in \{\mathsf{q}, \mathsf{a}, \mathsf{r}\}$. By functionality and non-redundancy, computation $x = m_1 \bar{m}_1 m_2 \bar{m}_2 m_3 \bar{m}_3 \top(A) \top(B)$ is a computation of the protocol. Since $y = m_1 \bar{m}_1 m_2 \bar{m}_2 m_3 \top(A) \pi_3(B)$ is isomorphic to x w.r.t. A, theorem 2 implies $\pi_3(B) = \mathsf{R}(B)$. That is $\pi_3 = \mathsf{r}$. □

Lemma 3. *In any three-message protocol with resolve pattern (π_1, π_2, π_3) between p and q that satisfies fairness, timeliness and functionality, $\pi_2 = \mathsf{a}$.*

Proof. By lemma 1, $\pi_2 \in \{\mathsf{q}, \mathsf{a}, \mathsf{r}\}$. Consider computations $x = m_1 \pi_2(A) \pi_1(B)$ and $y = \epsilon \pi_1(B)$. In y, B quits the exchange, due to the protocol being optimistic. Since computations x and y are isomorphic w.r.t. B, we have $\pi_1(B) = \mathsf{Q}(B)$ in x. Then, fairness for x

implies that either $\pi_2(A) = \mathsf{A}(A)$ or $\pi_2(A) = \mathsf{Q}(A)$. That is $\pi_2 \in \{\mathsf{a}, \mathsf{q}\}$ (†). Below, we show $\pi_2 \neq \mathsf{q}$.

Consider the computation $x' = m_1 \bar{m}_1 m_2 \pi_2(A) \pi_3(B)$. Clearly x' is a maximal computation of the protocol. Assume towards a contradiction that $\pi_2(A) = \mathsf{Q}(A)$. Due to lemma 2, then $\pi_3(B) = \mathsf{R}(B)$, and indeed the computation x' would be TTP-consistent. This computation, however, clearly violates the fairness property. Therefore, $\pi_2 \in \{\mathsf{a}, \mathsf{r}\}$. Given (†), we conclude $\pi_2 = \mathsf{a}$. □

Theorem 3 is now immediate by lemmas 1, 2 and 3.

A.2 Proof of proposition 1

The resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$ is not secure for fair exchange of non-idempotent items, using TDs with limited storage capacity.

Proof. Consider the generic protocol of figure 1. Let us assume A sends m_1 to B, but m_2 is intercepted. Next, B runs the recovery sub-protocol and obtains i_A and destroys i_B (recall that i_A and i_B are non-idempotent items). When this session is terminated from B's point of view (while A is still waiting), message m_1 is replayed (by the adversary), and A and B finish this exchange session successfully. Suppose that at the beginning A had one instance of i_A, and B had two instances of i_B. At the end of this scenario, A has one instance of i_B, while B has two instances of i_A. Therefore, i_A is duplicated, witnessing that the pattern is insecure. This attack can be countered if B does not reply to the replay of message m_1 (note that freshness the messages is not guaranteed over reliable channels). Detecting replays would however require B to keep record of (fingerprints of) all its previous exchanges, which is not possible for B with a limited storage capacity. □