Efficiency of optimistic fair exchange using trusted devices

Mohammad Torabi Dashti
ETH Zürich, Switzerland

Efficiency of asynchronous optimistic fair exchange using trusted devices is studied. It is shown that three messages in the optimistic sub-protocol are sufficient and necessary for exchanging idempotent items. When exchanging non-idempotent items, however, three messages in the optimistic sub-protocol are sufficient only under the assumption that trusted devices have unbounded storage capacity. This assumption is often not satisfiable in practice. It is then proved that exchanging non-idempotent items using trusted devices with a bounded storage capacity requires exactly four messages in the optimistic sub-protocol.

Categories and Subject Descriptors: C.2.2 [**Network Protocols**]: Applications; D.2.4 [**Software/Program Verification**]: Formal methods; K.4.4 [**Electronic Commerce**]: Security

General Terms: Optimistic fair exchange protocols, Trusted computing devices, Minimal message complexity

Additional Key Words and Phrases: Logics of knoweldge, Non-idempotent items

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 [Even and Yacobi 1980]. Therefore, many FE protocols rely on impartial processes which are trusted by all the protocol participants, hence called trusted third parties (TTPs). In the presence of a TTP, there is a canonical solution to FE. The items subject to exchange can be sent to the trusted entity and then he would distribute them if all the items arrive in time. If some items do not arrive in time, the TTP would simply abandon the exchange. 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). By construction, TDs 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 the music industry. A common application of TDs pertains to protecting digital contents from unauthorized access (e.g. rendering a video file) and illicit distribution.

Note that TDs are not necessarily operated by honest owners, and they 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

Computing devices \ Items	non-idempotent	idempotent
non-trusted	–	4
trusted, with unbounded storage capacity	3	3
trusted, with bounded storage capacity	4	3

Table I. Optimal number of messages in two-party optimistic sub-protocol

however expect that using TDs results in simpler, more efficient, or value-added FE protocols. In this paper we show that using TDs does not increase the *efficiency* of two-party optimistic fair exchange protocols in *common practical scenarios*. Below, we explain what is meant by efficiency and by 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 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 [Asokan 1998; Pagnia et al. 2003]. 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 [Fujimura et al. 1999; Fujimura and Eastlake 2003] 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 [Kuntze and Schmidt 2007; Torabi Dashti et al. 2008]. 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.

1.1 Contributions

We confine to two-party asynchronous optimistic FE protocols. Pfitzmann, Schunter and Waidner have shown that under reasonable cryptographic assumptions four messages in the optimistic sub-protocol are sufficient and necessary for secure fair exchange of idempotent items between non-trusted devices [Pfitzmann et al. 1998]. 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 have a bounded 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 and indeed sufficient. Table I summarizes these results. Note that exchanging non-idempotent items between non-trusted devices is inherently insecure.

1.2 Proof technique

In order to prove our minimality results, we give a knowledge-based model of optimistic FE protocols between TDs. Our formalization follows the approach of Chandy and Misra [Chandy and Misra 1986]. Logics of knowledge have proved to be a useful tool in deriving communication lower bounds in various distributed systems, cf. [Fagin et al. 1995]. The main result of the formalization is to 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 the 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.

For FE of non-idempotent items between TDs with a bounded 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 aforementioned resolve pattern and give a generic replay attack on all the three-message protocols between TDs with a bounded storage capacity which aim at exchanging non-idempotent items. The replay attack can nonetheless be countered if TDs have unbounded storage capacity. This is proved by presenting a three-message protocol for exchanging non-idempotent items between TDs with unbounded storage capacity.

1.3 Related work

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 [Vogt et al. 2001], for exchanging electronic vouchers [Terada et al. 2006], and for robust efficient multi-party computation, which is a general form of fair exchange [Fort et al. 2006]. A class of FE protocols which use TDs to tolerate accidental failures of devices is presented in [Ezhilchelvan and Shrivastava 2005].

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 [Pfitzmann et al. 1998] for two-party protocols, and [Mauw et al. 2009] 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.

1.4 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. In section 4 we prove that four messages are sufficient and necessary for FE of non-idempotent items between TDs with a bounded storage capacity. There we also present a three-message protocol for this purpose, given that the TDs have unbounded storage capacity. Practical aspects of exchanging non-idempotent items using TDs are explored in section 5. Section 6 concludes the paper, with pointing out possible future extensions.

2. AN INFORMAL INTRODUCTION TO OPTIMISTIC FAIR EXCHANGE

2.1 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" [Fujimura and Eastlake 2003]. 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, obtains 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 distributing, exchanging and spending vouchers are in charge of preventing remote duplications, by ensuring that H destroys $(\langle I, P\rangle, H)$ before H' stores the voucher.

2.2 Trusted computing devices

Trusted devices are tamper-proof hardware that, though possibly operated by malicious owners, follow only their embedded sealed software. Here, we do not consider the problem of detecting, and revoking, tampered devices. An extensive body of work exists on this topic, cf. [Chor et al. 2000].

Trusted devices typically contain a secure scratch memory and non-volatile storage (of a bounded 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 mechanisms. For TDs, thus, we assume the crash-recovery failure model with no amnesia, e.g. see [Guerraoui and Rodrigues 2006]. 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 a secure persistent database of unbounded capacity. 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.

2.3 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 (bidirectional) communication channel between

any two device X and Y is 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 messages *can* be delayed, reordered and replayed over reliable channels. These operations are attributed to an omnipresent *adversary*.

Authenticity and confidentiality of the channels can be guaranteed using standard secure encryption and digital signature schemes. Note that here 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.

Below, we discuss the prerequisites and implications of the resilience assumption. Assuming resilient channels, as observed in [Asokan 1998], is unavoidable, in order to guarantee termination of FE protocols (cf. Gray's generals paradox [Gray 1978]). In practice, if two principals fail to establish a resilient channel to exchange messages 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 section 2.2).

2.4 Optimistic fair exchange

Below, we briefly introduce optimistic FE. For an extensive exposition of FE in general see [Asokan 1998; Torabi Dashti and Mauw 2010]. Below, 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 sufficient information for protocol participants to distinguish different protocol instantiations, and roles in protocols. Detailed specification of these issues is orthogonal to the purpose of this sub-section.

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, which does not involve the TTP, the principals first *commit*

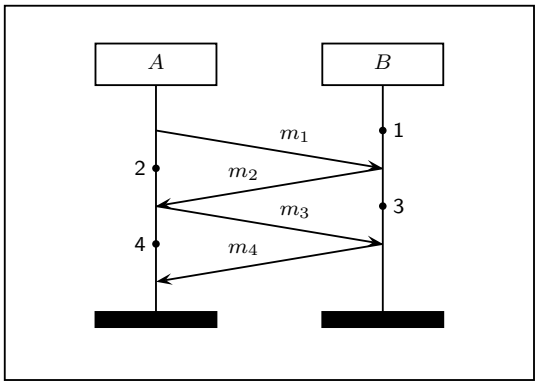

Fig. 1. Generic four message protocol

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 successfully exchange their items 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 participant can initiate an abort (recovery) 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 in 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. We remark that when communicating with the TTP (using abort or recovery 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, 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 (i.e. abort and recovery) sub-protocols involve the TTP. Without loss of generality we assume that the participant sends its message history (all

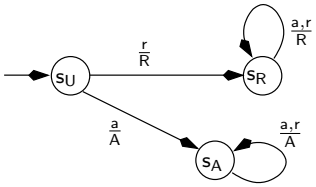

Fig. 2. Abstract Mealy machine of TTP

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 issued by the TTP is deemed adequate, cf. *weak fairness* [Pagnia et al. 2003].

The TTP logic matching the resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r}, \mathsf{r})$ for the protocol of figure 1 is shown in figure 2. 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 recovery 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.

2.5 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 properties in presence of any number of faulty processes [Asokan 1998]:

—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 (or R), 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 A obtains i_B and B obtains i_A, with no contact to the TTP.

Note that fairness as stated above is a safety trace property, and timeliness is a liveness trace property, cf. [Alpern and Schneider 1985]. Functionality is however not a trace property as it concerns existence of particular traces in the system.

Theorem 1 [Pfitzmann et al. 1998]. *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 [Pfitzmann et al. 1998] 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.*

Any secure protocol for exchanging non-idempotent items (between TDs) has to further guarantee the following requirement [Fujimura and Eastlake 2003]:

—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; cf. section 4.2.

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 [Chandy and Misra 1986], see also [Fagin et al. 1995].

3.1 A formal model for protocols

Given a 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 $x < y$ iff $x \leq y$ 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 that there exists a bijective function $\bar{\cdot} : S \to \bar{S}$ such that $\forall a \in S.\ \bar{a} \in \bar{S}$. 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 that 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. Note that in this model messages can get lost, duplicated and reordered in transmission.

3.2 A logic of knowledge

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 operator* $\mathcal{E}$, in order to define the syntax of our knowledge-based logic *EL*:

—Every element of Φ is an *EL* formula

—If e is an *EL* formula and p a process, then $\mathcal{E}_p(e)$ is an *EL* formula

—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".

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 [Chandy and Misra 1986]. Models of *EL* formulas are computations. For computation x and *EL* formula e, the relation $x \models e$ is inductively defined (modulo interpretation function $\mathcal{I}$) 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 $x \models e'$.

We remark that the semantics of *EL* can be defined using Kripke structures, e.g. as in [Fagin et al. 1995]. This is however unnecessary for our analysis. We have thus opted for the simpler setting of computations as models of *EL* formulas.

3.3 A formal model for fair exchange protocols using TDs

Fair exchange protocols between TDs constitute a particular class of protocols. In order to capture the features of this class, we place a number of constraints on the computations of the protocols, as described below. These constraints reflect and rely upon certain assumptions on the behaviors of TDs. It is straightforward that TDs can be programmed so that these constraints are realized in practice. In the following, TDs are referred to as processes.

Constraint 1. Any process p which finishes the optimistic sub-protocol successfully executes an internal action $\top_p$ and terminates:

$$\forall x \in p.\ \top_p \in x \implies \neg\exists y \in p.\ x < y$$

Recall that the set of computations of p is prefix closed.

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 R_p and A_p, with $p \in \mathcal{P}$, for receiving the recovery and abort tokens by p, respectively. Notation Q_p denotes p quiting the exchange.

Constraint 2. Any process p terminates immediately after executing any of the actions A_p, R_p and Q_p:

$$\forall x \in p.\ \mathsf{A}_p \in x \vee \mathsf{R}_p \in x \vee \mathsf{Q}_p \in x \implies \neg\exists y \in p.\ x < y$$

This assumption in particular implies that only one of A_p, R_p or Q_p events can appear in any execution x.

Constraint 3. When a process p crashes it simply executes $\perp_p$ and terminates:

$$\forall x \in p.\ \perp_p \in x \implies \neg\exists y \in p.\ x < y$$

Constraint 4. Any process p (except TTP) may crash at any moment (before terminating).

$$\forall x \in p.\ \top_p \notin x \wedge \perp_p \notin x \wedge \mathsf{A}_p \notin x \wedge \mathsf{R}_p \notin x \wedge \mathsf{Q}_p \notin x \implies x\perp_p \in p$$

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

—If R_p appears in x for process p, then 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.

—A computation x of protocol $\mathcal{P}$ is maximal iff for any $p \in \mathcal{P}$, for all $y \in p$ if $x_p < y$ and $\perp_p \notin y - x_p$, then $x(y - x_p)$ 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.

Constraint 5. By a computation, we mean a TTP-consistent and maximal computation, unless otherwise stated.

TTP-consistence and maximality can be seen as *fairness constraints* on protocol computations, cf. [Francez 1986]. Fairness constraints should of course not be confused with the fairness requirement in exchange protocols.

3.4 Fair exchange security requirements

Using the model defined above, we can formally define the security requirements of fair exchange protocols. 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 $\mathcal{P}$".

—Timeliness:

$$\forall x : \mathcal{P}.\forall p \in \mathcal{P}.\ \mathsf{Q}_p \in x \vee \mathsf{A}_p \in x \vee \mathsf{R}_p \in x \vee \top_p \in x \vee \perp_p \in x$$

—Fairness:

$$\forall x : \mathcal{P}.\forall p, q \in \mathcal{P}.\ (\mathsf{R}_p \in x \vee \top_p \in x) \implies (\mathsf{R}_q \in x \vee \top_q \in x \vee \perp_q \in x)$$

—Functionality:

$$\exists x : \mathcal{P}.\forall p \in \mathcal{P}.\ \top_p \in x$$

The following theorem is the technical conclusion of our formalization which relates fairness to knowledge. For a computation x and $p \in \mathcal{P}$, we fix the proposition $\mathbb{G}_p$ and the interpretation $\mathcal{I}$ so that $\mathcal{I}(x)\mathbb{G}_p = \mathsf{tt}$ 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 $x \models \mathcal{E}_p(\mathbb{G}_q \vee \bot_q)$.*

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 satisfies 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_p to V_q.

3.5 Three-message protocols for FE using TDs

In this section we characterize the resolve pattern of any three-message FE protocol that satisfies timeliness, fairness and functionality. Figure 3 shows a generic three message protocol. We focus 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 requirement $\pi_1 \notin \{\mathsf{r}, \mathsf{a}\}$ 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 quit 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 than $2\ell + 2$ 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.

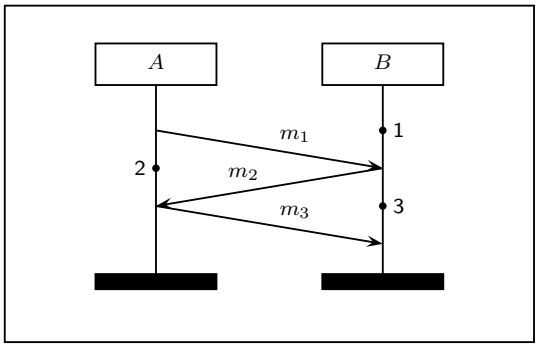

Fig. 3. Generic three message protocol.

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

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

PROOF. Immediate due to lemmas 1, 2 and 3, given below. □

In the following, by abusing the notation, 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: $x = x_1 \bot_p$ 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 timeliness, fairness 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 timeliness, fairness 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. Note that computations x and y are isomorphic w.r.t. B; this is because $x_B = y_B = \pi_1(B)$. Therefore, 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 that $\pi_2 \neq \mathsf{q}$. 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, $\pi_2 \in \{\mathsf{a}, \mathsf{r}\}$. Since $\pi_2 \in \{\mathsf{a}, \mathsf{q}\}$ (see above), we conclude $\pi_2 = \mathsf{a}$. □

The resolve pattern determined in theorem 3 is a necessary, but not generally sufficient, condition for satisfying fairness, timeliness and functionality. In particular, if processes are not trusted, then 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 a bounded 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 unbounded 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 a bounded 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 unbounded storage capacity.

4.1 A four-message FE protocol between TDs with bounded storage capacity

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 unbounded storage capacity. The attack is due to the no-duplication requirement.

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 a bounded 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 the new 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 of the messages is not guaranteed over reliable channels). Detecting replays would however require B to keep record of all its previous exchanges, which is not possible if B has a bounded 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 a bounded storage capacity. This resolve pattern is inspired by a protocol given in [Terada et al. 2006].

In order to give a detailed description of the protocol, we relax the 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 [Dolev and Yao 1983].

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 a bounded storage capacity.*

PROOF. Consider an instantiation of the four-message protocol of figure 1, and the TTP logic of figure 2, 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 are tenable assumptions since A and B are trusted devices. The specifications for initiator A and responder B are given in algorithms 1 and 2, respectively.

Algorithm 1 Specification of initiator A in theorem 4

```
V_A := V_A \ {v}
send m_1 to B
recv m_2 from B
if recv times out then
   V_A := V_A ∪ {v}
   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 ∪ {v}
   else if recv recovery token R from TTP then
      V_A := V_A ∪ {v'}
else
   V_A := V_A ∪ {v'}
```

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. 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.
—$m_3 := [h(n')]_A$
—$m_4 := n'$

Notice that in this protocol, v and v' are not secrets to A and B. The exchange is therefore meant to exchange the *permissions* to add v and v' to the local voucher sets of A and B, rather than exchanging v and v' themselves.

We assume that after 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: $\mathsf{a} := [f_1, A, B, v, v', n, h(n')]_B$, $\mathsf{r} := [f_2, A, B, v, v', n, h(n')]_A$, $\mathsf{A} :=$

$[ackf_1, A, B, v, v', n, h(n')]_{TTP}$, and $\mathsf{R} := [ackf_2, A, B, v, v', n, h(n')]_{TTP}$. Here f_1, f_2 and $ackf_1$ and $ackf_2$ are unique flags respectively denoting an abort request, a recovery request, an abort token, and a recovery token. Notice that in this protocol,

Algorithm 2 Specification of responder B in theorem 4

```
recv m_1 from A
if recv times out then
  quit
V_B := V_B \ {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 ∪ {v'}
  else if recv recovery token R from TTP then
    V_B := V_B ∪ {v}
else
  send m_4 to A
  V_B := V_B ∪ {v}
```

the TTP can readily extract v and v' from m_1 and m_2. As mentioned above, 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 brief security analysis of the protocol is given below.

—(timeliness) Note that whenever a participant is waiting for a message, it can unilaterally stop waiting, and contact the TTP. This entails timeliness.

—(fairness) It is crucial that B is a trusted device, and will not abort the exchange after receiving m_3. Therefore, even if the last message m_4 is intercepted, device A can run the recovery sub-protocol. Similarly, if A receive m_3 or B receives m_4 after completing the protocol using resolve sub-protocols, they will not act upon these due to assumption (2) in the beginning of the proof. It is also safe for A to quit the exchange if m_2 does not arrive in time, since B will afterward only abort the exchange.

—(functionality) With negligible delays, obviously correct A and B finish the exchange successfully with no TTP mediation.

—(no-duplication) Note that assumption (1) in the beginning of the 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.

This completes our proof. □

4.2 Four messages are necessary for FE between TDs with bounded storage capacity

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 a bounded 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 a bounded 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. Let computation x be a computation which has completed successfully without resorting to the TTP. Such a computation exists due to the functionality property. Let

$$x = m_1 \bar{m}_1 m_2 \bar{m}_2 m_3 \bar{m}_3 \top_p \top_q$$

Now, assume that the protocol is repeatedly executed between processes p and q. Since the processes have a bounded 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 for q: 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. □

Note 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 theoretically, in order to prevent repetition. That is, TDs require unbounded storage capacity to store the sequence numbers in general.

4.3 Three-message FE protocols between TDs with unbounded storage capacity

For exchanging non-idempotent items between TDs with unbounded storage capacity one can use a three-message FE protocol with resolve pattern $(\mathsf{q}, \mathsf{a}, \mathsf{r})$. The main idea of the protocol is that TDs can use their unbounded storage capacity 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 unbounded storage capacity.*

PROOF. Consider the three-message protocol of figure 3 with the TTP logic of figure 2 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 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

Algorithm 3 Specification of initiator A in theorem 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
  $V_A := V_A \cup \{v'\}$

they start the resolve sub-protocols, they will ignore all the messages from the main sub-protocol. These are tenable assumptions 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.

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

—$m_2 := [h(v, v', A, B, h(n))]_B$

—$m_3 := n$

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: $\mathsf{a} := [f_1, A, B, v, v', h(n)]_A$, $\mathsf{r} := [f_2, A, B, v, v', h(n)]_B$, $\mathsf{A} := [ackf_1, A, B, v, v', h(n)]_{TTP}$, and $\mathsf{R} := [ackf_2, A, B, v, v', h(n)]_{TTP}$. Here f_1, f_2 and $ackf_1$ and $ackf_2$ are unique flags respectively denoting an abort request, a recovery request, an abort token, and a recovery token.

A brief security analysis of the protocol is given below.

—(timeliness) Note that whenever a participant is waiting for a message, it can unilaterally stop waiting, and contact the TTP. This entails timeliness.

—(fairness) It is crucial that A is a trusted device, and will not abort the exchange after receiving m_2. Therefore, even if the last message m_3 is intercepted, device B can run the recovery sub-protocol. Similarly, if A or B receive m_2 or m_3 after completing the protocol via resolve sub-protocols, they will not act upon these according to assumption (2) in the beginning of the proof. It is also safe for B to quit the exchange if m_1 does not arrive in time, since A will afterward only abort the exchange.

Algorithm 4 Specification of responder B in theorem 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
    $V_B := V_B \cup \{v\}$

—(functionality) With negligible delays, obviously correct A and B finish the exchange successfully with no TTP mediation.

—(no-duplication) 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. Here the *history set* $\mathcal{H}_B$ in device B is used to prevent the replay attack described in theorem 5.

This completes the proof. □

REMARK 2. *Micali has proposed [Micali 2003] a three-message protocol for fair exchange of idempotent items between non-trusted devices, which has the resolve pattern* $(\mathsf{q}, -, \mathsf{r})$. *As observed in [Asokan 1998] and our theorem 3, timeliness is violated in Micali's protocol. The resolve pattern* $(\mathsf{q}, \mathsf{a}, \mathsf{r})$ *of theorem 6 has been used by [Schunter 2000] and [Vogt 2003] for fair exchange of* revocable *digital contents: Intuitively, TTP's testimony is necessary for the validity of the exchanged items in these protocols.*

5. EXCHANGING NON-IDEMPOTENT ITEMS USING TDS IN PRACTICE

The results presented in the previous sections consider non-idempotent items which do not lose their value. In practice, some non-idempotent items might be of only temporary value. Then, it is possible to devise heuristics to identify and eliminate the fingerprints of the previous exchanges which are obsolete, hence irrelevant, for the security of the system. Such "garbage collection" procedures can reduce the amount of required secure storage on trusted devices. For instance, let v be a one-time voucher, allowing the owner to attend a particular event at a particular time, viz. the value of v is nullified after the event. If such non-idempotent items are circulated in the system, then the no-duplication requirement (introduced in section 2) can be weakened, without posing any real threat to the system:

—No-duplication (revisited): The total number of instances of any non-idempotent item v, which has not been nullified, is never increased in the system.

Now, suppose the three message protocol of theorem 6 is deployed. Then, all TDs have to store the fingerprints of all their previous exchanges, in order to prevent replay attacks. Suppose once B spends v at the event, it is instructed to *delete* from its storage all the fingerprints which pertains to the executions in which v has been offered to B in exchange for some voucher $v_B \in V_B$. Now, if a replay attack is launched, v can be duplicated on B. More specifically, suppose A offers v to B in exchange for some $v_B \in V_B$. At the end of the exchange with a replay attack (as described in theorem 5), A will possess v_B and B will possess two instances of v. Nevertheless, the revisited no-duplication requirement is not violated, since v has already lost its value. Therefore, for vouchers with temporary value, replay attacks are not a real security threat to the system, once the vouchers have been nullified. There are however a number of security considerations pertinent to vouchers with temporary value:

—As trusted devices are typically operated by human owners, the owner needs to be able to validate the expiration date of the vouchers that are offered to their TD. This will prevent an attacker from tricking a device owner to accept nullified vouchers, after their expiration date. Usually, electronic vouchers carry meta data which specify information such as expiration dates [Fujimura and Eastlake 2003]. We remark that to detect expired vouchers, the clock of the TD and the voucher issuer should be (roughly) synchronized.

—Without proper security considerations, delete instructions can be used to enable replay attacks (and thus duplicate) valid vouchers in the system. The instruction to delete obsolete voucher fingerprints should therefore be authenticated in order to ensure that an authority, e.g. the voucher issuer, is the source of the instruction.

Note that the TTP need not be aware of nullified vouchers. If a TD wants, for some reason, to exchange a voucher for an obsolete one, the TTP should play its usual role. This reduces computational load of the TTP, since it does not need to check the list of obsolete vouchers for each dispute.

6. CONCLUDING REMARKS

We have analyzed the efficiency of optimistic protocols for fair exchange of idempotent and non-idempotent items using trusted devices. Four messages in the main sub-protocol is proved to be necessary for exchanging non-idempotent items, given that the trusted devices have access to a bounded storage capacity. With unbounded storage capacity, this number can however be reduced to three. For exchanging idempotent items between trusted devices, it is straightforward that three messages in the main sub-protocol are sufficient and necessary.

Investigating mechanisms for distributed detection of nullified non-idempotent items, as opposed to the centralized mechanism described in section 5, is a possible future research direction. It must also be interesting to explore the efficiency of FE protocols which guarantee *atomicity* for non-idempotent items, that is no-duplication and also no-destruction. Atomicity is a typical requirement for financial transactions [Tygar 1996].

ACKNOWLEDGMENTS

The work has been supported by AVANTSSAR, FP7-ICT-2007-1 project no. 216471.

REFERENCES

ALPERN, B. AND SCHNEIDER, F. 1985. Defining liveness. *Inf. Process. Lett. 21,* 4, 181–185.

ASOKAN, N. 1998. Fairness in electronic commerce. Ph.D. thesis, University of Waterloo.

CHANDY, K. AND MISRA, J. 1986. How processes learn. *Distrib. Comput. 1,* 1, 40–52.

CHOR, B., FIAT, A., NAOR, M., AND PINKAS, B. 2000. Tracing traitors. *IEEE Transactions on Information Theory 46,* 3, 893–910.

DOLEV, D. AND YAO, A. C. 1983. On the security of public key protocols. *IEEE Trans. on Information Theory IT-29,* 2, 198–208.

EVEN, S. AND YACOBI, Y. 1980. Relations among public key signature systems. Tech. Rep. 175, Computer Science Dept., Technion, Haifa, Isreal. March.

EZHILCHELVAN, P. D. AND SHRIVASTAVA, S. K. 2005. A family of trusted third party based fair-exchange protocols. *IEEE Trans. Dependable Secur. Comput. 2,* 4, 273–286.

FAGIN, R., HALPERN, J., MOSES, Y., AND VARDI, M. 1995. *Reasoning About Knowledge.* MIT Press, Cambridge, Mass.

FORT, M., FREILING, F., DRAQUE PENSO, L., BENENSON, Z., AND KESDOGAN, D. 2006. TrustedPals: Secure multiparty computation implemented with smart cards. In *ESORICS '06.* LNCS, vol. 4189. Springer, 34–48.

FRANCEZ, N. 1986. *Fairness.* Springer.

FUJIMURA, K. AND EASTLAKE, D. 2003. Requirements and Design for Voucher Trading System (VTS). RFC 3506.

FUJIMURA, K., KUNO, H., TERADA, M., MATSUYAMA, K., MIZUNO, Y., AND SEKINE, J. 1999. Digital-ticket-controlled digital ticket circulation. In *USENIX Security '99.* 229–240.

GRAY, J. 1978. Notes on data base operating systems. In *Operating Systems, An Advanced Course.* LNCS, vol. 60. Springer, 393–481.

GUERRAOUI, R. AND RODRIGUES, L. 2006. *Introduction to Reliable Distributed Programming.* Springer.

KUNTZE, N. AND SCHMIDT, A. 2007. Trusted ticket systems and applications. In *IFIP SEC '07.* IFIP, vol. 232. Springer, 49–60.

MAUW, S., RADOMIROVIC, S., AND TORABI DASHTI, M. 2009. Minimal message complexity of asynchronous multi-party contract signing. In *CSF '09.* IEEE CS, 13–25.

MICALI, S. 2003. Simple and fast optimistic protocols for fair electronic exchange. In *PODC '03.* ACM Press, 12–19.

PAGNIA, H., VOGT, H., AND GÄRTNER, F. C. 2003. Fair exchange. *Computer Journal 46,* 1, 55–7.

PFITZMANN, B., SCHUNTER, M., AND WAIDNER, M. 1998. Optimal efficiency of optimistic contract signing. In *PODC '98.* ACM Press, 113–122.

SCHUNTER, M. 2000. Optimistic fair exchange. Ph.D. thesis, Universität des Saarlandes.

TERADA, M., MORI, K., ISHII, K., HONGO, S., USAKA, T., KOSHIZUKA, N., AND SAKAMURA, K. 2006. A framework for distributed inter-smartcard communication. *IPSJ Digital Courier 2,* 120–132.

TORABI DASHTI, M. AND MAUW, S. 2010. *Handbook of Financial Cryptography and Security (G. Rosenberg, ed.).* Chapmand and Hall/CRC, Chapter Fair Exchange, 109–132.

TORABI DASHTI, M., NAIR, S. K., AND JONKER, H. 2008. Nuovo DRM Paradiso: Designing a secure, verified, fair exchange DRM scheme. *Fundam. Inform. 89,* 4, 393–417.

TYGAR, J. 1996. Atomicity in electronic commerce. In *PODC '96.* ACM press, 8–26.

VOGT, H. 2003. Asynchronous optimistic fair exchange based on revocable items. In *Financial Cryptography.* LNCS, vol. 2742. Springer, 208–222.

VOGT, H., PAGNIA, H., AND GÄRTNER, F. C. 2001. Using smart cards for fair exchange. In *Electronic Commerce '01.* LNCS, vol. 2232. Springer, 101–113.