Nuovo DRM Paradiso: Formal specification and verification of a DRM protocol

H. Jonker, S. Krishnan Nair, M. Torabi Dashti

Technische Universiteit Eindhoven, Vrije Universiteit Amsterdam, CWI Amsterdam
h.l.jonker@tue.nl, srijith@few.vu.nl, dashti@cwi.nl

Abstract. We present a DRM-preserving content redistribution scheme, based on the NPGCT scheme [15], that provides fairness in unsupervised exchanges. The proposed scheme is formally specified, verified and shown to achieve its design goals. The NPGCT mechanism of detection and revocation of circumvented devices is also reexamined here.
Keywords: DRM, formal verification, optimistic fair exchange protocols.

1 Introduction

In recent years, there has been a rapid increase in the popularity of personal devices able to render digital contents, some of them also having peer-to-peer communication facilities. With the advent of these personal digital devices, mainly aimed at the entertainment market, content providers are looking at various secure business models to tap into this growing market for selling their copyrighted materials. It naturally necessitates a mechanism to protect digital contents from illegal access and unauthorized distribution. Technologies used to enforce these predefined policies controlling access to digital contents are referred to as Digital Rights Management (DRM) techniques. The biggest challenge in DRM is to enforce DRM policies after contents have been distributed to consumers. The current approach to solve this problem is to limit the distribution of protected contents to only so-called *compliant* devices, which by construction are guaranteed to always observe the DRM policies associated with the digital contents they render.

A unique concept of DRM-preserving content *redistribution* was proposed in [15], where users also act as content distributors. It potentially allows a business model where a consumer can not only buy the rights to use a content, but also redistribute or resell the content in a controlled manner. From a security point of view, this is technically challenging, because the resulting system is a network of peer-to-peer independent devices, each of them a potential consumer, authorized distributer, and also an attacker. This paper uses the concepts presented in that work as a basis for a refined protocol, where payment orders are exchanged for protected contents.

Contributions Our contributions in this paper are as follows. We demonstrate some security issues with the NPGCT scheme proposed in [15], which are mainly due to underspecification. These issues are addressed by an exact formal specification based on the scheme, which is then simplified. The mechanism for detecting and revoking compromised devices in NPGCT is also revisited here and the underlying trade-offs between security, flexibility and communication costs are highlighted. Next, the exchange protocol for peer customers is augmented with a light-weight

recovery sub-protocol, which proves to be essential in preventing unfair situations in the exchange, as customers do not necessarily trust each other. The resulting scheme can then be seen as an *optimistic fair exchange* DRM scheme. Optimistic fair exchange protocols, e.g. see [2], guarantee fairness in exchange (as in Section 3.1) with the help of an off-line trusted third party. Here, due to the use of trusted computing devices, *strong* fairness [2] is achieved so that a suffered party can recover within the system without further communications with the other party. Finally, the refined scheme and its requirements for effectiveness, content secrecy, exchange fairness and resistance to content masquerading are formalized and a formal verification of the requirements is performed on a finite model of the scheme. The proposed protocols are shown to indeed meet the stated requirements.

Related work A fair exchange protocol for trading electronic vouchers is proposed by Terada et al. in [19]. It is similar to our work in that it relies on trusted computing devices to achieve strong fairness. It has however certain assumptions which make it not directly applicable for DRM scenarios. For instance, the protocol is not secure if the exchanged items are similar. In a related study Avoine et al. reduce the problem of fair exchange between trusted devices owned by malicious agents to the classic atomic commit problem [3]. They present a probabilistic fair exchange protocol such that the error rate is not controlled by untrusted agents.

In this paper we do not explicitly model handling and derivation of rights associated with DRM-protected contents. It would, in principle, require defining a formal syntax and semantic for the rights and an exact specification of the ways agents handle them. We refer the interested reader to the related literature, such as [17].

Structure of the paper In Section 2 we summarize the NPGCT scheme, which provides the basis for the refined scheme. Section 3 presents our refined scheme, dubbed "Nuovo DRM", and its goals. Nuovo DRM is then formally analyzed in Section 4 and shown to achieve its goals. Section 5 revisits the mechanisms for detecting and revoking compromised devices. Finally, Section 6 concludes the paper.

2 The NPGCT DRM scheme

The NPGCT scheme [15] was proposed as a DRM-preserving digital content redistribution system where a consumer can also act as a content reseller. The scheme consists of two main protocols: the first distributes contents from provider P to client C, the second allows C to resell contents to another client D.

We use the following notations for the rest of the paper:

- C and D are compliant devices, owned respectively by $owner(C)$ and $owner(D)$.
- P is a legitimate trusted content provider.
- h is an ideal one way hash function; therefore $h(x)$ uniquely describes x.
- $PK(X), SK(X)$ are respectively the public and private keys of entity X.
- $\{m\}_K$ represents an ideal[1] encryption of message m with key K. Encrypting with a private key denotes signing.
- The content of a signed message is always carried along with its signature. In other words, instead of the message $(M, \{M\}_{SK(X)})$, we write $\{M\}_{SK(X)}$ to save some space.

[1] As in [9], we assume perfect cryptographic functions.

– n_X is a nonce which is freshly generated by entity X at its first appearance in the protocol.
– M represents a protected content. The set of all protected contents in the protocol is denoted *Cont*. It is assumed that unique descriptions (such as hash values) of all $M \in Cont$ are publicly known.
– The set of all possible rights is denoted *Rgts*. The term $R_X(M)$ represents the rights of agent X for content M.

Provider-customer protocol (P2C) The protocol is initiated by the owner of C who wants to buy item M with rights R from provider P. From [15]:

1. $C \rightarrow P$: Request content
2. $C \leftrightarrow P$: Mutual authentication, [payment]
3. $P \rightarrow C$: $\{M\}_K, \{K\}_{PK(C)}, R, \sigma, \Lambda$
   σ=meta-data of M, Λ=$\{h(P, C, M, \sigma, R)\}_{PK(P)}$

Λ certifies that C has been granted rights R and helps in proving C's right to redistribute M to other clients. It also binds the meta-data σ to the content, which prevents spoofing attacks on M.

Customer-customer protocol (C2C) This part of the protocol is initiated by the owner of D who wants to buy M with rights R' from C. From [15]:

1. $D \rightarrow C$: Request content
2. $C \leftrightarrow D$: Mutual authentication
3. $C \rightarrow D$: $\{M\}_{K'}, \{K'\}_{PK(D)}, R_C(M), R', \sigma, \Lambda, \Lambda'$
   $\Lambda' = \{h(C, D, M, \sigma, R')\}_{PK(C)}$
4. D : Verifies σ, Λ' and $R_C(M)$ using Λ
5. $D \rightarrow C$: ψ, [*payment*]
   $\psi = \{h(C, P, \{M\}_{K'}, \sigma, R')\}_{PK(D)}$

ψ acts as an acknowledgment from D that it has received the content M with the rights $R_D(M)$, while Λ and Λ' form a chain and serve the same purpose Λ serves for C, i.e. proving that D has been granted rights R'. If R' includes the right to further redistribute, D can use Λ and Λ' in the same way C used Λ.

2.1 Security issues of NPGCT

We observe the following two attacks on the NPGCT scheme. First, in the P2C protocol (and similarly in the C2C protocol), a malicious customer can feed rights from a previous session to its trusted device, because the authentication phase is not extended to guarantee the freshness of the content-right bundle which is subsequently delivered. As a remediation, fresh nonces from the authentication phase can be used in Λ to ensure the freshness of the whole exchange. This remediation is detailed further in Section 3.

Second, in the C2C protocol, payment is not bound to the request/receive messages exchanged between two customers. Therefore, once D receives M in step 3, the owner of D can avoid paying C by aborting the protocol. Since the C2C protocol is unsupervised, the owners of compliant devices are forced to trust each other to complete transactions. While it seems reasonable to extend such trust to a legitimate content provider, it should not be assumed for the C2C exchanges.

3 The Nuovo DRM scheme

The main contribution of the NPGCT scheme is to enable redistributing contents between compliant devices, while preserving DRM. This section describes a refined version of NPGCT, which addresses the security concerns noted in the previous section.

The main differences between Nuovo DRM and NPGCT are the following: First, both attacks on NPGCT explained in the previous section are addressed. The explicit authentication in the P2C and C2C protocols in Nuovo prevents the replay attack on the NPGCT; to prevent monetary loss due to early abortion, Nuovo's C2C protocol provides a recovery sub-protocol.

Second, the payment scheme is left open in the NPGCT, so various on-line (requiring contact with the bank, e.g. payment using ATM cards) and off-line (not requiring contact with the bank, e.g. payment by cash) payment systems can fit there. In contrast, Nuovo DRM protocol is limited to on-line payment schemes, where payment orders exchanged for contents can be encashed only at a bank entity. As a direct benefit of assuming such a payment system, the process of chaining of rights (see Section 2), which is used in [15] to detect circumvented devices, is no longer needed in Nuovo DRM as this can be achieved by inspecting payment orders (see Section 5). It can be seen as a trade-off between flexible payment schemes and communication costs in the protocol.

The following assumptions are made in the protocol (and its subsequent specification and verification):

A1 Consumer devices (C and D) are compliant and therefore can be seen as trusted for our purpose. Notably, compliant devices are able to perform atomic actions.

A2 A legitimate trusted content provider is assumed not to engage in C2C exchanges and is impartial in its behavior. It uses a persistent and secure database and its actions on the database are atomic. It thus always provides consistent responses.

A3 There is a hierarchy of public keys, with the public key of the root embedded in each compliant device and available to content providers. Using such an infrastructure, each participant can prove its identity or verify another's authentication without requiring contact with a trusted party.

A4 The banking system (responsible for encashing payment orders) cooperates with content providers[2] to catch malevolent users. Here, for the sake of simplicity, the content provider and the bank are the same entity.

A5 When a compliant device signs a payment order, the payment order is cash-able. This can be accomplished, for instance, by providing each compliant device with some credit, which can be spent and recharged.

A6 Price negotiations are outside the scope of Nuovo DRM. We assume that the price of content is negotiated in advance by the participants. In Nuovo DRM, the price of the content being traded is bundled with the requested rights.

Provider-customer protocol (P2C) The owner of C wants to buy item M with rights R from content provider P.

[2] Though this assumption may not be universally acceptable, e.g. due to geographical diversity of content providers and banking systems used by customers, the required degree of collaboration makes the assumption practically tenable.

1. $owner(C) \rightarrow C : P,\ h(M),\ R$
2. $C \rightarrow P : C,\ n_C$
3. $P \rightarrow C : \{n_P,\ n_C,\ C\}_{SK(P)}$
4. $C \rightarrow P : \{n_C,\ n_P,\ h(M),\ R,\ P\}_{SK(C)}$
5. $P \rightarrow C : \{M\}_K,\ \{K\}_{PK(C)},\ \{R,\ n_C\}_{SK(P)}$

In the first step, the hash of the desired content, possibly retrieved from a trusted public directory, with a right and the identity of a legitimate provider is provided to the compliant device C. Following assumption A6, $owner(C)$ and P have already reached an agreement on the price. Whether P is a legitimate provider can be checked by C (assumption A3) and vice versa. In step 2, C generates a fresh nonce n_C and sends it to P, who will continue the protocol only if C is a compliant device. Step 3 completes mutual authentication[3] between C and P (based on the ISO/IEC 9798-3 three-pass mutual authentication protocol, see e.g. [5]). Message 4 constitutes a payment order from C to P. After receiving this message, P checks if R was previously agreed upon (assumption A6) and only if so, it stores the payment order (for future/immediate encashing) and performs step 5 after generating a random key K. When C receives message 5, it decrypts $\{K\}_{PK(C)}$, extracts M and checks if it matches $h(M)$ in message 1, and n_C is the same as the nonce in message 2. If these tests pass, C updates $R_C(M)$ with R.

We define a set of *abstract* actions merely to highlight important steps of the protocol. These actions are used in the formalization to define desired (or undesired) behaviors of the protocol. For the P2C protocol, C performs the abstract action $request(C, h(M), R, P)$ at step 4, P performs $issue(P, h(M), R, C)$ at step 5 and finally C performs $update(C, h(M), R, P)$ upon accepting message 5.

Customer-customer protocol (C2C) The owner of D wants to buy item M with rights R' from reseller C.

1. $owner(D) \rightarrow D : C,\ h(M),\ R'$
2. $D \rightarrow C : D,\ n_D$
3. $C \rightarrow D : \{n_C,\ n_D,\ D\}_{SK(C)}$
4. $D \rightarrow C : \{n_D,\ n_C,\ h(M),\ R',\ C\}_{SK(D)}$
5. $C \rightarrow D : \{M\}_K,\ \{K\}_{PK(D)},\ \{R',\ n_D\}_{SK(C)}$

At step 4, D performs the abstract action $request(D, h(M), R', C)$. At step 5, C performs $issue(C, h(M), R', D)$ and more importantly, atomicly updates the right associated with M (reflecting that some part of $R_C(M)$ has been used for reselling M) and stores the payment order signed by D. Note that the atomicity of these actions is important as it guarantees that C does not store the payment order without simultaneously updating the right $R_C(M)$. Upon accepting message 5, D performs $update(D, h(M), R', C)$. This protocol is similar to the provider-customer protocol and its detailed description is skipped here. We however remark that checking for compliance is not as easy as in the provider-customer protocol, mainly because of revoked devices. This issue is examined in Section 5 in more detail.

[3] One could argue that since C is a trusted computing device, P does not need to check the freshness of the communication. However, in this case P would need to remember nonces to prevent replaying payment orders (message 4). It can thus be thought as a trade-off, stateless providers with mutual authentication vs. stateful providers and one-way authentication.

As mentioned earlier, a malicious $owner(C)$ can abort before delivering message 5 to D or this message can simply be lost because of a hardware failure. To prevent such unfair situations (for D) we provide a recovery mechanism to obtain the lost content.

Recovery sub-protocol In case of a failure in communicating message 5 in the C2C protocol, D can start a recovery sub-protocol with the content provider. The purpose of the recovery is to assure that D receives the content and rights that $owner(D)$ wished (and ostensibly paid for). The owner of D can initiate the following protocol at any time after D has sent message 4 in the customer-customer protocol and a connection with the provider becomes available.

$$
\begin{array}{ll}
5^r. & owner(D) \rightarrow D : resolves(D) \\
6^r. & D \rightarrow P : D,\ n'_D \\
7^r. & P \rightarrow D : \{n_P,\ n'_D,\ D\}_{SK(P)} \\
8^r. & D \rightarrow P : \{n'_D,\ n_P,\ \langle n_D,\ n_C,\ h(M),\ R',\ C\rangle,\ P\}_{SK(D)} \\
9^r. & P \rightarrow D : \{M\}_K,\ \{K\}_{PK(D)},\ \{R',\ n'_D\}_{SK(P)}
\end{array}
$$

The abstract action $request(D, h(M), R', P)$ is performed by D at step 8^r. At step 9^r, P performs $issue(P, h(M), R', D)$. Upon accepting message 9^r, D performs the action $update(D, h(M), R', P)$. This protocol is similar to the provider-customer protocol and its detailed description is skipped here, only the behavior of P in resolving failed exchanges is explained below.

Assume that D tries to resolve an unsuccessful exchange with C. As a result of the atomicity of C's actions in customer-customer protocol, only the following situations are possible: either C has updated $R_C(M)$ and has the payment order of message 4 (which it is thus entitled to have), or C does not have the payment order and has not updated $R_C(M)$. In the latter case, the combination of the failed customer-customer protocol and its subsequent resolve simply boils down to a provider-customer exchange. However, when C owns the payment order from D, two different cases can happen: (1) If C tries to encash the payment order after D has resolved, P is the one who pays the money to C, as D has already paid P. Note that P is not paid for sending M to D, which is fair because it has already been paid by C when C received a right to resell M. (2) If D resolves after C has encashed the payment order, P will not charge D, because D has already paid the price and C has updated the right $R_C(M)$, for which it has already (directly or indirectly) paid P.

One can argue that the recovery sub-protocol may also fail due to lossy communication channels. As a way to mitigate it, one can build a more persistent communication channel for content providers e.g. by using an FTP server as an intermediary. The provider would upload the content, and the device would download it from the server. As the content is encrypted, no special access control on the server would be needed. Such resilient communication channels are generally necessary to guarantee fairness [2]. See Section 4.2 for related discussions.

As a final note, we emphasize that we only consider compliant devices (assumption A1) here and this protocol can be trivially attacked if the devices are tampered. Section 5 discusses methods for revoking tampered devices and resisting systematic content pirating attacks.

3.1 Nuovo DRM's goals

The NPGCT scheme was designed to resist against certain types of attacks. From the original requirements, providing *secrecy* and resisting *content masquerading* are relevant to our formalization[4]. Adding sub-protocols for providing fairness in exchange, however, means that Nuovo DRM protocol has to also provide *effectiveness* and *fairness*.

- **Secrecy** states that no outsider may access *secret* items, which are usually encrypted for intended receivers. NPGCT scheme and consequently Nuovo DRM scheme manage the distribution of DRM-protected contents by encrypting them for intended compliant devices. These schemes thus have to guarantee that a DRM-protected content never appears unencrypted in the protocol runs.
- **Content masquerading** occurs when content M is being passed off as content M', for $M \neq M'$. Preventing this ensures that a malicious participant cannot feed M' to a device which has requested M.
- **Effectiveness** states that if honest participants run a protocol, the protocol terminates successfully. Effectiveness is a sanity check for the functionality of the protocol, and may thus be checked in a system without an attacker.
- **Fairness** in exchange informally means that either all or none of the involved parties receive a desired item in exchange for their own. It has been shown that fair exchange is impossible without a trusted third party [16]. In our protocol, the content provider plays the role of a trusted third party (TTP) for C2C exchanges. Moreover, since the content provider can provide the exact missing content, strong fairness can be achieved in the system. As this property is a liveness property[5], a resilience assumption on the communication channels is required for fairness to hold in general. For an in-depth discussion of fairness, strong fairness and resilient communication channels, we refer readers to [2].

4 Formal analysis

Formal verification techniques can bring to light issues that have been overlooked by protocol designers and provide a certain level of confidence in the correctness and security of protocols [18] In this section we briefly describe the necessary machinery to formally verify whether Nuovo DRM achieves its design goals. This formal verification can be seen as a sequence of steps: specify the protocol, model the intruder, state the desired properties and verify the protocol. To this end, the Nuovo DRM scheme is formalized in μCRL [11], a process algebraic language (briefly explained in appendix A). μCRL, an extension of ACP [4] with abstract data types, is a language for specifying distributed systems and protocols in an algebraic style. A μCRL specification describes a labeled transition system, in which states represent process terms and edges are labeled with actions.

[4] The rest of the requirements in [15] concern mainly tracking circumvented devices and possible attacks on the storage backup systems, which are out of the scope of our analysis.

[5] Properties of systems can be divided into two classes: *safety* properties, stating unwanted situations do not happen, and *liveness* properties, stipulating desired events eventually happen. For a formal definition of these property classes see [1].

In our modeling approach, each participant of a protocol is seen as a process. The intruder, also specified as a process, is part of the environment and may also have a legitimate role in the protocol. Communications between protocols' participants happen via a network, which is controlled by the intruder. (For a complete specification of the Nuovo DRM scheme in μCRL see appendix B.)

In the analysis technique used, the intruder is allowed to have access to unbounded resources of data (like fresh nonces), should it need those to exploit a protocol. Our analysis method is fully automatic and the verification algorithms do not need human intervention. Here we consider only a (small) finite number of concurrent sessions of the protocol (see Section 4.4). Although this generally cannot constitute a theoretical proof of correctness or security for a protocol, in many practical situations it suffices. The fact that the problem of the security of cryptographic protocols is not decidable (e.g. see [8]) implies that a trade-off has to be made between completeness of the proofs and their automation. Similar to [9], we do not consider attacks resulting from weaknesses of the cryptographic functions used in the protocol. Type flaw attacks[6] are omitted from our analysis. They can however be easily prevented [12].

The communication model used here is further detailed in Section 4.1. The characteristic features of the intruder model used in this study are described in Section 4.2. The requirements of the protocol are then expressed in the regular alternation-free μ-calculus [13], which is sketched in Section 4.3. The model checker EVALUATOR 3.0 [13] from the CADP toolset [10], capable of efficiently checking regular alternation-free μ-calculus formulas, is used here to verify the stated properties of the protocol. The formalization of these properties and the results of the verification are presented in Section 4.4.

4.1 Communication model

We consider two different communication models. The first model is used to verify the effectiveness property. In this model there is no intruder and all participants are honest. A process A can send a message m to B (denoted by $send(A, m, B)$) only if B at the same time is in a position to receives it from A (denoted by $recv(A, m, B)$). The synchronization between $send(A, m, B)$ and $recv(A, m, B)$ actions is denoted $com(A, m, B)$, which formalizes the "$A \rightarrow B : m$" notation of Sections 2 and 3.

In order to verify the remaining properties, a model where the intruder has complete control over the communication network is used. When a process A sends a message m with the intention that it should be received by B, it is in fact the intruder that receives it, and it is only from the intruder that B may receive m. The send and receive actions (between the intruder and other participants) are again synchronized.

[6] A type-flaw attack happens when a field in a message that was originally intended to have one type is interpreted as having another type.

4.2 Intruder model

We mainly follow Dolev and Yao's approach to model the intruder [9]. The Dolev-Yao intruder has complete control over the network. It intercepts and remembers all messages that have been transmitted. It can decrypt and sign messages, if it knows the corresponding key. It can compose and send new messages from its knowledge. It can also remove or delay messages in favor of others being communicated. In this model, cryptographic primitives, such as signing and hashing, are assumed ideal. Under certain assumptions such as perfect cryptography, the Dolev-Yao intruder has been shown to be the most powerful attacker model [7]. In our formalization, this intruder can be seen as a non-deterministic process that blindly exhausts all possible sequences of actions, and then checks the effect of these on the protocol by exploring the resulting state space.

The model of intruder used here is however different from the standard Dolev-Yao intruder in certain aspects (for its formal specification see appendix A). These differences root in two characteristics of the protocol analyzed here and its requirements, as is described below.

Trusted devices play a crucial role in this protocol. They limit the power of the intruder significantly[7]. However, the intruder has the ability to turn its (trusted) devices off deliberately. This is a threat to fairness in exchange, because the devices will deviate from the protocol description. This ability has been implemented in our model by letting the intruder's device(s) non-deterministically select between continuing the protocol and aborting communications at each step.

Resilient networks are generally required to guarantee liveness of protocols. No liveness property can be proved in the Dolev-Yao model, since this intruder can simply block all communications (e.g. see [14]). To verify fair exchange properties, resilient communication channels (abbreviated as *RCC*) are often assumed. These guarantee that all transmitted messages will *eventually* reach their destination, provided there is a recipient for them [2]. The behavior of our intruder model is limited by *RCC*, i.e. it cannot indefinitely block the network[8]. Since the intruder is a non-deterministic process in our model, that part of the behavior that violates *RCC* has to be purged afterward. The action $com^{\dagger}$, used in Section 4.4, represents communication actions not required by RCC. This action is used to purge unnecessary collaborative behaviors of the intruder. A formalization of an intruder model that respects *RCC* is given in [6].

To indicate violation of the secrecy requirement, the intruder process performs the abstract action *revealed* (see appendix A) when it gets access to a plain (non-encrypted) version of any DRM-protected content. This action clearly does not indicate the case that the intruder can only render an item using its trusted device, which is a normal behavior in the system.

[7] For the moment we dismiss the possibility of tampering a trusted devices. Countermeasures for such cases are discussed in Section 5.

[8] E.g. a wireless channel provides *RCC* for mobile devices, assuming that jamming can only be locally sustained.

4.3 Regular alternation-free μ-calculus

Here we briefly sketch the regular alternation-free μ-calculus that is used to express the properties from Section 3.1. For a complete treatment of its syntax and semantics see [13]. This logic consists of *regular formulas* and *state formulas*. Regular formulas, which describe sets of traces, are built upon *action formulas* and the standard regular expression operators. We use '.', '$\vee$', '$\neg$' and '*' for concatenation, choice, complement and transitive-reflexive closure, respectively, of regular formulas. State formulas, expressing properties of states, are built upon propositional variables, standard boolean operators, the possibility modal operator $\langle \cdots \rangle$ ($\Diamond$ in modal logics), the necessity modal operator $[\cdots]$ ($\Box$ in modal logics) and the minimal and maximal fixed point operators μ and ν. The state formulas are assumed to be "alternation-free", which intuitively means that alternation of minimal and maximal fixed points is prohibited. The symbols F and T are used in both action formulas and state formulas. In action formulas they represent *no action* and *any action* and in state formulas they denote the empty set and the entire state space, respectively. The wildcard action parameter '$-$' represents any parameter of an action.

4.4 Analysis results

In this section we describe the results obtained from the formal analysis of the Nuovo DRM scheme. The formal analysis considers two scenarios. The first verifies effectiveness using the first communication model of Section 4.1. The second scenario uses the second communication model to verify the remaining properties. Both scenarios consist of two compliant devices (C and D) which are controlled by an intruder I. P denotes the legitimate content provider. The desired properties are encoded in the regular alternation μ-calculus.

Honest scenario S_0: The communication network is assumed operational and no malicious agent is present. C is ordered to buy an item from P. Subsequently C resells the purchased item to D.

This scenario was model-checked using the CADP toolset, confirming that it is deadlock-free, and effective as specified below.

Result 1 *Nuovo DRM is effective for scenario S_0, meaning that it satisfies the following properties:*

1. *Each request is eventually responded.*

$$\begin{aligned}\forall m \in \mathit{Cont},\ r \in \mathit{Rgts}.\ & [\mathsf{T}^*.\mathit{request}(C,m,r,P)]\ \mu X.(\langle \mathsf{T}\rangle \mathsf{T}\ \wedge\ [\neg \mathit{update}(C,m,r,P)]X)\\ & \wedge\\ & [\mathsf{T}^*.\mathit{request}(D,m,r,C)]\ \mu X.(\langle \mathsf{T}\rangle \mathsf{T}\ \wedge\ [\neg \mathit{update}(D,m,r,C)]X)\end{aligned}$$

2. *Each received item is preceded by its payment.*

$$\begin{aligned}&\forall m \in \mathit{Cont},\ r \in \mathit{Rgts}.\\ &[(\neg \mathit{paid}(P,m,r,C))^*.\mathit{update}(C,m,r,P)]\mathsf{F}\ \wedge\ [(\neg \mathit{paid}(C,m,r,D))^*.\mathit{update}(D,m,r,C)]\mathsf{F}\end{aligned}$$

Dishonest scenario S_1: The intruder controls the communication network and is the owner of the compliant devices C and D. The intruder can instruct the compliant devices to purchase items from the provider P, exchange items between

themselves and resolve a pending transaction. Moreover, the compliant device C can non-deterministically choose between following or aborting the protocol at each step, which models the ability of the intruder to turn a compliant device off (see Section 4.2). We model three concurrent runs of the content provider P, and three sequential runs of each of C and D. The resulting model was checked with the CADP toolset and the following results were proven.

Result 2 *Nuovo DRM provides secrecy in scenario S_1, i.e. no protected content is revealed to any non-compliant device (in this case: the intruder).*

$$\forall m : Cont.\ [\mathsf{T}^*.revealed(m)]\mathsf{F}$$

Result 3 *Nuovo DRM provides fairness in S_1 for P, i.e. no compliant device receives a protected content, unless the corresponding payment has already been made to P.*

$$\begin{aligned}\forall a \in \{C, D\},\ m \in Cont,\ r \in Rgts.\ & [(\neg issue(P, m, r, a))^*.update(a, m, r, P)]\mathsf{F} \\ & \wedge \\ & [\mathsf{T}^*.update(a, m, r, P).(\neg issue(P, m, r, a))^*.update(a, m, r, P)]\mathsf{F}\end{aligned}$$

Result 4 *Nuovo DRM provides strong fairness in S_1 for D, as formalized below*[9]*:*

1. *As a customer: If a compliant device pays (a provider or reseller device) for a content, it will eventually receive it.*
   Note that there are only finitely many TTPs available in the model, so the intruder, in principle, can keep all of them busy, preventing other participants from resolving their pending transactions. It corresponds to a denial of service attack in practice, which can be mitigated by putting time limits on transactions with TTPs. As we abstract away from timing aspects here, instead the action $last_{ttp}$ is used to indicate that all the available TTPs in the model are exhausted by the intruder. In other words, if this action does not happen, there is at least one TTP available for honest participants.

$$\begin{aligned}\forall m \in Cont,\ r \in Rgts.\ & [\mathsf{T}^*.request(D, m, r, C).(\neg(resolves(D) \vee update(D, m, r, C)))^*] \\ & < (\neg com^\dagger(-, -, -))^*.(resolves(D) \vee update(D, m, r, C)) > \mathsf{T} \\ & \wedge \\ & [(\neg last_{ttp})^*.request(D, m, r, C).(\neg(update(D, m, r, P) \vee last_{ttp}))^*. \\ & resolves(D).(\neg(update(D, m, r, P) \vee last_{ttp}))^*] \\ & < (\neg com^\dagger(-, -, -))^*.update(D, m, r, P) > \mathsf{T}\end{aligned}$$

2. *As a reseller: no compliant device receives a protected content from a reseller device, unless the corresponding payment has already been made to the reseller.*

$$\begin{aligned}\forall m \in Cont,\ r \in Rgts.\ & [(\neg issue(D, m, r, C))^*.update(C, m, r, D)]\mathsf{F} \\ & \wedge \\ & [\mathsf{T}^*.update(C, m, r, D).(\neg issue(D, m, r, C))^*.update(C, m, r, D)]\mathsf{F}\end{aligned}$$

[9] Fairness for C is not guaranteed here, because it may quit the protocol prematurely. A protocol, in principle, guarantees security only for the participants that follow the protocol.

Result 5 *Nuovo DRM resists content masquerading attacks in S_1, ensuring that compliant devices only receive content which they requested.*

$$\forall a \in \{C, D\},\ m \in Cont,\ r \in Rgts.\ [(\neg request(C, m, r, D))^*.update(C, m, r, D)]\mathsf{F} \wedge [(\neg request(D, m, r, C))^*.update(D, m, r, C)]\mathsf{F} \wedge [(\neg request(a, m, r, P))^*.update(a, m, r, P)]\mathsf{F}.$$

Additionally, the intruder cannot feed intruder-constructed content m_0 to compliant devices:

$$\forall a \in \{C, D\},\ r \in Rgts.\ [\mathsf{T}^*.update(C, m_0, r, D)]\mathsf{F} \wedge [\mathsf{T}^*.update(D, m_0, r, C)]\mathsf{F} \wedge [\mathsf{T}^*.update(a, m_0, r, P)]\mathsf{F}.$$

5 Detection and revocation of compromised devices

The security of Nuovo DRM hinges on the compliance of the user devices. However, it is reasonable to assume that over time, some of these devices will be compromised. In this section, we examine how to detect compromised devices and propose a method to limit interactions with these devices. As in [15], the proposed mechanism aims at detecting powerful attackers and systematic content pirating, rather than occasionally misbehaving users.

A compromised device can perform two obvious attacks [10]. First, it can overuse its reselling rights. To detect large scale overselling, the provider reconstructs the the chain of sold rights. This is possible because of assumption A4 – to acquire payment for sold rights, devices need to contact the provider.

In detail: the provider maintains a directed weighted graph $G = (V, E)$ for each sold content-right combination. Each node of G represents a compliant device and the function $E : V \times V \to Nat$, where Nat denotes natural numbers, represents right transfers between each two compliant devices. For each $v \in V$, *weight difference* is defined as $\Delta(v) = \sum_{v' \in V} E(v, v') - \sum_{v' \in V} E(v', v)$ (outgoing weight minus incoming weight). We define $U \subseteq V$, the set of nodes which has sold a bundle, but has not yet encashed it. We argue that if v_c is a compromised device which engages in large scale overselling, after a reasonable amount of time, the provider will detect v_c's behavior by noting that the weight difference of v_c plus the number of yet-to-cash rights are positive, i.e. $\Delta(v_c) + \sum_{u \in U} \Delta(u) > 0$. By putting time limits on encashing payment orders, a provider can control the time bound on detecting compromised devices.

Secondly, a compromised device can refuse to pay for the content it receives. According to assumption A5, user devices are provided with (and thus aware of) credits. Therefore, the second attack could easily be detected by the banking entity (collaborating with the providers) when a device signs a payment order without having enough credit for that, as a compliant device would not cause this error.

Given that cheating – and thus compromised – devices are detected, countermeasures can be taken. Confiscation of the compromised device is of course preferred.

[10] We do not discuss illegal content extraction attacks. For related discussions and countermeasures see [15].

However, in practice this will not always be possible. Instead, a Device Revocation List (DRL), containing public keys of detected compromised devices, can be used to limit interactions of compliant devices with them. To ensure correct working of device revocation, soundness of this DRL is required – no compliant device is ever listed on the DRL.

Completeness of the DRL is also desirable: all compromised devices are listed on the DRL. Such a list could grow quite large over time, as more and more compromised devices are detected. However, not all devices are equally likely to interact. Therefore, there is a trade-off between security and the size of the DRL stored on a compliant device. There are various possibilities of how to update that part of the DRL, that is stored on a device. The following notation is used in the discussion of some of these possibilities:

L The main DRL, as kept by P

l_d the DRL as kept by device D

f_d the list of devices with which device D has had contact.
This list is extended each time a device is contacted, which is not in the list. To keep the size of this list within reasonable bounds, it could be erased after each contact with the provider that updates the DRL.

Below, four update schemes are described: two extremes ("complete copy" and "friend-check") and two between those extremes. They vary in size of the list stored on devices and security provided (the speed with which relevant updates to the DRL are propagated to devices).

complete copy: Each device keeps a copy of the entire DRL.
Update: On contact with P: $l_c := L$.
On contact with D: $l_c := l_c \cup l_d$.
Advantage: additions to the DRL quickly propagate to compliant devices.
Disadvantage: Large list on each device; large communication overhead.

friend-check: A device only lists those revoked devices, with which it has had contact.
Update: On contact with P: $l_c := f_c \cap L$.
On contact with D: $l_c := l_c$.
Advantage: small lists on each device; low communication overhead
Disadvantage: compromised devices only appear on the DRL list of D after they have cheated and have had contact with D. This can be too late to prevent D from being cheated.

propagated list: Each device includes the DRL of all devices it has contacted.
Update: On contact with P: $l_c := l_c \cup (f_c \cap L)$.
On contact with D: $l_c := l_c \cup l_d$
Advantage: relatively quick propagation of DRL.
Disadvantage: relatively large list stored.

restricted propagation: each device includes the DRL of all devices it has contacted, but does not propagate this further.
In order to do this, new lists $rest_s, rest_o$ are introduced. They partition the DRL

into a part C has contacted themselves ($restr_s(C)$) and a part that other devices have contacted and told C about ($restr_o(C)$). In both cases, l_c is given by $l_c = restr_s(C) \cup restr_o(C)$.
Update: On contact with P: $restr_s(C) := f_c \cap L$.
On contact with D: $restr_o(C) := restr_o(C) \cup rest_s(D)$.
Advantage: relatively short list stored.
Disadvantage: relatively slower propagation of DRL.

Of these four methods, restricted propagation seems to balance the length of the list stored the best against the propagation of the DRL. On a further note, the impact of a corrupted DRL can be limited if each device cleans its DRL every time it contacts the provider ($l_c := l_c \cap L$). This would also allow the provider to "un-revoke" devices.

6 Conclusions

We presented a refined version of the NPGCT scheme, formally verified it and proved that it meets its requirements. As possible future research we consider studying the accountability of the provider and incorporating the payment phase into the formal model, besides studying the practical implementation feasibility of the proposed scheme.

Acknowledgements

We are grateful to Bruno Crispo and Wan Fokkink for reviewing this work and to Bert Lisser for his help in distributed model checking.

References

1. B. Alpern and F. B. Schneider. Defining liveness. Technical Report TR 85-650, Dept. of Computer Science, Cornell University, Ithaca, NY, October 1984.
2. N. Asokan. *Fairness in electronic commerce*. PhD thesis, University of Waterloo, 1998.
3. G. Avoine, F. Gartner, R. Guerraoui, K. Kursaweothers, S. Vaudenay, and M. Vukolic. Reducing fair exchange to atomic commit. Technical Report IC/2004/11, Swiss Federal Institute of Technology (EPFL), 2004.
4. J. Bergstra and J. Klop. Process algebra for synchronous communication. *Information and Control*, 60(3):109–137, 1984.
5. C. Boyd and A. Mathuria. *Protocols for authentication and key establishment*. Information Security and Cryptography. Springer-Verlag, 2003.
6. J. Cederquist and M. Torabi Dashti. An intruder model for verifying termination in security protocols. Technical Report TR-CTIT-05-29, University of Twente, Enschede, The Netherlands, 2005.
7. I. Cervesato. Data access specification and the most powerful symbolic attacker in MSR. In *ISSS '02*, volume 2609 of *LNCS*, pages 384–416. Springer, 2003.
8. H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *J. of Telecommunications and Information Technology*, 4:3–13, 2002.
9. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.
10. J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In *CAV '98*, volume 1102 of *LNCS*, pages 437–440. Springer-Verlag, 1996.

11. J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer-Verlag, 1995.
12. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *CSFW '00*, page 255, Washington, DC, USA, 2000. IEEE Computer Society.
13. R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free μ-calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.
14. C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.
15. S. Nair, B. Popescu, C. Gamage, B. Crispo, and A. Tanenbaum. Enabling DRM-preserving digital content redistribution. In *7th International IEEE Conference on E-Commerce Technology*, pages 151–158, München, Germany, 19-22, July 2005. IEEE Computer Society.
16. H. Pagnia and F. C. Gartner. On the impossibility of fair exchange without a trused third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology, 1999.
17. R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *CSFW '02*, page 282, Washington, DC, USA, 2002. IEEE Computer Society.
18. P. Y. A. Ryan, S. A. Schneider, M. H. Goldsmith, G. Lowe, and A. W. Roscoe. *The Modelling and Analysis of Security Protocols : the CSP Approach*. Addison-Wesley, 2001.
19. M. Terada, M. Iguchi, M. Hanadate, and K. Fujimura. An optimistic fair exchange protocol for trading electronic rights. In *CARDIS '04*, pages 255–270. Kluwer, 2004.

A A μCRL specification of the intruder model

In this section we briefly introduce the μCRL specification language, which is then used to formalize the intruder process described in Section 4.2.

The μCRL specification language In a μCRL specification processes are represented by process terms, which describe the order in which the actions may happen in a process. A process term consists of action names and recursion variables combined by process algebraic operators. The operators '$\cdot$' and '$+$' are used for the sequential and alternative composition ("choice") of processes, respectively. The operator $\sum_{d\in D} P(d)$ behaves like $P(d_1)+P(d_2)+\cdots$. The process expression $p \lhd b \rhd q$, where b is a term of sort **Bool** and p and q are processes, behaves like p if b is true, and like q if b is false. The predefined action δ represents a deadlock, i.e. from then on, no action can be performed.

The intruder model In the specification below, *Agent* represents the set of all honest participants of the protocol and *Msg* represents the set of all messages. X is the intruder's knowledge set. Y contains messages buffered for delivery. The set operators $\cup$ and $\setminus$ have their usual meanings. The set $\mathit{synth}(X)$ represents the (infinite) set of messages that the intruder is able to synthesize from the messages in set X, e.g. by applying pairing, signing and so on. For a complete description of this model we refer to [6].

$$
\begin{aligned}
I(X,Y) = &\sum_{\substack{p\in Agent\\ m\in Msg}} \mathit{recv}(p,m,I).I(X\cup\{m\}, Y\cup\{m\}) + \\
&\sum_{\substack{p\in Agent\\ m\in Msg}} \mathit{send}(I,m,p).I(X, Y\setminus\{m\}) \lhd m\in Y \rhd \delta + \\
&\sum_{\substack{p\in Agent\\ m\in Msg}} \mathit{send}^{\dagger}(I,m,p).I(X,Y) \lhd m\in \mathit{synth}(X)\setminus Y \rhd \delta + \\
&\sum_{m\in Cont} \mathit{revealed}(m).\delta \lhd m\in \mathit{synth}(X) \rhd \delta
\end{aligned}
$$

B μCRL code

The following code presents a μCRL specification of the Nuovo DRM scheme for scenario S_1 (Section 4.4).

```
% Nuovo DRM protocol spec
% C and D are two complaint devices.
% P is a legitimate content provider and banking entity.
% The intruder, I, is the owner of the compliant devices and
% controls the communication media.
% We model three concurrent runs of P and three sequential runs
% of each of C and D.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bool Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Bool
func    T,F: -> Bool
map     and,or: Bool # Bool -> Bool
        not: Bool -> Bool
        eq: Bool # Bool -> Bool
        leq: Bool # Bool -> Bool
        if: Bool # Bool # Bool -> Bool
var     x,x': Bool
rew
        if(T,x,x')=x
        if(F,x,x')=x'

        and(T,T)=T
        and(F,x)=F
        and(x,F)=F
        or(T,x)=T
        or(x,T)=T
        or(F,F)=F
        not(F)=T
        not(T)=F

        %implication.
        leq(x,x)=T
        leq(F,x)=T
        leq(T,F)=F

        eq(x,x')=and(leq(x,x'),leq(x',x))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Key Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Key
func
        %symmetric keys used by P
        k1 :-> Key
        k2 :-> Key
        k3 :-> Key
        %symmetric key used by C
        kc :-> Key
        %symmetric key used by D
        kd :-> Key
        %symmetric key owned by I
```

```
        ki :-> Key
map     eq:Key#Key->Bool
        leq:Key#Key->Bool
var     k,k': Key
rew     leq(k,k)=T
        leq(k1,k2)=T    leq(k1,k3)=T    leq(k1,ki)=T    leq(k1,kc)=T    leq(k1,kd)=T
        leq(k2,k1)=F    leq(k2,k3)=T    leq(k2,ki)=T    leq(k2,kc)=T    leq(k2,kd)=T
        leq(k3,k1)=F    leq(k3,k2)=F    leq(k3,ki)=T    leq(k3,kc)=T    leq(k3,kd)=T
        leq(ki,k1)=F    leq(ki,k2)=F    leq(ki,k3)=F    leq(ki,kc)=T    leq(ki,kd)=T
        leq(kc,k1)=F    leq(kc,k2)=F    leq(kc,k3)=F    leq(kc,ki)=F    leq(kc,kd)=T
        leq(kd,k1)=F    leq(kd,k2)=F    leq(kd,k3)=F    leq(kd,ki)=F    leq(kd,kc)=F

        eq(k,k')=and(leq(k,k'),leq(k',k))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Item Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Item
func
        %items that P provides
        dp1,dp2: -> Item
        %item that I owns
        di :-> Item
map     eq: Item # Item -> Bool
        leq: Item # Item -> Bool
var     d,d': Item
rew     leq(d,d)=T
        leq(dp1,dp2)=T  leq(dp1,di)=T
        leq(dp2,dp1)=F  leq(dp2,di)=T
        leq(di,dp1)=F   leq(di,dp2)=F

        eq(d,d')=and(leq(d,d'),leq(d',d))


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Right Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Right
func
        %rights, known to every one
        r1,r2: -> Right
map     eq: Right # Right -> Bool
        leq: Right # Right -> Bool
var     r,r': Right
rew     leq(r,r)=T
        leq(r1,r2)=T
        leq(r2,r1)=F

        eq(r,r')=and(leq(r,r'),leq(r',r))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Nonce Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Nonce
func    n1: Player -> Nonce
        n2: Player -> Nonce
```

```
        n3: Player -> Nonce
        nf :-> Nonce
map     eq: Nonce # Nonce -> Bool
        leq: Nonce # Nonce -> Bool
        nxt : Nonce -> Nonce
var     p,p': Player
        n,n': Nonce
rew
        leq(nf,nf)=T
        leq(nf,n1(p))=T
        leq(nf,n2(p))=T
        leq(nf,n3(p))=T
        leq(n1(p),nf)=F
        leq(n2(p),nf)=F
        leq(n3(p),nf)=F

        leq(n1(p),n1(p'))=leq(p,p')
        leq(n1(p),n2(p'))=leq(p,p')
        leq(n1(p),n3(p'))=leq(p,p')
        leq(n2(p),n1(p'))=if(leq(p,p'),not(eq(p,p')),F)
        leq(n2(p),n2(p'))=leq(p,p')
        leq(n2(p),n3(p'))=leq(p,p')
        leq(n3(p),n1(p'))=if(leq(p,p'),not(eq(p,p')),F)
        leq(n3(p),n2(p'))=if(leq(p,p'),not(eq(p,p')),F)
        leq(n3(p),n3(p'))=leq(p,p')

        eq(n,n')=and(leq(n,n'),leq(n',n))


% imposing finite number of runs on each process and
% providing different nonces to different instances of P
        nxt(n1(p))=n2(p)
        nxt(n2(p))=n3(p)
        nxt(n3(p))=nf
        nxt(nf)=nf

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Player Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort    Player
func
        P,D,C,I :-> Player
map     eq : Player # Player->Bool
        leq: Player # Player->Bool
var     p,p': Player
rew     leq(p,p)=T
        leq(I,C)=T      leq(I,D)=T      leq(I,P)=T
        leq(C,I)=F      leq(C,D)=T      leq(C,P)=T
        leq(D,I)=F      leq(D,C)=F      leq(D,P)=T
        leq(P,I)=F      leq(P,C)=F      leq(P,D)=F

        eq(p,p')=and(leq(p,p'),leq(p',p))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Message Data Type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
sort    Message
func
        rgt: Right -> Message
        itm: Item->Message
        plyr: Player -> Message
        keym: Key -> Message
        nnc : Nonce -> Message
        pair: Message#Message->Message
        h: Message->Message
        sign: Player#Message->Message   %signed message -by- a player
        penc : Player#Message->Message  %public-key encryption -for- a player
        enc: Key#Message->Message

map     eq: Message#Message->Bool
        leq: Message#Message->Bool
var     m1,m2,m3,m4: Message
        p,p': Player
        d,d': Item
        k,k': Key
        n,n': Nonce
        r,r': Right

rew
        eq(m1,m2)=and(leq(m1,m2),leq(m2,m1))

        leq(itm(d),itm(d'))=leq(d,d')
        leq(rgt(r),rgt(r'))=leq(r,r')
        leq(nnc(n),nnc(n'))=leq(n,n')
        leq(plyr(p),plyr(p'))=leq(p,p')
        leq(keym(k),keym(k'))=leq(k,k')
        leq(pair(m1,m2),pair(m3,m4))=if(eq(m1,m3),leq(m2,m4),leq(m1,m3))
        leq(h(m1),h(m2))=leq(m1,m2)
        leq(sign(p,m1),sign(p',m2))=if(eq(p,p'),leq(m1,m2),leq(p,p'))
        leq(penc(p,m1),penc(p',m2))=if(eq(p,p'),leq(m1,m2),leq(p,p'))
        leq(enc(k,m1),enc(k',m2))=if(eq(k,k'),leq(m1,m2),leq(k,k'))

        leq(itm(d),rgt(r'))=T
        leq(itm(d),nnc(n'))=T
        leq(itm(d),plyr(p'))=T
        leq(itm(d),pair(m3,m4))=T
        leq(itm(d),h(m2))=T
        leq(itm(d),sign(p',m2))=T
        leq(itm(d),penc(p',m2))=T
        leq(itm(d),enc(k',m2))=T
        leq(itm(d),keym(k'))=T

        leq(rgt(r'),itm(d))=F
        leq(rgt(r'),nnc(n'))=T
        leq(rgt(r'),plyr(p'))=T
        leq(rgt(r'),pair(m3,m4))=T
        leq(rgt(r'),h(m2))=T
        leq(rgt(r'),sign(p',m2))=T
        leq(rgt(r'),penc(p',m2))=T
        leq(rgt(r'),enc(k',m2))=T
        leq(rgt(r'),keym(k'))=T
```

```
leq(nnc(n'),itm(d))=F
leq(nnc(n'),rgt(r'))=F
leq(nnc(n'),plyr(p'))=T
leq(nnc(n'),pair(m3,m4))=T
leq(nnc(n'),h(m2))=T
leq(nnc(n'),sign(p',m2))=T
leq(nnc(n'),penc(p',m2))=T
leq(nnc(n'),enc(k',m2))=T
leq(nnc(n'),keym(k'))=T

leq(plyr(p'),itm(d))=F
leq(plyr(p'),rgt(r'))=F
leq(plyr(p'),nnc(n'))=F
leq(plyr(p'),pair(m3,m4))=T
leq(plyr(p'),h(m2))=T
leq(plyr(p'),sign(p,m2))=T
leq(plyr(p'),penc(p,m2))=T
leq(plyr(p'),enc(k',m2))=T
leq(plyr(p'),keym(k'))=T

leq(pair(m3,m4),itm(d))=F
leq(pair(m3,m4),rgt(r'))=F
leq(pair(m3,m4),nnc(n'))=F
leq(pair(m3,m4),plyr(p'))=F
leq(pair(m3,m4),h(m2))=T
leq(pair(m3,m4),sign(p,m2))=T
leq(pair(m3,m4),penc(p,m2))=T
leq(pair(m3,m4),enc(k',m2))=T
leq(pair(m3,m4),keym(k'))=T

leq(h(m1),itm(d))=F
leq(h(m1),rgt(r'))=F
leq(h(m1),nnc(n'))=F
leq(h(m1),plyr(p'))=F
leq(h(m1),pair(m3,m4))=F
leq(h(m1),sign(p,m2))=T
leq(h(m1),penc(p,m2))=T
leq(h(m1),enc(k',m2))=T
leq(h(m1),keym(k'))=T

leq(sign(p,m1),itm(d))=F
leq(sign(p,m1),rgt(r'))=F
leq(sign(p,m1),nnc(n'))=F
leq(sign(p,m1),plyr(p'))=F
leq(sign(p,m1),pair(m3,m4))=F
leq(sign(p,m1),h(m2))=F
leq(sign(p,m1),penc(p',m2))=T
leq(sign(p,m1),enc(k',m2))=T
leq(sign(p,m1),keym(k'))=T

leq(penc(p,m2),itm(d))=F
leq(penc(p,m2),rgt(r'))=F
leq(penc(p,m2),nnc(n'))=F
leq(penc(p,m2),plyr(p'))=F
leq(penc(p,m2),pair(m3,m4))=F
leq(penc(p,m2),h(m1))=F
```

```
        leq(penc(p',m2),sign(p,m1))=F
        leq(penc(p,m1),enc(k',m2))=T
        leq(penc(p,m2),keym(k'))=T

        leq(enc(k',m1),itm(d))=F
        leq(enc(k',m1),rgt(r'))=F
        leq(enc(k',m1),nnc(n'))=F
        leq(enc(k',m1),plyr(p'))=F
        leq(enc(k',m1),pair(m3,m4))=F
        leq(enc(k',m1),h(m2))=F
        leq(enc(k',m1),sign(p,m2))=F
        leq(enc(k',m1),penc(p,m2))=F
        leq(enc(k',m1),keym(k))=T

% This ordering, putting the symmetric keys at the end, is used
% in the implementation of -- decrypt -- function below.
        leq(keym(k'),itm(d))=F
        leq(keym(k'),rgt(r'))=F
        leq(keym(k'),nnc(n'))=F
        leq(keym(k'),plyr(p'))=F
        leq(keym(k'),pair(m3,m4))=F
        leq(keym(k'),h(m2))=F
        leq(keym(k'),sign(p,m1))=F
        leq(keym(k'),penc(p,m1))=F
        leq(keym(k'),enc(k,m1))=F

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ordered set of Messages
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sort    Knowledge
func    emptyset :-> Knowledge
        set : Message # Knowledge -> Knowledge
map     eq: Knowledge # Knowledge -> Bool
        in: Message # Knowledge -> Bool
        if: Bool # Knowledge # Knowledge -> Knowledge
        add: Message # Knowledge -> Knowledge
        rem: Message # Knowledge -> Knowledge
        synthadd: Message # Knowledge -> Knowledge
        union : Knowledge # Knowledge -> Knowledge

var     u,v: Knowledge
        m,m': Message

rew
        eq(emptyset,emptyset)=T
        eq(emptyset, set(m,u))=F
        eq(set(m,u), emptyset)=F
        eq(set(m,u),set(m',v))=if(eq(m,m'),eq(u,v),F)

        in(m,emptyset)=F
        in(m,set(m',u))=if(leq(m,m'),eq(m,m'),in(m,u))

        if(T,u,v)=u
        if(F,u,v)=v
```

```
        add(m,emptyset)= set(m,emptyset)
        add(m,set(m',u))=if(leq(m,m'),
                            if(eq(m,m'),set(m',u),set(m,set(m',u)))
                            ,set(m',add(m,u)))

        rem(m,emptyset)=emptyset
        rem(m,set(m',u))=if(leq(m,m'),if(eq(m,m'),u,set(m',u)),set(m',rem(m,u)))

        synthadd(m,u)=if(synth(m,u),u,add(m,u))

        union(u,emptyset)=u
        union(u,set(m,v))=synthadd(m,union(u,v))

% What the intruder can synthesize
map     synth: Message # Knowledge -> Bool
var
        u : Knowledge
        m,m': Message
        p: Player
        r: Right
        d: Item
        n: Nonce
        k: Key
rew
        % Intruder has no hard-coded knowledge,
        % except for the public keys of other agents
        % and its own private key.

        synth(itm(d),u)=in(itm(d),u)
        synth(rgt(r),u)=in(rgt(r),u)
        synth(nnc(n),u)=in(nnc(n),u)
        synth(plyr(p),u)=in(plyr(p),u)
        synth(pair(m,m'),u)=and(synth(m,u),synth(m',u))
        synth(h(m),u)=or(in(h(m),u),synth(m,u))
        synth(sign(p,m),u)=or(in(sign(p,m),u),and(eq(p,I),synth(m,u)))
        synth(penc(p,m),u)=or(in(penc(p,m),u),synth(m,u))
        synth(enc(k,m),u)=or(in(enc(k,m),u),and(synth(keym(k),u),synth(m,u)))
        synth(keym(k),u)=in(keym(k),u)


% decomp function
map     decomp: Message -> Knowledge
var
        m,m': Message
        p: Player
        r: Right
        d: Item
        n: Nonce
        k: Key
rew
        decomp(itm(d))=set(itm(d),emptyset)
        decomp(rgt(r))= set(rgt(r),emptyset)
        decomp(nnc(n))=set(nnc(n),emptyset)
        decomp(plyr(p))= set(plyr(p),emptyset)
        decomp(pair(m,m'))=union(decomp(m),decomp(m'))
        decomp(h(m))=set(h(m),emptyset)
```

```
        % A signature includes also the (signed) plain message.
        decomp(sign(p,m))=synthadd(sign(p,m),decomp(m))
        decomp(penc(p,m))=if(eq(p,I),decomp(m),set(penc(p,m),emptyset))
        % Recursive decryption is left for 'decrypt' function
        decomp(enc(k,m))=set(enc(k,m),emptyset)
        decomp(keym(k))=set(keym(k),emptyset)

% decrypt function
map     decrypt: Knowledge -> Knowledge
var
        u : Knowledge
        m,m': Message
        p: Player
        r: Right
        d: Item
        n: Nonce
        k: Key
rew
        decrypt(emptyset)=emptyset
        decrypt(set(itm(d),u))=add(itm(d),decrypt(u))
        decrypt(set(rgt(r),u))=add(rgt(r),decrypt(u))
        decrypt(set(nnc(n),u))=add(nnc(n),decrypt(u))
        decrypt(set(plyr(p),u))=add(plyr(p),decrypt(u))
        % Never happens:
        % decrypt(set(pair(m,m'),u))
        decrypt(set(h(m),u))=add(h(m),decrypt(u))
        decrypt(set(sign(p,m),u))=add(sign(p,m),decrypt(u))
        decrypt(set(penc(p,m),u))=add(penc(p,m),decrypt(u))
        decrypt(set(enc(k,m),u))=if(in(keym(k),u),union(decomp(m),decrypt(u)),
                                                    add(enc(k,m),decrypt(u)))
        % The keys come at the very end of an ordered set and have no
        % decryption, so there is no point in continuing the decryption.
        decrypt(set(keym(k),u))= set(keym(k),u)

% Unbounded rounds of decryption
% Since the compiling rewriter does not terminate on this function,
% the jitty rewriter should be used in instantiation phase.

map     re-decrypt : Knowledge -> Knowledge
var
        u : Knowledge
rew
        re-decrypt(u)=if(eq(u,decrypt(u)),u,re-decrypt(decrypt(u)))

% Intruder's initial knowledge (ordered set)
map     X0 :-> Knowledge
rew
        X0 =
                set(itm(di),
                set(rgt(r1),
                set(rgt(r2),
                set(nnc(n1(I)),
                set(nnc(n2(I)),
                set(plyr(I),
                set(plyr(C),
                set(plyr(D),
```

```
                set(plyr(P),
                set(h(itm(dp1)),
                set(h(itm(dp2)),
                set(keym(ki),
                emptyset))))))))))))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Declaration of Actions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
act     send,sendx,recv,com,comx: Player#Message#Player
        update: Player # Item # Right # Player
        issue: Player # Item # Right # Player
        paid : Player # Item # Right # Player
        request: Player # Item # Right # Player
        revealed : Item
        resolves : Player
        last_ttp
        off

comm    send|recv=com
        sendx|recv=comx


proc
% The owner of compliant devices, the intruder.

I(X:Knowledge,Y:Knowledge)=
        sum(p:Player,sum(m:Message,
                recv(p,m,I).I(decrypt(union(decomp(m),X)),add(m,Y))
        +
                sendx(I,m,p).I(X,Y)<|and(synth(m,X),not(in(m,Y)))|>delta
        +
                send(I,m,p).I(X,rem(m,Y))<|in(m,Y)|>delta
        ))
        +
        sum(d:Item,
                (revealed(d).delta<|synth(itm(d),X)|>delta)
        <|or(eq(d,dp2),eq(d,dp1))|>delta)

% Compliant device C
% An "off" action has been added as an alternative to the critical
% actions of C. Critical actions are updating rights, engaging in
% a transaction and sending an item to another customer in the
% re-sell phase. It characterizes the intruder's power to deliberately
% turn a compliant device off.

iPod1(Y:Knowledge, nc:Nonce)=
delta<|eq(nc,nf)|>
(
        %buy from the provider
        sum(d:Item,sum(r:Right,
                recv(I,pair(plyr(P),pair(h(itm(d)),rgt(r))),C).
                send(C,pair(plyr(C),nnc(nc)),I).
                sum(n':Nonce,
                        recv(I,sign(P,pair(pair(nnc(n'),nnc(nc)),plyr(C))),C).
                        send(C,sign(C,
```

```
                        pair(pair(pair(nnc(nc),nnc(n')),
                             pair(h(itm(d)),rgt(r))),
                        plyr(P))),I).
                (request(C,d,r,P)+off.iPod1(Y,nxt(nc))).
                sum(k:Key,
                        recv(I,pair(enc(k,itm(d)),pair(penc(C,keym(k)),
                                        sign(P,pair(rgt(r),nnc(nc))))),C).
                        (update(C,d,r,P)+off.iPod1(Y,nxt(nc)))
                )
        ).
        iPod1(add(pair(itm(d),rgt(r)),Y),nxt(nc))
))
+
%buy from a customer
sum(d:Item,sum(r:Right,
        recv(I,pair(plyr(D),pair(h(itm(d)),rgt(r))),C).
        send(C,pair(plyr(C),nnc(nc)),I).
        sum(n':Nonce,
                recv(I,sign(D,pair(pair(nnc(n'),nnc(nc)),plyr(C))),C).
                send(C,sign(C,pair(pair(pair(nnc(nc),nnc(n')),
                        pair(h(itm(d)),rgt(r))),plyr(D))),I).
                (request(C,d,r,D)+off.iPod1(Y,nxt(nc))).
                (
                (
                %normal run
                sum(k:Key,
                        recv(I,pair(enc(k,itm(d)),pair(penc(C,keym(k)),
                                        sign(D,pair(rgt(r),nnc(nc))))),C).
                        (update(C,d,r,D)+off.iPod1(Y,nxt(nc)))
                        )
                )
                +
                (
                %resolve protocol
                        (resolves(C).send(C,pair(plyr(C),nnc(nc)),I)+off.iPod1(Y,nxt(nc))).
                        sum(nz:Nonce,
                          recv(I,sign(P,pair(pair(nnc(nz),nnc(nc)),plyr(C))),C).
                          send(C,sign(C,pair(pair(pair(nnc(nc),nnc(nz)),
                             %the obselete request to D.
                             pair(pair(pair(nnc(nc),nnc(n')),
                             pair(h(itm(d)),rgt(r))),plyr(D))
                                ),plyr(P))),I).
                             (request(C,d,r,P)+off.iPod1(Y,nxt(nc))).
                             sum(k:Key,
                                recv(I,pair(enc(k,itm(d)),pair(penc(C,keym(k)),
                                        sign(P,pair(rgt(r),nnc(nc))))),C).
                                (update(C,d,r,P)+off.iPod1(Y,nxt(nc)))
                             )
                        )
                 )
                )
        ).
        iPod1(add(pair(itm(d),rgt(r)),Y),nxt(nc))
))
+
%sell to a customer
```

```
        sum(n:Nonce,
                recv(I,pair(plyr(D),nnc(n)),C).
                send(C,sign(C,pair(pair(nnc(nc),nnc(n)),plyr(D))),I).
                sum(d:Item,sum(r:Right,
                        recv(I,sign(D,
                                pair(pair(pair(nnc(n),nnc(nc)),
                                     pair(h(itm(d)),rgt(r))),
                                plyr(C))),C).
                        (
                        (issue(C,d,r,D)+off.iPod1(Y,nxt(nc))).
                        (send(C,pair(enc(kc,itm(d)),pair(penc(D,keym(kc)),
                                        sign(C,pair(rgt(r),nnc(n))))),I)+off.iPod1(Y,nxt(nc)))
                        )<|in(pair(itm(d),rgt(r)),Y)|>delta
                )).
                iPod1(Y,nxt(nc))
        )
)


% Compliant device D
iPod2(Y:Knowledge, nd:Nonce)=
delta<|eq(nd,nf)|>
(
        %buy from the provider
        sum(d:Item,sum(r:Right,
                recv(I,pair(plyr(P),pair(h(itm(d)),rgt(r))),D).
                send(D,pair(plyr(D),nnc(nd)),I).
                sum(n':Nonce,
                        recv(I,sign(P,pair(pair(nnc(n'),nnc(nd)),plyr(D))),D).
                        send(D,sign(D,
                                pair(pair(pair(nnc(nd),nnc(n')),
                                     pair(h(itm(d)),rgt(r))),
                                plyr(P))),I).
                        request(D,d,r,P).
                        sum(k:Key,
                                recv(I,pair(enc(k,itm(d)),pair(penc(D,keym(k)),
                                                sign(P,pair(rgt(r),nnc(nd))))),D).
                                update(D,d,r,P)
                        )
                ).
                iPod2(add(pair(itm(d),rgt(r)),Y),nxt(nd))
        ))
        +
        %buy from a customer
        sum(d:Item,sum(r:Right,
                recv(I,pair(plyr(C),pair(h(itm(d)),rgt(r))),D).
                send(D,pair(plyr(D),nnc(nd)),I).
                sum(n':Nonce,
                        recv(I,sign(C,pair(pair(nnc(n'),nnc(nd)),plyr(D))),D).
                        send(D,sign(D,
                                pair(pair(pair(nnc(nd),nnc(n')),
                                     pair(h(itm(d)),rgt(r))),
                                plyr(C))),I).
                        request(D,d,r,C).
                        (
                        (
```

```
                        %normal run
                        sum(k:Key,
                                recv(I,pair(enc(k,itm(d)),pair(penc(D,keym(k)),
                                                sign(C,pair(rgt(r),nnc(nd))))),D).
                                update(D,d,r,C)
                                )
                        )
                        +
                        %resolve protocol
                        (
                                resolves(D).
                                send(D,pair(plyr(D),nnc(nd)),I).
                                sum(nz:Nonce,
                                  recv(I,sign(P,pair(pair(nnc(nz),nnc(nd)),plyr(D))),D).
                                  send(D,sign(D,pair(pair(pair(nnc(nd),nnc(nz)),
                                     %the obselete request to C.
                                     pair(pair(pair(nnc(nd),nnc(n')),
                                     pair(h(itm(d)),rgt(r))),plyr(C))
                                        ),plyr(P))),I).
                                     request(D,d,r,P).
                                     sum(k:Key,
                                        recv(I,pair(enc(k,itm(d)),pair(penc(D,keym(k)),
                                                sign(P,pair(rgt(r),nnc(nd))))),D).
                                        update(D,d,r,P)
                                     )
                                )
                         )
                )
                ).
                iPod2(add(pair(itm(d),rgt(r)),Y),nxt(nd))
        ))
        +
        %sell to a customer
        sum(n:Nonce,
                recv(I,pair(plyr(C),nnc(n)),D).
                send(D,sign(D,pair(pair(nnc(nd),nnc(n)),plyr(C))),I).
                sum(d:Item,sum(r:Right,
                        recv(I,sign(C,
                                pair(pair(pair(nnc(n),nnc(nd)),
                                     pair(h(itm(d)),rgt(r))),
                                plyr(D))),D).
                        (
                        issue(D,d,r,C).
                        send(D,pair(enc(kd,itm(d)),pair(penc(C,keym(kd)),
                                        sign(D,pair(rgt(r),nnc(n))))),I)
                        )<|in(pair(itm(d),rgt(r)),Y)|>delta
                )).
                iPod2(Y,nxt(nd))
        )
)


% Provider
iTune1=
sum(n:Nonce,sum(g:Player,
        (
```

```
        recv(I,pair(plyr(g),nnc(n)),P).
        send(P,sign(P,pair(pair(nnc(n1(P)),nnc(n)),plyr(g))),I).
        sum(d:Item,sum(r:Right,
        (
                (
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n1(P))),
                             pair(h(itm(d)),rgt(r))),plyr(P))),P).
                issue(P,d,r,g).
                send(P,pair(enc(k1,itm(d)),pair(penc(g,keym(k1)),
                        sign(P,pair(rgt(r),nnc(n))))),I)
                )
                +
                (
                sum(n':Nonce,sum(n'':Nonce,sum(g':Player,
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n1(P))),
                        %the obselete request to a customer.
                        pair(pair(pair(nnc(n'),nnc(n'')),
                        pair(h(itm(d)),rgt(r))),plyr(g'))
                        ),plyr(P))),P)
                ))).
                issue(P,d,r,g).
                send(P,pair(enc(k1,itm(d)),pair(penc(g,keym(k1)),
                        sign(P,pair(rgt(r),nnc(n))))),I)
                )
        )
        <|or(eq(d,dp1),eq(d,dp2))|>delta))
        )
        <|or(eq(g,C),eq(g,D))|>delta)).delta


% Provider
iTune2=
sum(n:Nonce,sum(g:Player,
        (
        recv(I,pair(plyr(g),nnc(n)),P).
        send(P,sign(P,pair(pair(nnc(n2(P)),nnc(n)),plyr(g))),I).
        sum(d:Item,sum(r:Right,
        (
                (
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n2(P))),
                             pair(h(itm(d)),rgt(r))),plyr(P))),P).
                issue(P,d,r,g).
                send(P,pair(enc(k2,itm(d)),pair(penc(g,keym(k2)),
                        sign(P,pair(rgt(r),nnc(n))))),I)
                )
                +
                (
                sum(n':Nonce,sum(n'':Nonce,sum(g':Player,
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n2(P))),
                        %the obselete request to C.
                        pair(pair(pair(nnc(n'),nnc(n'')),
                        pair(h(itm(d)),rgt(r))),plyr(g'))
                        ),plyr(P))),P)
                ))).
                issue(P,d,r,g).
                send(P,pair(enc(k2,itm(d)),pair(penc(g,keym(k2)),
```

```
                        sign(P,pair(rgt(r),nnc(n)))))),I)
                )
        )
        <|or(eq(d,dp1),eq(d,dp2))|>delta))
        )
        <|or(eq(g,C),eq(g,D))|>delta)).delta

% Provider
iTune3=
last_ttp.
sum(n:Nonce,sum(g:Player,
        (
        recv(I,pair(plyr(g),nnc(n)),P).
        send(P,sign(P,pair(pair(nnc(n3(P)),nnc(n)),plyr(g))),I).
        sum(d:Item,sum(r:Right,
        (
                (
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n3(P))),
                             pair(h(itm(d)),rgt(r))),plyr(P))),P).
                issue(P,d,r,g).
                send(P,pair(enc(k3,itm(d)),pair(penc(g,keym(k3)),
                        sign(P,pair(rgt(r),nnc(n)))))),I)
                )
                +
                (
                sum(n':Nonce,sum(n'':Nonce,sum(g':Player,
                recv(I,sign(g,pair(pair(pair(nnc(n),nnc(n3(P))),
                        %the obselete request to C.
                        pair(pair(pair(nnc(n'),nnc(n'')),
                        pair(h(itm(d)),rgt(r))),plyr(g'))
                        ),plyr(P))),P)
                ))).
                issue(P,d,r,g).
                send(P,pair(enc(k3,itm(d)),pair(penc(g,keym(k3)),
                        sign(P,pair(rgt(r),nnc(n)))))),I)
                )
        )
        <|or(eq(d,dp1),eq(d,dp2))|>delta))
        )
        <|or(eq(g,C),eq(g,D))|>delta)).delta

init
hide({com,off},encap({send,sendx,recv},I(X0,emptyset)||
                        iPod1(emptyset,n1(C))||iPod2(emptyset,n1(D))||
                        iTune1||iTune2||iTune3))
```