Consiglio Nazionale delle Ricerche

Formal models and analysis of secure multicast in wired and wireless networks

R. Gorrieri, F. Martinelli, M. Petrocchi

IIT TR-07/2006

Technical report

Giugno 2006

Istituto di Informatica e Telematica

Formal models and analysis of secure multicast in wired and wireless networks*

Roberto Gorrieri
Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy and
Fabio Martinelli, Marinella Petrocchi
IIT - CNR, Pisa, Italy

Abstract. The spreading of multicast technology enables the development of group communication and so, dealing with digital streams becomes more and more common over the Internet. Given the flourishing of security threats, the distribution of streamed data must be equipped with sufficient security guarantees. To this aim, some architectures have been proposed in the last few years, to supply the distribution of the stream with guarantees of, *e.g.*, authenticity, integrity and confidentiality of the digital contents. This paper shows a formal capability of capturing some features of secure multicast protocols. In particular, both the modeling and the analysis of some case studies are shown, starting from basic schemes for signing digital streams, passing through protocols dealing with packet loss and time-synchronization requirements, concluding with a secure distribution of a secret key. A process-algebraic framework will be exploited, equipped with schemata for analysing security properties and compositional principles for evaluating if a property is satisfied over a system with more than two components.

Keywords. Formal security models and analysis, multicast communication.

1 Introduction

Multicast communication and security issues. With the wide use of Internet, the popularity of multicast technology has grown considerably. Examples include live-broadcasts, digitized audio and video, news feeds, stock quotes, multi-party video games, multi-party video conferences, data applets, software updates.

Dealing with multicast communication means, in the terminology currently present in the literature, dealing with digital streams, *i.e.*, long (potentially infinite) sequences of bits. The stream is typically sent from one sender to a set of receivers.

* This research was partly supported by the EU project FP6-IST-4-027748-IP BIONETS (*BIOlogically-inspired autonomic NETworks and Services*) and by the EU project FP6-IST-3-016004-IP-09 Sensoria (*Software Engineering for Service-Oriented Overlay Computers*).

Given that network security threats have flourished as well, the increasing trend to distribute streamed data over the Internet must be equipped with sufficient security guarantees. In particular, the so called *stream signature protocols* were born with the intent of efficiently solving the problem to sign digital streams. This class of protocols, designed for open architectures, makes usually use of hashing techniques and a thrifty use of standard digital signatures to ensure the authenticity of the sender and the integrity of the stream.

In some cases, confidentiality requirements are also due, as in a *pay per view* environment, where only a restricted group of authorized users must have the ability to consume the stream.

Formal verification of secure multicast. Along with the development of schemes for secure multicast, the use of formal techniques for their analysis represents an interesting challenge because of the diversity of such protocols from standard cryptographic schemes. Indeed, two peculiarities are: *i*) a sender broadcasts a continuous (and possibly unbounded) stream of messages to a (possibly unbounded) set of receivers; *ii*) receivers use information retrieved in earlier packets to legitimate later packets or vice-versa.

Thus, such a formal analysis have raised some interest among researchers. Some proposals have been given in the past few years. In [1], Archer states a formal analysis based on model checking techniques (*i.e.*, checking all the reachable states of a system with respect to the fulfillment of a certain property) is not feasible. In her opinion, this is because "an infinite state system is required to represent the inductive relationship between an arbitrary n-th packet and the initial packet". Instead, she exploits theorem proving techniques to analyze the basic version of a well known stream authentication protocol, the TESLA protocol, [22]. On the other hand, in [3] Broadfoot and Lowe show their successful results derived applying model checking techniques on TESLA, motivating, even though informally, several steps of the analysis. In particular, they have shown how to build a finite model of TESLA, despite of the possibly unboundedness of the stream of messages (and cryptographic keys) broadcasted by the sender.

Formal methods have been also exploited for analysing a multicast key management scheme, [26]. The authors model the scheme by a relational modeling language and perform the analysis using the Alloy Analyzer, an automatic simulation tool. The analysis highlights some flaws of the scheme previously unknown.

The analysis approach we are going to use throughout this paper is quite different from what has been proposed in the literature so far and focuses its attention on the verifiability of a system with an arbitrary number of components (as in the case of stream signature protocols). In particular, some compositional principles will be applied to the case studies we present. These principles will allow us to safely compose processes, in such a way that the overall system preserves the security property that each subsystem separately enjoys.

Goal of the paper. Goal of this paper is to show the formal capability of capturing security features of multicast protocols, like sensor networks protocols

with real time requirements and secret communication within a multicast group. Some case studies will be shown, modeled and analyzed by exploiting a process-algebraic framework, equipped with schemata for analysing security properties and compositional principles for evaluating if a property is satisfied over a system with more than two components.

Process algebras. Process algebras represent an algebraic approach to the study of concurrent processes. They are executable languages for the description of distributed systems. They allow both the specification of the processes and the formulation of statements about them, together with methodologies for the verification of these statements.

To facilitate a comparison between processes, several notions of behavioral equivalences have been defined within the algebras. We mainly deal with the notion of *weak bisimulation*, [20], recalled in the appendix. Also, two extensions of CCS (the Calculus of Communicating Systems, [20]), namely Crypto-CCS and tCryptoSPA, will be used throughout the paper (their syntax is concisely presented in Appendix A).

Compositional strategy and general schemata for security properties. A compositional principle gives sufficient conditions to conclude that the composition of two (or more) processes satisfies the composition of two (or more) properties, provided that the single processes satisfy the single properties. As an example, such a principle could work as follows: in order to check if a compound system $P||Q$ satisfies a formula f (where f says, *e.g.*, that the system is correct), it is enough to check whether both P and Q separately satisfy f. (Notation $||$ represents the parallel composition of processes, see also the appendix.) The existence of such a principle would be particularly appealing for the target of our analysis. Indeed, the state-space of the system $P||Q$ is usually considerably bigger than those of P and Q, separately. Above all, it would help in analyzing systems with a possibly unbounded number of components. Indeed, consider the parallel composition of n instances of process P:

$$\overbrace{P||\ldots||P}^{n}$$

To prove that the overall system enjoys f (for whatever n) it is sufficient to prove that P enjoys f.

Compositional principles will be used to verify i) an integrity property, *i.e.*, a sort of robustness against packet modification, and ii) a secrecy property requiring, informally, that the contents of the stream remains unknown to everybody but the sender and the intended receivers.

Some of these principles were first introduced in [13] for the Generalized Non Deducibility on Compositions scheme of properties, (GNDC for short), defined in [6, 9, 8]. In turn, the scheme (reminded in Appendix B) is based on the seminal notion of non-interference ([11]). We use these principles to verify an instance of a stream signature protocol dealing with packet loss, see [23] and Subsection 2.2 of this paper. Also, Subsection 3.4 shows an application of compositionality to

analyze confidentiality in a multicast protocol that use group encryption techniques [27].

A variant of this principle was introduced in [15, 13] within a formal framework aimed at verifying timed security properties, *i.e.*, security properties whose fulfilment is based on timed conditions. In turn, the timed formal framework, namely the timed-Generalized Non Deducibility on Compositions (tGNDC for short) has been introduced in [13]. Part of Appendix B is devoted to recall the tGNDC schema. The principle is applied in Subsection 3.3 to verify a protocol for broadcast authentication of data in wireless sensor networks [24].

Case studies. The case studies considered throughout the paper are: 1) the Gennaro-Rohatgi protocol [10], a pioneering protocol introduced in 1997 to sign digital streams; 2) the Efficient Multi-chained Stream Signature protocol (EMSS) proposed in [23]. This stream signature protocol implements a significant improvement over the Gennaro-Rohatgi protocol, since EMSS guarantees some robustness against packet loss; 3) the μTESLA ("micro" Timed Efficient Stream Loss-tolerant Authentication, [24]), a protocol to provide authenticated broadcast in wireless sensor networks environments; 4) finally, a protocol to distribute a secret key to a multicast group, [27].

Contributions. The main contributions of this paper are the following.

i) We formally model and analyze some relevant proposals for authenticating data streams and for giving them data confidentiality. To the best of our knowledge, this is the first attempt to prove some of the security properties of those protocols (by means of compositional rules).

ii) Starting from modeling the basic scheme of Gennaro and Rohatgi, passing through protocols dealing with packet loss, concluding with a time-dependent security wireless protocol and with a secure distribution of a key, the proposed analysis aims at allowing the modeling and formal validation of a spectrum of secure multicast and wireless protocols.

iii) Contrary to previous work in the area, *e.g.*, [1, 3], the proposed analysis is able to check a specification with an unbounded number of components.

Summary. The paper is organized as follows. Section 2 presents the informal description and the formal model of the case studies. Section 3 is dedicated to the analysis of some security properties of the presented protocols. Finally, some conclusions are given.

Even though our effort was to write down a self contained paper, the appendixes report more information regarding the theory behind the application. In particular, they report the syntax of the formal languages used in the paper, they recall the GNDC and tGNDC schemata and they present some proofs.

2 Modeling multicast communication

In this section, we present and formally model some security protocols aiming at ensuring integrity and authenticity of the so called *digital streams*, while

section 3.4 proposes a formal model for distributing a secret key to a multicast group, [27].

Typically, communication involves one sender and an arbitrary number of receivers. We start to describe the Gennaro-Rohatgi protocol (in its off-line version), [10], in order to introduce the reader to the architecture of a simple scheme for signing streams, and to give basic concepts of our modeling.

2.1 The Gennaro-Rohatgi protocol

In [10], Gennaro and Rohatgi developed a mechanism to sign digital streams. They aim at assuring a receiver that the information he received is exactly what the sender has intended.

Applications that deal with streams are typically digitized audio and video, data feeds, applets. This kind of applications requires the user to consume the data it receives at almost the input rate, without excessive delay. For this reason, signing digital streams represents a different problem compared with the signature of finite messages. Traditional digital signature schemes do not fit properly because they require the receiver to process the entire message in order to verify the signature.

The Gennaro-Rohatgi protocol should be considered paradigmatic, being essentially, in its 1997 version, one of the first proposals to efficiently solve the problem to sign digital streams. Efficient cryptographic solutions (*i.e.*, fast to be computed and verified, with respect to the time in which these authors made the proposal) have been adopted in the protocol to allow the entities at stake to minimize their communication and computation overhead.

The authors present two solutions to the problem, distinguishing two cases: i) the off-line case: a finite stream which is entirely known to the sender (*e.g.*, a movie); ii) the on-line case: a potentially infinite stream not known in advance to the sender (*e.g.*, a live broadcast for news feed).

We model the off-line scheme below. For details about the on-line scheme, the reader is referred to [10, 14].

The off-line scheme relies on the basic idea to divide the stream into blocks and to add cryptographic information in each block such that receivers use information retrieved in earlier blocks to legitimate later blocks.

We first use an intuitive notation usually reported in literature. We consider a set of agents able to receive messages. With the following notation,

$$label\ c_j\ A \rightarrow B : msg$$

we represent the transmission of message msg from a sender A to a receiver B. c_j is the j-th communication channel, $label$ is the name of msg.

Thus, let $\{b_i\} \in Msgs$ be the set of meaningful payloads, $i = 1 \ldots l$[1]. $h(m)$ is the digest of m after applying the hash function; $\{m\}_{sk(S)}$ is message m digitally signed by the sender's private key $sk(S)$. Then, the protocol for the off-line case is:

[1] It is assumed that the sender's private key $sk(S)$ does not occur in the set $\{b_i\}$

$$
\begin{array}{ll}
\text{Block } b_0^{'} & c_0\ S \rightarrow R : \{h(b_1^{'})\}_{sk(S)} \\
\text{Block } b_i^{'} & c_i\ S \rightarrow R : b_i, h(b_{i+1}^{'})\ i = 1..l-1 \\
\text{Block } b_l^{'} & c_l\ S \rightarrow R : b_l
\end{array}
$$

It exploits the technique of embedding the hash of the following block in the current block. Bootstrapping integrity of the digital stream is obtained by applying a single traditional signature in combination with hash chaining.

The sender S first divides the stream into l blocks. Then, S generates block $b_0^{'}$, *i.e.*, the digital signature of the hash of the subsequent block $b_1^{'}$. After verification of the signature the receiver knows what the hash of the first block should be and then it starts receiving the full stream (blocks $b_i^{'}$). When the receiver receives the first block $b_1^{'}$, it computes its hash and checks the hash against what the signature was verified upon. The other blocks consist of an authentication chain, in which each block contains the hash of the subsequent block. Note that embedding the hash of the subsequent block implies that the sender knows the stream in advance, hence the non feasibility of this construction for applications like live broadcasts.

It is worth noticing that in the original paper [10], the first block contains an encoding of the length of the stream. The structure of the first block is here simplified (without however affecting the results of the analysis that will follow). Furthermore, we assume the receiver knows in advance the number of blocks in which the stream is divided. It is also worth noticing that to avoid replay attacks when executing multiple runs of the protocol one can simply include nonces in the digitally signed block.

Crypto-CCS specification of the Gennaro-Rohatgi protocol To formally specify the protocol, the sender and the receiver are modelled as Crypto-CCS processes, see App. A. The modelling of possible operations on messages is given by an inference system, consisting of a set of rules r, *e.g.*, . $r = \frac{m_1 \quad \ldots \quad m_n}{m_0}$ where $\{m_1, \ldots, m_n\}$ is a set of messages (called premises, possibly empty) and m_0 is the conclusion.

In particular, a suitable inference system that is used to model the Gennaro-Rohatgi protocol is shown in Fig. 1. Rule $(pair)$ builds the pair of two messages x and y; rules (fst) and (snd) return the components of a pair; rule $(sign)$ allows message x to be digitally signed by applying the secret key $sk(y)$ of agent y; rule (ver) allows a digital signature $\{x\}_{sk(y)}$ to be verified by applying the public key of signer y, $pk(y)$; rule $(hash)$ allows an agent to apply a one-way hash function to message x and obtain digest $h(x)$.

In the following, the application of rule r to messages $m_0, \ldots, m_n$ and a consequent behaviour of the process is denoted as $[m_0, \ldots, m_n \vdash_r x_m]A_1; A_2$, where A_1 and A_2 are Crypto-CCS processes too, and it represents the inference construct. Each conclusion x_m of an inference construct is a message variable and it means "variable x should contain message m". If, by applying rule r to premises $m_0, \ldots, m_n$ a message m can be inferred, then the process behaves as

$$\frac{x \quad y}{Pair(x,y)}(pair) \qquad \frac{Pair(x,y)}{x}(fst) \qquad \frac{Pair(x,y)}{y}(snd)$$

$$\frac{x \quad sk(y)}{\{x\}_{sk(y)}}(sign) \qquad \frac{\{x\}_{sk(y)} \quad pk(y)}{x}(ver) \qquad \frac{x}{h(x)}(hash)$$

Fig. 1. Inference system for the Gennaro-Rohatgi protocol.

A_1, otherwise the process behaves as A_2. When A_2 is missing, if no message m can be inferred, the process aborts. Notation $c!m$ is message m sent on channel c; notation $c?x$ is some message variable $x(m)$ received on channel c. Finally, $\mathbf{0}$ is the process that does nothing.

Example 1. A typical use of the Crypto-CCS inference construct may be as follows, where a process receives a signed message x over channel c and tries to verify the signature. If it succeeds, then the value is sent over channel *out*, otherwise the process outputs an error message *err*.

$c?x.$	*receive x on channel c*
$[x \quad pk \vdash_{ver} y]$	*verify signature*
$out!y.\mathbf{0};$	*in the positive case. output y and stop*
$out!err.\mathbf{0}$	*otherwise, output an error message and stop*

It follows the specification of the Gennaro-Rohatgi protocol.

The sender process builds the initialization block $b_0^{'}$ (more precisely, he builds a variable containing $b_0^{'}$) to bootstrap the chain: by means of inference rules *hash* and *sign* in Fig. 1 the sender computes block $b_0^{'}$, sends it on communication channel c_0 and travels to the next state $Sender_1$. The generic $Sender_i, 1 \leq i < l$ now sends payloads b_i together with hashed blocks $h(b_{i+1}^{'})$ until the last state l is reached.

$Sender_0 \doteq$	
$[b_1^{'} \vdash_{hash} x_{h(b_1^{'})}]$	*Compute hash of next block*
$[x_{h(b_1^{'})} \quad sk(S) \vdash_{sign} x_{b_0^{'}}]$	*Sign computed hash*
$c_0!x_{b_0^{'}}.Sender_1$	*Output* $b_0^{'}$ *and go to next state*
$Sender_i \doteq$	$1 \leq i < l$
$[b_{i+1}^{'} \vdash_{hash} x_{h(b_{i+1}^{'})}]$	*Compute hash of next block*
$[b_i \quad x_{h(b_{i+1}^{'})} \vdash_{pair} x_{b_i^{'}}]$	*Pair payload and hash next block*
$c_i!x_{b_i^{'}}.Sender_{i+1}$	*Output* $b_i^{'}$ *and go to next state*
$Sender_l \doteq$	
$c_l!b_l.\mathbf{0}$	*Output last block and stop*

The receiver process is parameterized by the hashed blocks he receives from the sender (more precisely, variables that should contain the hashed blocks).

$Receiver_0(null) \doteq$
$c_0?x_{b'_0}.$ *Receive initial block*
$[x_{b'_0} \quad pk(S) \vdash_{ver} x_{h(b'_1)}]$ $Verify\ signature$
$Receiver_1(x_{h(b'_1)})$ *Go to next state*

$Receiver_i(x_{h(b'_i)}) \doteq$ $1 \leq i < l$
$c_i?x_{b'_i}.$ *Receive* $i-th$ *block*
$[x_{b'_i} \vdash_{hash} x_{h(b'_{MYi})}]$ *Compute my hash* $h(b'_{MYi})$
$[x_{h(b'_i)} = x_{h(b'_{MYi})}]$ *Compare hash*
$[x_{b'_i} \vdash_{fst} x_{b_i}]$ *Extract payload*
$c_{out_i}!x_{b_i}.$ *Send payload to application level*
$[x_{b'_i} \vdash_{snd} x_{h(b'_{i+1})}]$ *Extract hash of next block and*
$Receiver_{i+1}(x_{h(b'_{i+1})})$ *go to next state*

$Receiver_l(x_{h(b'_l)}) \doteq$
$c_l?x_{b'_l}.$ *Receive last block*
$[x_{b'_l} \vdash_{hash} x_{h(b'_{MYl})}]$ *Compute my hash* $h(b'_{MYl})$
$[x_{h(b'_l)} = x_{h(b'_{MYl})}]$ *Compare hash*
$c_{out_l}!x_{b'_l}.\mathbf{0}$ *Send block to application level and stop*

In the initial state the receiver aims at verifying the digital signature (we assume he has previously retrieved the public key $pk(S)$ corresponding to the private key of the supposed sender). Then, it travels to the next state $Receiver_i(x_{h(b'_1)})$, by maintaining history of the (supposed) next hashed block $h(b'_1)$. Acceptance of the subsequent blocks is conditioned to the successful outcome of the equality tests between the hash it maintains as a parameter and the hash it computes from what it has presently received, respectively $x_{h(b'_i)}$ and $x_{h(b'_{MYi})}$. If the hashes coincide, the receiver sends the meaningful payload contained in x_{b_i} to the application level to consume it. This sending operation is over channel c_{out_i}. The receiver then extracts the supposed hash of the block to be received immediately later. This mechanism is repeated until the reception of the l-th block. Whether the verification of the signature in the initial state or the equality tests in subsequent states do not succeed the receiver should abort.

Extending the model to multiple receivers Extending the model to the treatment of multicast and broadcast communication (*i.e.*, , by allowing a potentially unbounded number of receivers) is as follows: a new process MB is added, that is responsible for potentially sending each block an unbounded number of times in order to simulate a one-to-many (one-to-all) sending typical of a multicast (broadcast) communication. The new process is parameterized by the block

the sender is to multicast (or broadcast).

$$MB_i(x_{b_i'}) \doteq c_i!x_{b_i'}.MB_i(x_{b_i'})$$

Thus, in the light of this new process, the specification for the sender process can be re-written as follows. $P_1||P_2$ denotes a parallel execution of two processes (details in App. A).

$Sender_0 \doteq$
$[b_1' \vdash_{hash} x_{h(b_1')}]$
$[x_{h(b_1')} \quad sk(S) \vdash_{sign} x_{b_0'}]$
$(Sender_1||MB_0(x_{b_0'}))$ *Output* b_0' *and go to next state*

$Sender_i \doteq$ $1 \leq i < l$
$[b_{i+1}' \vdash_{hash} x_{h(b_{i+1}')}]$
$[b_i \quad x_{h(b_{i+1}')} \vdash_{pair} x_{b_i'}]$
$(Sender_{i+1}||MB_i(x_{b_i'}))$ *Output* b_i' *and go to next state*

$Sender_l \doteq MB_l(b_l)$ *Output* b_l *and go to next state*

2.2 The EMSS protocol

Digital streams are usually sent over UDP, the User Datagram Protocol, [25]. UDP is considered to be an unreliable transport protocol, *i.e.*, when UDP sends packets over a network, it just sends them and forgets about them. This does not mean that UDP is ineffective, only that it does not handle reliability of the communication. If a stream is received incomplete, we would still like to be able to prove the integrity of all the packets that were not lost.

Along with the pioneering protocol modelled in the previous section, protocols dealing with the problem of securing streamed data over channels with packet loss have been recently proposed, [23, 21, 12]. They all can be basically considered as valuable extensions of the Gennaro-Rohatgi constructions.

In particular, in [23], Perrig *et al.* presented the Efficient Multi-chained Stream Signature (EMSS) protocol to sign digital streams. EMSS exploits a combination of hash functions and digital signatures and–contrary to previous proposals [10]–achieves (some) robustness against packet loss.

The basic idea of EMSS is the following: a hash of packet P_{i-1} is appended to packet P_i, whose hash is in turn appended to packet P_{i+1} and so on. A signature packet, containing the hash of the final data packet along with a signature, is sent at the end of the stream. To achieve robustness against packet loss (the event of one or more packets loss would break the chain) each packet contains multiple hashes of previous packets and the signature packet signs hashes of multiple packets. [23] uses both deterministic and random distribution of hashes per packet.

Here we focus on a specific instance of the EMSS, viz. the deterministic (1,2) schema, where packet P_i contains hashes of packets $i-1, i-2$ and whose hash is contained in packets $i+1, i+2$. After an initial phase, each packet P_i contains a meaningful payload m_i[2] together with the hashes $h(P_{i-1})$ and $h(P_{i-2})$ of the previous two packets sent. Packets are sent over channels $c_i, 0 \leq i \leq last$ from a sender S to a set of receivers $\{R_n \mid n \geq 1\}$. The end of a stream is indicated by a signature packet P_{sign} over channel c_{sign}, containing the hashes of the final two packets, along with a digital signature. The protocol can formally be described as follows.

$$\begin{array}{ll} \text{Packet } P_0 & c_0\ S \rightarrow \{R_n\} : m_0, null, null \\ \text{Packet } P_1 & c_1\ S \rightarrow \{R_n\} : m_1, h(P_0), null \\ \text{Packet } P_i & c_i\ S \rightarrow \{R_n\} : m_i, h(P_{i-1}), h(P_{i-2})\ 2 \leq i \leq last \end{array}$$

Let P_{last} be the last packet of the stream. Upon sending P_{last} a signature packet is sent:

$$\text{Sign-Pack} \quad P_{sign} \quad c_{sign}\ S \rightarrow \{R_n\} : \{h(P_{last}), h(P_{last-1})\}_{sk(S)}$$

A packet P_i is said to be verifiable if there exists a path (in terms of hashes contained in a chain of packets) from P_i to the signature packet. Given a set of verifiable packets, we intend to prove the correctness of the construction in terms of packet integrity, *i.e.*, to assure a receiver that the information it received is exactly what the sender has originally intended. For the analysis, see Section 3.1.

Crypto-CCS specifications of the (1,2) EMSS. We present the Crypto-CCS specifications of the (1,2) scheme of the EMSS protocol.

We remind that the whole formalization, in particular the way a receiver process acts, is based on implementative choices of the authors since some details are not explicitly given in [23].

A suitable inference system that is used to model EMSS is shown in Fig. 2. Rule $(tuple)$ builds the tuple of three messages x, y and z; rules $(1-st)$, $(2-nd)$ and $(3-rd)$ return, respectively, the first, second and third component of a tuple; rules $(sign)$, (ver) and $(hash)$ are the same as in the inference system for the Gennaro-Rohatgi protocol.

The sender process is parameterized by variables containing the hashes it should insert in the following packet. As in the formalization of the Gennaro-Rohatgi protocol, with notation x_m we mean "variable x should contain message

[2] We assume the sender's private key $sk(S)$ cannot be deduced from the set of messages $\{m_i\}$.

$$\frac{x \quad y \quad z}{(x,y,z)}(tuple) \qquad \frac{(x,y,z)}{x}(1-st) \qquad \frac{(x,y,z)}{y}(2-nd) \; \frac{(x,y,z)}{z}(3-rd)$$

$$\frac{x \quad sk(y)}{\{x\}_{sk(y)}}(sign) \; \frac{\{x\}_{sk(y)} \quad pk(y)}{x}(ver) \qquad \frac{x}{h(x)}(hash)$$

Fig. 2. Inference system for EMSS.

m". Hereafter, state $last+1$ coincides with state $sign$.

$S_0(null, null) \doteq$	
$[m_0 \vdash_{tuple} x_{P_0}]$	*Create tuple* P_0
$[x_{P_0} \vdash_{hash} x_{h(P_0)}]$	*Compute hash of* P_0
$(S_1(x_{h(P_0)}, null) \| \| MB_0(x_{P_0}))$	*Output* P_0 *and go to next state*
$S_1(x_{h(P_0)}, null) \doteq$	
$[m_1 \quad x_{h(P_0)} \vdash_{tuple} x_{P_1}]$	*Create tuple* P_1
$[x_{P_1} \vdash_{hash} x_{h(P_1)}]$	*Compute hash of* P_1
$(S_2(x_{h(P_1)}, x_{h(P_0)}) \| \| MB_1(x_{P_1}))$	*Output* P_1 *and go to next state*
$S_i(x_{h(P_{i-1})}, x_{h(P_{i-2})}) \doteq$	$2 \leq i \leq last$
$[m_i \quad x_{h(P_{i-1})} \quad x_{h(P_{i-2})} \vdash_{tuple} x_{P_i}]$	*Create tuple* P_i
$[x_{P_i} \vdash_{hash} x_{h(P_i)}]$	*Compute hash of current packet*
$(S_{i+1}(x_{h(P_i)}, x_{h(P_{i-1})}) \| \| MB_i(x_{P_i}))$	*Output* P_i *and go to next state*
$S_{sign}(x_{h(P_{last})}, x_{h(P_{last-1})}) \doteq$	
$[x_{h(P_{last})} \quad x_{h(P_{last-1})} \vdash_{tuple} x_t]$	*Create tuple of final hashes*
$[x_t \quad sk(S) \vdash_{sign} x_{P_{sign}}]$	*Sign the tuple*
$MB_{sign}(x_{P_{sign}})$	*Output the signature packet*

Again, the special process MB is responsible for potentially sending each packet an unbounded number of times, in order to simulate a one-to-many (one-to-all) sending. The process is parameterized by the packet the sender is to multicast (or broadcast).

$$MB_i(x_{P_i}) \doteq c_i!x_{P_i}.MB_i(x_{P_i}) \quad 0 \leq i \leq last$$
$$MB_{sign}(x_{P_{sign}}) \doteq c_{sign}!x_{P_{sign}}.MB_{sign}(x_{P_i})$$

Among the set of receivers, each process behaves in the same way. The generic receiver process at step i is parameterized by: 1) the two last packets it received (let them be P_{j_1}, P_{j_2}) - over an ideal channel, without packet loss, we have that $P_{j_1} = P_{i-1}$ and $P_{j_2} = P_{i-2}$; 2) a tuple $tup^{i-1}_{\{m_j\}}$. $tup_{\{m_j\}}$ consists of the ordered sequence of payloads among $\{m_j\}_{j=0,1,\ldots last}$ whose corresponding packets' hashes $h(P_j)$ the receiver was able to check[3]. $tup^{i-1}_{\{m_j\}}$ is the tuple updated

[3] For the sake of readability we assume the receiver may infer the sequence number of a packet by simply observing the packet itself. Otherwise, we should arrange the

at step i, by inserting either $x_{m_{i-2}}$ or $x_{m_{i-3}}$. The updated tuple could be either $(x_{m_{i-2}}, tup^{i-2}_{\{m_j\}})$ or $(x_{m_{i-3}}, tup^{i-2}_{\{m_j\}})$. Also, it may remain unchanged, when both m_{i-2} and m_{i-3} are lost. Similarly, $tup^{last}_{\{m_j\}}$ may either be $(x_{m_{last}}, tup^{last-1}_{\{m_j\}})$ or $(x_{m_{last-1}}, tup^{last-1}_{\{m_j\}})$ or, unchanged, $tup^{last}_{\{m_j\}}$.

The unreliability of the transmission over UDP is modeled by considering that process *Rec* non deterministically chooses whether to receive a packet or not. Finally, we assume that the signature packet P_{sign} is always received (this is likely since in the original protocol multiple copies of the signature packets are sent). In the specification, $2 \leq i \leq last$ and $last + 1 \equiv sign$.

receiver with more parameters or arrange a "sequence number" field in the packet structure and let the receiver retrieve it. This could introduce a too clumsy notation.

$Rec_0(null, null, null) \doteq$
$Rec_1(null, null, null) +$ — *Packet loss : go to next state, otherwise*
$(c_0?x_{P_0}.$ — *Receive initial packet*
$Rec_1(x_{P_0}, null, null))$ — *Go to next state*

$Rec_1(null, null, null) \doteq$
$Rec_2(null, null, null) +$ — *Packet loss : go to next state, otherwise*
$(c_1?x_{P_1}.$ — *Receive packet P_1*
$Rec_2(x_{P_1}, null, null))$ — *Go to next state*

$Rec_1(x_{P_0}, null, null) \doteq$
$Rec_2(null, x_{P_0}, null)+$ — *Packet loss : go to next state, otherwise*
$(c_1?x_{P_1}.$ — *Receive packet P_1*
$[x_{P_1} \vdash_{2-nd} x_{h(P_0)}]$ — *Extract hash of previous packet P_0*
$[x_{P_0} \vdash_{hash} x_{h_{MY}(P_0)}]$ — *Compute my hash $h_{MY}(P_0)$*
$[x_{h(P_0)} = x_{h_{MY}(P_0)}]$ — *Compare the hashes*
$([x_{P_0} \vdash_{1-st} x_{m_0}]$ — *IF equal : extract previous payload*
$Rec_2(x_{P_1}, x_{P_0}, x_{m_0})$ — *Update parameters and go to next state*
$); \mathbf{0}$ — *ELSE abort*
$)$
$Rec_i(x_{P_{j_1}}, x_{P_{j_2}}, tup^{i-1}_{\{m_j\}}) \doteq$
$Rec_{i+1}(x_{P_{j_1}}, x_{P_{j_2}}, tup^{i-1}_{\{m_j\}}) +$ — *Packet loss : go to next state, otherwise*
$(c_i?x_{P_i}.$ — *Receive packet P_i*
$([j_1 = i-1]$ — *Was P_{i-1} received?*
$Rec'_i(x_{P_i}, x_{P_{i-1}}, tup^{i-1}_{\{m_j\}});$ — *Go to Rec'_i; otherwise*
$([j_2 = i-2]$ — *Was P_{i-2} received?*
$Rec''_i(x_{P_i}, x_{P_{i-2}}, tup^{i-1}_{\{m_j\}})$ — *Go to Rec''_i; otherwise*
$)$
$); Rec_{i+1}(x_{P_i}, x_{P_{j_1}}, tup^{i-1}_{\{m_j\}})$ — *Go to next state :*
$)$ — *P_{1-1} and P_{i-2} were not received*
$Rec'_i(x_{P_i}, x_{P_{i-1}}, tup^{i-1}_{\{m_j\}}) \doteq$
$[x_{P_i} \vdash_{2-nd} x_{h(P_{i-1})}]$ — *Extract $h(P_{i-1})$ from P_i*
$[x_{P_{i-1}} \vdash_{hash} x_{h_{MY}(P_{i-1})}]$ — *Compute my hash $h_{MY}(P_{i-1})$*
$[x_{h_{MY}(P_{i-1})} = x_{h(P_{i-1})}]$ — *Compare the hashes*
$([x_{P_{i-1}} \vdash_{1-st} x_{m_{i-1}}]$ — *IF equal : extract m_{i-1} from P_{i-1}*
$Rec_{i+1}(x_{P_i}, x_{P_{j_1}}, (x_{m_{i-1}}, tup^{i-1}_{\{m_j\}}))$ — *Update parameters and go to next state*
$); \mathbf{0}$ — *ELSE : abort*
$Rec''_i(x_{P_i}, x_{P_{i-2}}, tup^{i-1}_{\{m_j\}}) \doteq$
$[x_{P_i} \vdash_{3-rd} x_{h(P_{i-2})}]$ — *Extract $h(P_{i-2})$ from P_i*
$[x_{P_{i-2}} \vdash_{hash} x_{h_{MY}(P_{i-2})}]$ — *Compute my hash $h_{MY}(P_{i-2})$*
$[x_{h_{MY}(P_{i-2})} = x_{h(P_{i-2})}]$ — *Compare the hashes*
$([x_{P_{i-2}} \vdash_{1-st} x_{m_{i-2}}]$ — *IF equal : extract m_{i-2} from P_{i-2}*
$Rec_{i+1}(x_{P_i}, x_{P_{j_1}}, (x_{m_{i-2}}, tup^{i-1}_{\{m_j\}}))$ — *Update parameters and go to next state*
$); \mathbf{0}$ — *ELSE : abort*

$Rec_{sign}(x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}}) \doteq$

$c_{sign}?x_{P_{sign}}.$	*Receive signature packet*
$Rec^{*}_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}})$	*Go to intermediate state* Rec^{*}_{sign}

$Rec^{*}_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{P_{sign}} \quad pk(S) \vdash_{ver} x_{ver}]$	*Verify the signature*
$[j_1 = last]$	*Was* P_{last} *received?*
$Rec'_{sign}(x_{ver}, x_{P_{last}}, tup^{last}_{\{m_j\}});$	*If so, go to* Rec'_{sign}*; otherwise*
$([j_2 = last - 1]$	*Was* P_{last-1} *received?*
$Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup^{last}_{\{m_j\}});$	*If so, go to* Rec''_{sign}*; otherwise*
$(c_{app}!tup^{last}_{\{m_j\}}.\mathbf{0})$	P_{last} *and* P_{last-1} *were not received.*
)	*Send the stream of verifiable payloads to the application level*

$Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{ver} \vdash_{2-nd} x_{h(P_{last-1})}]$	*Extract* $h(P_{last-1})$ *from* P_{sign}
$[x_{P_{last-1}} \vdash_{hash} x_{h_{MY}(P_{last-1})}]$	*Compute my hash* $h_{MY}(P_{last-1})$
$[x_{h_{MY}(P_{last-1})} = x_{h(P_{last-1})}]$	*Compare the hashes*
$[x_{P_{last-1}} \vdash_{1-st} x_{m_{last-1}}]$	IF *equal* : *extract* m_{last-1} *from* P_{last-1}
$c_{app}!(x_{m_{last-1}}, tup^{last}_{\{m_j\}}).\mathbf{0};$	*Send the stream of verifiable payloads*
$\mathbf{0}$	*to the application level and stop*; $ELSE$ *abort*

$Rec'_{sign}(x_{ver}, x_{P_{last}}, tup^{last}_{\{m_j\}}) \doteq$

$[x_{ver} \vdash_{1-st} x_{h(P_{last})}]$	*Extract* $h(P_{last})$ *from* P_{sign}
$[x_{P_{last}} \vdash_{hash} x_{h_{MY}(P_{last})}]$	*Compute my hash* $h_{MY}(P_{last})$
$[x_{h_{MY}(P_{last})} = x_{h(P_{last})}]$	*Compare the hashes*
$[x_{P_{last}} \vdash_{1-st} x_{m_{last}}]$	IF *equal* : *extract* m_{last} *from* P_{last}
$c_{app}!(x_{m_{last}}, tup^{last}_{\{m_j\}}).\mathbf{0};$	*Send the stream of verifiable payloads*
$\mathbf{0}$	*to the application level and stop*; $ELSE$ *abort*

In the final state Rec_{sign} (along with intermediate states $Rec^{*}_{sign}, Rec'_{sign}, Rec''_{sign}$) the receiver aims at verifying the digital signature (we assume it has previously retrieved the public key $pk(S)$ corresponding to the private key of the supposed sender). The correct verification of the signature implies the receiver to have guarantees on the integrity of the verifiable payloads. It can now send the stream to the application level to consume it. In our formalization, this is modeled by a scenario where the receiver sends the content of its parameter tuple (the accepted stream) over channel c_{app}. If the verification of the signature in the final state or the equality tests in the previous states do not succeed the receiver should abort.

2.3 The μTESLA protocol

In [24], Perrig *et al.* presented μTESLA ("micro" Timed Efficient Stream Loss-tolerant Authentication), a protocol to provide authenticated broadcast in wireless sensor networks environments. [24] considers a scenario where sensors communicate with a base-station connected to the external world. The base station may broadcast to all nodes

messages for routing updates, reprogramming, reset requests. The protocol is an extension of the TESLA stream authentication protocol developed in [22] and it was intentionally developed for providing authenticated broadcast for the limited computing environments that are encountered in sensor networks.

In the original TESLA schema, a single sender broadcasts a continuous stream of packets. Receivers may use information in later packets to authenticate earlier packets. Each packet contains a message authentication code (MAC), *i.e.*, a value computed by applying a public algorithm and a secret encryption key to the packet itself. Given a message m and an encryption key k, we call *mac(m, k)* the message authentication code of m. The algorithm is known by all the receivers, while the encryption keys are disclosed by the sender after a certain amount of time. When a receiver receives a key K_i it can use it to compute the MAC from the related packet P_i and compare the computed MAC with that previously received. If the two MACs match, the receiver can consider the packet P_i authentic. To avoid the event that an intruder could use a disclosed key K_i to fake the packet P_i a time synchronization protocol between the sender and the receivers is needed. Then, each receiver will not accept the packet P_i if the sender might have already disclosed the key K_i.

Bootstrapping authentication of the whole scheme is achieved in TESLA by signing the first packet with a regular digital signature scheme. Nevertheless, computation, communication and storage overhead make the use of asymmetric cryptography unfeasible for the net of sensors under investigation. Thus, μTESLA has been proposed as an optimized extension for sensor networks. It just makes use of MACs. The base-station randomly generates the last MAC key to be used, K_{last}, and derives a key chain by repeatedly applying a publicly known one-way function F to that key, such that $K_i = F(K_{i+1})$. Given the non-reversibility property (at least with high probability) of function F, the disclosure of key K_i should not lead to any knowledge of K_{i+1} and subsequent keys.

Receivers' requirements for correctly joining and executing the protocol are: i) they are time synchronized with the base station; ii) they know the disclosure schedule of the MAC keys; iii) they know at least one authenticated key of the key chain, serving as a commitment to the entire chain. A protocol providing time synchronization and one authenticated key has been proposed in [24]. Basically, the base-station shares with each sensor a symmetric secret key K_{SM} and establishes a secure channel over which the exchange of a commitment to the key chain, K_0, and a set of temporal parameters, set_t, takes place[4]. More formally, the initial step of μTESLA is the following:

$$\text{Packet } P_0 \quad c_0\ S \rightarrow \{R_n\} : K_0, set_t, mac(K_0, set_t, K_{SM})$$

where $c_0 \in \{c_i\}_{i \in \mathcal{N}}$, *i.e.*, the set of communication channels, S is the identifier of the sender (*i.e.*, the base station) and $\{R_n\}$ is the set of receivers (*i.e.*, the sensors)[5].

μTESLA is parameterized by the schedule time at which MAC keys are disclosed. For the description of further steps in the protocol we consider a basic formalization, Fig. 3, where we suppose that the sender discloses a MAC key with a delay $\delta = 1$,

[4] There are as many symmetric keys as the number of sensors and the communication over channel c_0 is supposed to be a point to point communication. Nevertheless, to simplify our formalization, we assume a unique key and a unique communication channel. This means to implicitly assume that possible adversaries are not in the set of receivers.

[5] To assure freshness when executing multiple runs of the same sender, one can simply insert nonces in the message authentication code of packet P_0.

assumed to fall in the interval after that key has been used to compute the MAC. Further, we suppose the sender sends one packet per time interval. Basically, in each time slot a packet and a key packet will be sent, see Fig. 3. First of all, each receiver should check the integrity of the received key, say K_i, by verifying it w.r.t. an authenticated commitment (*e.g.*, by checking $K_0 = F^i(K_i)$), then the verified key will be used to verify the integrity of the packet received in the previous time slot.

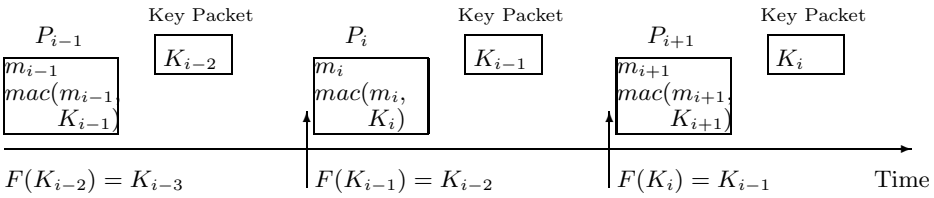

Fig. 3. A μTESLA instantiation.

$$\text{Packet } P_i \quad c_i \; S \rightarrow \{R_n\} : m_i, mac(m_i, K_i) \; i \geq 1$$

Packet P_i consists of a meaningful payload m_i plus the message authentication code computed on m_i with key K_i. We assume that K_{SM} cannot be deduced from the sets $\{m_i\}, \{K_i\}$.

Upon receiving the packet, the sensor stores the packet until its MAC can be verified, *i.e.*, until the sender broadcasts packet disclosing K_i:

$$\text{Key-Packet } KP_i \quad c_{i+1} \; S \rightarrow \{R_n\} : K_i$$

The integrity of key K_i can be checked by verifying $K_0 = F^i(K_i)$ (or, equivalently, $K_{i-1} = F(K_i)$). Packets may be lost in transit from the base station to the sensors. In particular μTESLA is tolerant to packet loss in the sense that receivers may still be able to authenticate all the received packets P_i even when the corresponding keys' disclosure packets are lost. Suppose K_j is lost, then a receiver is not able to verify MAC packet P_j. The following key the receiver recovers, let it be K_{j+1}, can be verified w.r.t. a previous authenticated key (*e.g.*, $K_0 = F^{j+1}(K_{j+1})$) and is used to derive K_j, i.e. $K_j = F(K_{j+1})$.

The *tCryptoSPA* specifications of the μTESLA protocol Part of the complexity in the construction of protocols like μTESLA consists of the temporal constraints that are present, since, *e.g.*, a time synchronization is needed among the actors in the protocol.

Within a formal framework aimed at modeling timed constraints in protocols and at verifying security properties whose fulfilment is based on timed conditions, we give here the *tCryptoSPA* specification of the basic μTESLA presented in Fig. 3.

Indeed, the fundamental requirement of a time synchronization between the base station and each sensor in μTESLA is naturally captured in *tCryptoSPA* (Subappendix A.2) by its time modeling action *tick*, upon which sender and receivers' pro-

cesses may synchronize (this allows us to avoid the explicit presence of set_t in packet P_0).

A suitable inference system that is used to model μTESLA is shown in Fig. 4. Rule $(one-way)$ allows to apply a one-way hash function F to message m and obtain digest $F(m)$; rule mac computes the message authentication code (MAC) of a message with a key; rules $(pair)$, (fst) and (snd) are the same as in the inference system for the Gennaro-Rohatgi protocol.

$$\frac{m \quad m'}{(m,m')}(pair) \qquad \frac{(m,m')}{m}(fst) \quad \frac{(m,m')}{m'}(snd)$$

$$\frac{F \quad m}{F(m)}(one-way) \quad \frac{m \quad k}{mac(m,k)}(mac)$$

Fig. 4. Inference system for μTESLA.

We consider a sender machine with ample resources. It can be parallelized or split into n senders, each of them possibly sending different streams, $\{m_i^j\}_{i\geq 1, 1\leq j\leq n}$. We first present the generic sender process S^j, parameterized by an arbitrary but finite sequence of MAC keys (tied together by means of a key chain)[6]. We assume the symmetric key K_{SM}, the keys belonging to the key chain and the streams of packets to be different for each process $S^j, 1 \leq j \leq n$[7].

$$\begin{array}{ll} S_0^j(K_{SM}^j, K_0^j, K_1^j, \ldots) \doteq & \\ \quad [K_0^j \quad K_{SM}^j \vdash_{mac} y] & \textit{Compute MAC} \\ \quad [K_0^j \quad y \vdash_{pair} P_0] & \textit{Create packet } P_0 \\ \quad B_0^j(P_0) & \textit{Start to broadcast } P_0 \end{array}$$

$$\begin{array}{ll} S_1^j(K_0^j, K_1^j, \ldots) \doteq & \\ \quad [m_1^j \quad K_1^j \vdash_{mac} x] & \textit{Compute MAC} \\ \quad [m_1^j \quad x \vdash_{pair} P_1] & \textit{Create packet } P_1 \\ \quad B_1^j(P_1) & \textit{Start to broadcast } P_1 \end{array}$$

$$\begin{array}{ll} S_i^j(K_{i-1}^j, K_i^j, \ldots) \doteq & \\ \quad [m_i^j \quad K_i^j \vdash_{mac} x] & \textit{Compute MAC} \\ \quad [m_i^j \quad x \vdash_{pair} P_i] & \textit{Create packet } P_i \\ \quad B_i^j(P_i, K_{i-1}^j) & \textit{Start to broadcast } P_i \textit{ and disclose key } K_{i-1} \end{array}$$

[6] Actually, we consider constants with an arbitrary number of parameters. We could avoid this by considering, for modeling purposes, a special function fun, not available to possible adversaries, that may be used to represent the keys as a sequence.

[7] We remind the reader that the whole formalization we are going to give is based on personal choices since some details are not explicitly given in [24]. In particular, the mechanism through which a receiver possibly identifies each sender process (and consequently each stream) is not defined in [24], since the original construction is described with a single sender.

$$B_i^j(P_i) \doteq \overline{c_i}P_i.B_i^j(P_i) + tick.S_{i+1}^j(K_i^j, \ldots) \quad i = 0, 1$$
$$B_i^j(P_i, K_{i-1}^j) \doteq \overline{c_i}P_i.\overline{c_i}K_{i-1}^j.B_i^j(P_i, K_{i-1}^j) + tick.S_{i+1}^j(K_i^j, \ldots) \quad i \geq 2$$

Construct $B_i^j(\ldots)$ is responsible for potentially sending packets (and keys) an unbounded number of times, in order to simulate multicast sessions. Sender S^j remains in the same state repeatedly sending messages unless the non-deterministic choice is resolved by choosing the derivative of the second summand in B_i^j; this causes a time unit to pass (a *tick* action is performed). The construction models the behaviour of a wireless antenna making signals available only in a particular time interval. The presence of a non-deterministic choice in the construct makes it possible the passage to the following time interval without performing any number of, possibly zero, communication. This may implicitly model the unreliability of the wireless transmission and the occurrence of packet loss.

Among the receivers' set, each process behaves in the same way. The generic receiver process at step i is parameterized by a commitment to the key chain (let it be K_0^j) and by the packets it should still authenticate. We assume the receiver's set is divided into subgroups, each of them sharing a particular K_{SM} with one sender process. Sender S^j and receivers belonging to subgroup number j share K_{SM}^j. K_{SM}^j may denote a particular service each element in subgroup j is devoted to. Let us consider pay per view-based applications: among the receivers' set, the subgroup knowing K_{SM}^j may consist of all the paying spectators for movie number j. For environments closer to those depicted for μTESLA, let us consider a scenario in which sensors are used to periodically transmit readings regarding heating and air conditioning control in a building (and consequently receive broadcasted messages for routing updates or reprogramming): sensors in subgroup j may be all the sensors devoted to carry out the service for room number j. (S^j being the base station responsible for room number j.).

Below, we refer to $R_i^{j,q}$ to indicate the q-th receiver process belonging to subgroup j and acting at step i.

$R_0^{j,q}(null) \doteq$
$c_0(x).$ *Receive first packet*
$[x \vdash_{fst} x_{K_0}]$ *Extract commitment to the key chain*
$[x_{K_0} \quad K_{SM}^j \vdash_{mac} z]$ *Compute MAC*
$[x \vdash_{snd} x_{mac}]$ *Extract MAC*
$[z = x_{mac}]$ *Verify MAC : if verified :*
$tick.R_1^{j,q}(x_{K_0});$ *Allow a time unit to pass and go to next state;*
$R_0^{j,q}(null)$ *Otherwise, wait for the correct key*

Upon receiving a value x on channel c_0, the receiver verifies the correctness of the commitment to the key chain, x_{K_0}: he computes $mac(x_{K_0}, K_{SM}^j)$ and he compares it with the message authentication code in the received packet. If the two MACs match, a time unit passes and the receiver goes to the next state, otherwise the receiver remains in the same state waiting for the right key K_{SM}^j.

Throughout the formalization, *null* means an empty field.

$R_1^{j,q}(x_{K_0}) \doteq$
$(c_1(y).$ *Receive packet and*
$tick.R_2^{j,q}(y, x_{K_0})$ *Allow a time unit to pass and go to next state*
$) + tick.R_2^{j,q}(null, x_{K_0})$ *Or : go to next state after a time unit*

$R_1^{j,q}$ is willing to accept any arbitrary packet, because it cannot perform any verification yet. If nothing is received before the end of a time unit, a transition takes place to next state $R_2^{j,q}$.

$$
\begin{array}{l}
R_i^{j,q}(p_{i-1}, x_{K_0}) \doteq \\
\quad c_i(p_i).R_i'^{j,q}(p_i, p_{i-1}, x_{K_0}) \quad \textit{Receive } i-\textit{th packet; go to intermediate state } R_i'^{j,q} \\
\quad +tick.R_{i+1}^{j,q}(null, x_{K_0}) \quad \textit{Or : go to next state after a time unit}
\end{array}
$$

$R_i^{j,q}$ is willing to accept packet P_i and travels to an intermediate state $R_i'^{j,q}$. If nothing is received before the end of a time unit, a transition takes place to the next state.

$$
\begin{array}{ll}
R_i'^{j,q}(p_i, p_{i-1}, x_{K_0}) \doteq & \\
\quad c_i(x_{K_{i-1}}). & \textit{Receive key packet} \\
\quad [x_{K_0} = F^{i-1}(x_{K_{i-1}})] & \textit{Verify the key w.r.t. the commitment} \\
\quad [p_{i-1} \vdash_{fst} y_{pay}] & \textit{Extract payload} \\
\quad ([y_{pay} \quad x_{K_{i-1}} \vdash_{mac} z] & \textit{If } x_{K_{i-1}} = K_{i-1}^j \textit{ then : Compute MAC} \\
\quad\quad [p_{i-1} \vdash_{snd} y_{mac}] & \textit{Extract MAC} \\
\quad\quad [z = y_{mac}] & \textit{Verify MAC} \\
\quad\quad \overline{app} y_{pay}. & \textit{Send } m_1^j \textit{ to application level} \\
\quad\quad tick.R_{i+1}^{j,q}(p_i, x_{K_0}) & \textit{Allow a time unit to pass and go to next state;} \\
\quad); R_i'^{j,q}(p_i, p_{i-1}, x_{K_0}) & \textit{Otherwise, wait for the correct key}
\end{array}
$$

In intermediate state $R_i'^{j,q}$ receives a key packet and verifies the correctness of the key w.r.t. the authenticated commitment $x_{K_0} = K_0^j$. Given the collision-free property of one-way functions, if the verification does not succeed it means $x_{K_{i-1}} \neq K_{i-1}^j$ and $R_i'^{j,q}$ simply stays in the same state waiting for the right subgroup key. If the verification succeeds, the correctness of P_{i-1} is verified by checking that the enclosed MAC is authentic. The successful outcome is here modeled by a scenario where the receiver sends the payload of the accepted packet over channel app[8].

Suppose packet P_{i-1} was correctly received, suppose also packet disclosing K_{i-1}^j is lost. At step i the receiver still cannot authenticate packet P_{i-1}. The key chain mechanism of the original protocol takes into account such a possibility: in interval $i+1$ the base station broadcasts key K_i^j, which the receiver authenticates by verifying $K_0^j = F^i(K_i^j)$. The receiver can authenticate P_i and derives $K_{i-1}^j = F(K_i^j)$, so it can also authenticate P_{i-1}. Actually, our formalization does not take into account recovering lost keys. For the sake of simplicity, we prefer to suppose that the key packet related to subgroup j is received (state $R_i'^{j,q}$).

We report below the formalization at step i, with $i \geq 2$, when a packet was not received at step $i-1$.

$$
\begin{array}{l}
R_i^{j,q}(null, x_{K_0}) \doteq \\
\quad c_i(p_i).tick.R_{i+1}^{j,q}(p_i, x_{K_0}) \quad \textit{Receive } i-\textit{th packet; go to next state} \\
\quad +tick.R_{i+1}^{j,q}(null, x_{K_0}) \quad \textit{Or : go to next state after a time unit}
\end{array}
$$

[8] We omitted to insert an idling behavior when a deduction construct fails to be executed and in our formalization the system simply stops without letting time pass. This is not realistic, but it has no consequences since we use trace semantics for the analysis and makes it simpler.

2.4 The N Root/Leaf pairwise keys protocol

Secrecy in multicast groups means that only the group members (and all of them) should be able to decipher transmitted data ([4]).

To achieve secrecy, the approach presented in [27] is a "brute force method to provide a common multicast group key to the group participants".

The N Root/Leaf pairwise keys protocol assumes the existence of a multicast session with an initiator that controls the multicast group. Each of the N members of the multicast group is called a leaf. The initiator is the root of the group. In a preliminary phase, the initiator generates a pairwise key with each of the leaves in the multicast group (*e.g.*, using some standard public key exchange technique).

Then, it generates the group key K and, in order to distribute it to the leaves, the initiator encrypts K with the pairwise keys shared with them. This distribution can happen through a transmission to the whole group via multicast (Packet 1). On receiving Packet 1, each leaf can retrieve K from the appropriate segment of the message using its own secret pairwise key.

Once the group key has been distributed, it can be used to multicast to the group ciphered messages m (Packet 2).

$$\begin{array}{ll} \text{Packet 1} & c_1\ I \rightarrow \{L_n\} : \{K\}_{K_{IL_1}} | \{K\}_{K_{IL_2}} | \ldots | \{K\}_{K_{IL_N}} \\ \text{Packet 2} & c_2\ I \rightarrow \{L_n\} : \{m\}_K \end{array}$$

In the above notation, I is the initiator of the multicast group. $\{Ln\}$ is the set of the N leaves. | stands for concatenation. K_{IL_i} is the pairwise key shared between the initiator and the i-th leaf. K is the group key.

The Crypto-CCS specification of the protocol is the following. The initiator process is parameterized by the group key and by the message to convey to the group.

$$\begin{array}{ll} I_1(k,m) \doteq & \\ [k \quad k_{IL1} \vdash_{enc} \{k\}_{k_{IL_1}}] & \textit{Encrypt group key} \\ \ldots & \textit{Repeat encryption N times} \\ [\{k\}_{k_{IL_1}} \ldots \{k\}_{k_{IL_N}} \vdash_{tuple} x_{P_1}] & \textit{Create tuple} \\ I_2(k,m)||MB_1(x_{P_1}) & \textit{Output } P_1 \textit{ and go to next state} \end{array}$$

$$\begin{array}{ll} I_2(k,m) \doteq & \\ [m \quad k \vdash_{enc} x_{k_m}] & \textit{Encrypt message} \\ MB_1(x_{k_m}) & \textit{Output } x_{k_m} \end{array}$$

Process MB simulates a one-to-many sending and it is specified as follows.

$$MB_i(x) \doteq c_i!x.MB_i(x) \quad i = 1,2$$

The specification of the n-th receiver process is $L_1^n(K_{IL_n})$:

$$\begin{array}{ll} L_1^n(K_{IL_n}) \doteq & 1 \leq n \leq N \\ c_1?x. & \textit{Receive concatenation} \\ [x \vdash_{nth} z] & \textit{Retrieve encryption} \\ [z \quad K_{IL_n} \vdash_{dec} x_K] L_2^n(K_{IL_n,x_K}) & \textit{Decrypt and go to next state} \end{array}$$

$$\begin{array}{ll} L_2^n(K_{IL_n}, x_K) \doteq & \\ c_2?x. & \textit{Receive encrypted message} \\ [x \quad x_K \vdash_{dec} x_m].\mathbf{0} & \textit{Retrieve } m \end{array}$$

3 Analysis

In this section, we perform a security analysis of the protocols presented in the previous sections. In particular, we consider two integrity properties, one in an untimed version and one in a timed version, for what concerns, respectively, the EMSS protocol and the μTESLA protocol, while a secrecy property will be took into account for a study on the N Root/Leaf pairwise protocol. As far as the analysis of an integrity property of the Gennaro-Rohatgi protocol is concerned, here we will limit ourselves to recall the guidelines of the procedure, since it is very similar to what has be done for EMSS.

For details about the analysis methodology, the reader is referred to the appendixes, as well as to several references cited throughout the paper. However, aiming at producing, as much as possible, a self contained paper, we recall here the general flavor of the methodology (in the untimed version only).

The foundation of the analysis is the seminal idea of *non interference*, [11] for investigating the unauthorized information flow in multilevel systems, *e.g.*, from a high level to a lower one. By starting from there, a general schema for the definition of security properties has been formulated, [6, 9, 8], in order to encompass in a uniform way a variety of properties. The schema, namely Generalized Non Deducibility on Compositions, GNDC for short, basically compares what it is expected to be the correct behaviour of a system with a modified behaviour due to the fact that the system is not running in isolation, but it is running together with a malicious process, the so called intruder, trying to interfere with the normal execution of the system. If the two behaviors appear to be the same, then it means that the intruder has not sufficient means to significantly interfere with the honest system and that the investigated property is guaranteed.

More formally, a system P satisfies property $GNDC^{\alpha}_{\lhd}$ if the behavior of P, despite the presence of a hostile environment X that can interact with P only through a fixed set of channels C, *appears* to be same (w.r.t. a behavioral relation $\lhd$ of observational equivalence) to the behavior of a modified version α of P that represents the *expected* (correct) behavior of P. As behavioral relation between processes we consider hereafter the trace inclusion relation $\leq_{trace}$ (App. A).

The formula expressing the GNDC schema is as follows:

Definition 1. *Given a behavioral relation $\lhd$ between processes, $\lhd : \mathcal{P} \to \mathcal{P}$, a function α between processes, $\alpha : \mathcal{P} \to \mathcal{P}$, and a set $\mathcal{E}_C^{\phi_X}$ of all the admissible hostile processes, (App. B), we say that a process $P \in GNDC^{\alpha}_{\lhd} \iff \forall X \in \mathcal{E}_C^{\phi_X} : (P||X) \setminus C \lhd \alpha(P)$.*

Basically, what we are going to do in the following subsections is, for each protocol: 1) first, to define, as a Crypto-CCS process, the correct behaviour that the system P should have with respect to the security property to be investigated (*e.g.*, in the next subsection, α_{int} will denote the correct behaviour of EMSS with respect to integrity); 2) then, to verify that the behaviour of system P, when considering just one sender plus the intruder and one receiver plus the intruder, is included in the defined correct behaviour (this is done accordingly to the GNDC theory); 3) finally, to exploit compositional principles in order to assert the validity of the property within the whole system.

A compositional principle (in its untimed version) is the following:

Definition 2. STABILITY OF A PROCESS. *We say that a process P is stable w.r.t. ϕ_X if, whenever $(P||X_{\phi_X}) \setminus C \stackrel{\gamma}{\Longrightarrow} (P'||X'_{\phi'_X}) \setminus C$, then $\mathcal{D}(\phi_X) = \mathcal{D}(\phi'_X)$.*

This was introduced in [13]. We denote the set of messages initially known by process X as ϕ_X. $\mathcal{D}(\phi'_X)$ is a set of messages representing what can be inferred by X at the end of a certain computation γ run in parallel with process P, while $\mathcal{D}(\phi_X)$ represents what can be inferred with ϕ_X solely. Basically, process P is stable when X does not increase significantly ϕ_X during the execution of P.

When two (or more) processes are stable with respect to a certain knowledge ϕ_X, and they enjoy a certain GNDC property, the following compositionality proposition holds ([13]).

Proposition 1. *Given ϕ_X and a set of public channels C, assume processes $P_r \in GNDC^{\alpha_r(P_r)}_{\leq trace}$ with $1 \leq r \leq n$ and P_r stable w.r.t. ϕ_X. It follows that $(P_1|| \ldots ||P_n)$ is stable w.r.t. ϕ_X and $(P_1|| \ldots ||P_n) \in GNDC^{\alpha_1(P_1)||\ldots||\alpha_n(P_n)}_{\leq trace}$.*

Also, a timed version of GNDC, namely $tGNDC$, a timed version of the stability principle, [15, 13], and a timed version of the proposition of compositionality of a GNDC property exist, [13]. They will be used in the analysis of μTESLA.

3.1 An analysis of the EMSS protocol: integrity

The specification of the (1,2) EMSS has been given in Subsection 2.2. Here, we perform a protocol analysis for verifying the integrity of the packets received by the receiver process.

Integrity for EMSS is defined within the GNDC schema as the ability to accept only the message m_i by a receiver as the $i-th$ message sent by the sender (assuming m_i is not lost). Let us assume that a receiver signals the acceptance of a stream of messages as a legitimate one, by issuing it, as a unique list of messages, on a special channel c_{app}. Thus, let α_{int} be the Crypto-CCS process $Spec_{sign} = \sum_{s \in streams} c_{app}!s.\mathbf{0}$, where $streams$ is the set of all the possible ordered sub streams of $m_0 \ldots m_{last}$.

Definition 3. *A system P, consisting of a sender of a stream of messages $\{m_i\}$ and a receiver, enjoys the integrity property whenever $P \in GNDC^{\alpha_{int}}_{\leq trace}$.*

Basically, integrity holds when the receiver accepts exactly a subset of the messages m_i in the correct order even in presence of an adversary. The key point is that the intruder will never acquire the private key of the sender to successfully sign the final packet of the stream.

In a multi-receiver setting with one sender, a protocol guarantees integrity whenever each receiver accepts only the stream of messages that the sender wishes to deliver. In our case, the specification for n receivers is simply the parallel composition of α_{int} n-times.

The first part of the analysis consist of verifying the stability of the involved sender and receiver processes. S_0, Rec_0 are stable w.r.t. the following initial knowledge ϕ_X:

$$\phi_X = \{P_0\} \cup \{P_1\} \cup \{P_i \mid i = 2, \ldots, last\} \cup \{pk(S), P_{sign}\}$$

This can be proved by looking at the specifications of S_0 and Rec_0 given in Subsection 2.2.

The initial knowledge ϕ_X includes indeed all the messages an adversary would be able to add to its knowledge by eavesdropping on a run of the protocol (in other words, X does not increase its knowledge when S_0 and Rec_0 run). This implies that the considered intruder has the most powerful means to act since the beginning of the computation. One may comment that this is not correct, since it does not follow the reality. On the other hand, this is only a trick in the model, and, if the protocol satisfies the integrity property in this very hostile environment, then it means that it will satisfy this property in a less powerful one. This may be formally justified, [9]. Here, we prefer to give an informal discussion of the matter: let us suppose that there exists a sequence of actions, leading to an attack w.r.t. a procedure, performed by an intruder whose initial knowledge is ϕ. Then, let us suppose that the intruder knows ϕ', with $\phi \in \phi'$. Again, there will be at least the attack found starting from ϕ. On the other hand, if no attack exists with ϕ', one may reasonably conclude that no attack will exist by starting from a subset ϕ of ϕ'.

Now, we check if the specifications of the sender and the receiver, separately, satisfy the integrity property. We can prove that S_0 enjoys $GNDC^{\mathbf{0}}_{\leq_{trace}}$ and Rec_0 enjoys $GNDC^{\alpha_{int}}_{\leq_{trace}}$, that is to say that, for all $X \in \mathcal{E}_C^{\phi_X}$, we have $(S_0||X)\backslash C \leq_{trace} \mathbf{0}$ and $(Rec_0||X)\backslash C \leq_{trace} \alpha_{int}$. This may be done by finding a suitable weak simulation relation between $(S_0||X) \setminus C$ and $\mathbf{0}$, and between $(Rec_0||X) \setminus C$ and $Spec_{sign}$ ($\forall X \in \mathcal{E}_C^{\phi_X}$), respectively. (The easier way is to prove the same with one check, by simply considering the top element Top_ϕ^C).

Let $C = \{c_{sign}\} \cup \{c_i \mid 0 \leq i \leq last\}$ be the set of channels over which each element of set $\mathcal{E}_C^{\phi_X}$ is able to communicate.

The candidate weak simulation relation we consider for dealing with the sender specifications is the following:

$$\begin{aligned}\mathcal{R}_{\mathcal{S}} = (&((S_i(...)||X)\backslash C, \mathbf{0}) \mid X \in \mathcal{E}_C^{\phi_X}, 0 \leq i \leq last)\\ &\cup(((S_{sign}(...)||X)\backslash C, \mathbf{0}) \mid X \in \mathcal{E}_C^{\phi_X})\end{aligned}$$

The candidate weak simulation relation we consider for dealing with the receiver specifications is the following:

$$\begin{aligned}\mathcal{R}_{\mathcal{R}} = &(((Rec_0(null, null, null)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec_1(null, null, null)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec_1(x_{P_0}, null, null)||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec_i(x_{P_{j1}}, x_{P_{j2}}, tup^{i-1}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last)\\ &\cup(((Rec'_i(x_{P_i}, x_{P_{i-1}}, tup^{i-1}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last)\\ &\cup(((Rec''_i(x_{P_i}, x_{P_{i-2}}, tup^{i-1}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X}, 2 \leq i \leq last)\\ &\cup(((Rec_{sign}(x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec^*_{sign}(x_{P_{sign}}, x_{P_{j1}}, x_{P_{j2}}, tup^{last}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec'_{sign}(x_{ver}, x_{P_{last}}, tup^{last}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\\ &\cup(((Rec''_{sign}(x_{ver}, x_{P_{last-1}}, tup^{last}_{\{m_j\}})||X)\backslash C, Spec_{sign}) \mid X \in \mathcal{E}_C^{\phi_X})\end{aligned}$$

$tup^{i-1}_{\{m_j\}}, tup^{last}_{\{m_j\}}$ are lists of meaningful payloads (also updated). By inspection of the possible cases we may show that $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}_{\mathcal{R}}$ are weak simulations. We omitted to explicitly put in $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}_{\mathcal{R}}$ the pairs in which the first process performs deduction constructs.

We give a sketch of the proof dealing with the receiver specification. When the first process performs inference (or match) constructs and it gets stuck because an inference rule does not apply, or it simply travels to the next state, it can be weakly simulated by whatever process, in particular $Spec_{sign}$. When Rec_0 performs a receiving action, the process on the left may perform a τ action and it can be weakly simulated by whatever process, in particular $Spec_{sign}$. The significant case is when the first process outputs a tuple of messages $tup_{\{m_j\}}$ over channel $c_{app} \notin C$. In this case, it must be $\{x_{ver}\}_{sk(S)} = P_{sign}$ and, assuming that digital signatures and hash functions cannot be forged, all the messages in $tup_{\{m_j\}}$ must be replaced with one of all the possible ordered sub streams of $m_0 \dots m_{last}$. This can be weakly simulated by $Spec_{sign}$ that has been defined as the process sending all the possible ordered sub streams of $m_0 \dots m_{last}$.

Each resulting pair consisting of the derivatives still belong to $\mathcal{R}_{\mathcal{R}}$.

Proposition 2. *$S_0 \in GNDC^{\mathbf{0}}_{\leq trace}$ and $Rec_0 \in GNDC^{\alpha_{int}}_{\leq trace}$.*

The following proposition follows by the fact that S_0, Rec_0 are stable w.r.t. ϕ_X, by Proposition 1 and Proposition 2.

Proposition 3. *$S_0||Rec_0 \in GNDC^{\alpha_{int}}_{\leq trace}$.*

Then, the following statement holds because Proposition 1 is applicable once again.

Proposition 4. *The (1,2) EMSS Protocol enjoys integrity for whatever number of receivers.*

To check a system with an arbitrary number of components, what we do is simply consider the components separately. The result follows by Proposition 1 where index r is not fixed *a priori* and $P_1 = S_0$ and $P_r, 2 \leq r \leq n$ is Rec_0.

3.2 Hints to an analysis of the Gennaro-Rohatgi protocol: integrity

In [14], the stability principle of Def. 2 and the compositionality proposition Prop. 1 have been applied also to the Gennaro-Rohatgi scheme. The steps of the analysis are very similar to those presented for the EMSS case study.

The correct behaviour of the system was specified to be $\alpha_{int} = Spec_1$ where $Spec_i = c_{out_i}!b_i.Spec_{i+1}$ with $1 \leq i \leq l-1$, and $Spec_l = c_{out_l}!b_l.\mathbf{0}$.

$Sender_0$ and $Receiver_0$, specified in Subsection 2.1, are stable with respect to the following initial knowledge ϕ:

$$\phi = \{pk(S), b'_0\} \cup \{b'_i, b_i, h(b'_i) \mid i = 1, \dots, l-1\} \cup \{b'_l, b_l\}$$

Then, it is possible to find suitable weak simulation relations between $(Sender_0||X)\backslash C$ and $\mathbf{0}$ and between $(Receiver_0||X) \setminus C$ and $Spec_1$, respectively.

Finally, one can apply Proposition 1 to prove integrity on whatever number of receivers.

3.3 An analysis of the μTESLA protocol: timed integrity

The specification of the μTESLA protocol has been given in Subsection 2.3. Here, we perform an analysis of the protocol concerning one of its timed security properties. To do this, we use the timed version of the GNDC scheme, namely tGNDC, App. B. The definition of tGNDC is similar to Def. 1, provided that one could consider a timed behavioral relation between processes, timed functions between processes expressing the expected correct behaviour and a set of timed admissible hostile processes.

So called timed integrity belongs to a new class of security properties defined in [13]. A stream signature protocol guarantees timed integrity on a set of messages $\{m_i\}$ if, whenever the generic receiver accepts an item in a time interval i, let us say item x, then $x = m_{i-\delta}$, $i - \delta$ being the time interval in which x has been received. ($\delta = 1$ in the formalization of μTESLA given in Subsection 2.3).

In μTESLA, let us assume that a receiver signals the acceptance of a payload as a legitimate one, by issuing it on a special channel *app*.

Let $P^q \doteq S_0^j || R_0^{j,q}$ be the system consisting of a single sender and the q-th receiver in subgroup j, sharing K_{SM}^j. Thus, we define the correct behaviour of the system P^q to be the $tCryptoSPA$ process $\alpha_{tInt}(P^q) \doteq tSpec_0$, where

$$
\begin{array}{l}
tSpec_0 \doteq tick.tSpec_1 \\
tSpec_1 \doteq tick.tSpec_2 \\
tSpec_i \doteq tick.tSpec_{i+1} + \overline{app}(m_{i-1}^j).tick.tSpec_{i+1} \; i \geq 2
\end{array}
$$

In the first two steps, $\alpha_{tInt}(P^q)$ simply let time pass, while in further steps it may either let time pass (denoting packet loss) or let a verified payload to be sent on the special channel *app* and then let time pass. The set of all messages sent on channel *app* is the set of all the possible ordered substreams of $\{m_i^j\}_{i \geq 1}$. Let function $\alpha_{tInt}^j(P^j) \doteq \Pi_{1 \leq q \leq n_j} \alpha_{tInt}(P^q)$, n_j being the cardinality of the receivers in subgroup j.

Definition 4. *The system $P^j \doteq S_0^j || R_0^{j,1} || R_0^{j,2} || \ldots || R_0^{j,n_j}$, consisting of a sender of streamed data $\{m_i^j\}$ and the receivers in subgroup j enjoys the timed integrity property whenever $P^j \in tGNDC_{\leq ttrace}^{\alpha_{tInt}^j(P^j)}$.*

Basically, it means that each receiver accepts exactly the messages belonging to $\{m_i^j\}$ in the correct order and within the time interval following the one in which the sender actually sent the messages, even in presence of an intruder (unless packets P_i are lost). The key point is that the intruder will never acquire the shared key K_{SM}^j to establish a secure channel over which the commitment to the key chain is exchanged[9].

For the analysis of timed security properties, we use a refined notion of stability, called time-dependent stability ([15, 13]).

We let γ be a sequence of actions (possibly empty) ranging over $Act \backslash \{\tau\}$. Let $\#^{tick}(\gamma)$ be the number of occurrences of *tick* actions in the sequence γ.

Definition 5. *We say that a process P is time-dependent stable w.r.t. the sequence of knowledges $\{\phi_i\}_{i \geq 0}$ if, whenever $(P||X_{\phi_0}) \backslash C \stackrel{\gamma}{\Longrightarrow} (P'||X_{\phi'}') \backslash C$ and $\#^{tick}(\gamma) = i$, then $\mathcal{D}(\phi') = \mathcal{D}(\phi_i)$.*

[9] We remind the reader that $K_{SM}^m \neq K_{SM}^n$ if $m \neq n$ and $K_i^m \neq K_l^n$ if $m \neq n$ or $i \neq l$.

The concept of time-dependent stability is similar to the one of stability introduced in Section 3. Basically, a process P is time-dependent stable if process X cannot increase significantly its knowledge when P runs in the space of a time slot.

When two (or more) processes are t. d. stable with respect to a certain sequence of knowledges $\{\phi_i\}_{i\geq 0}$, and they enjoy a certain tGNDC property (Appendix B), the following compositionality proposition holds (proofs in [13]).

Proposition 5. *Given a sequence $\{\phi_i\}_{i\geq 0}$ and a set of public channels C, assume $P_r \in tGNDC^{\alpha_r(P_r)}_{\leq_{ttrace}}$ with $1 \leq r \leq n$. Assume also P_r t. d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$. It follows that $(P_1||P_2||\ldots||P_n) \in tGNDC^{\alpha_1(P_1)||\alpha_2(P_2)||\ldots||\alpha_n(P_n)}_{\leq_{ttrace}}$ and $(P_1||P_2||\ldots||P_n)$ is t. d. stable w.r.t. $\{\phi_i\}_{i\geq 0}$.*

S_0^j and $R_0^{j,q}$ (Subsection 2.3) are t.d. stable w.r.t. the sequence $\{\phi_i\} = \phi_0, \phi_1, \phi_2, \ldots$ defined as follows:

$$\begin{aligned}
&\phi_0 = \{K_0^j, mac(K_0^j, K_{SM}^j) \mid 1 \leq j \leq n\}\\
&\phi_1 = \phi_0 \cup \{m_1^j, mac(m_1^j, K_1^j) \mid 1 \leq j \leq n\}\\
&\phi_2 = \phi_1 \cup \{m_2^j, mac(m_2^j, K_2^j), K_1^j \mid 1 \leq j \leq n\}\\
&\ldots\\
&\phi_i = \phi_{i-1} \cup \{m_i^j, mac(m_i^j, K_i^j), K_{i-1}^j \mid 1 \leq j \leq n\}\\
&\ldots
\end{aligned}$$

where n is the number of senders. This can be verified by inspection of the specifications in Subsection 2.3.

ϕ_i is equal to ϕ_{i-1} plus the set of all the messages an intruder would be able to add to its knowledge by eavesdropping on a run of the protocol during the whole time interval i (of course including those messages coming from all the other senders processes). The same considerations about the power of the intruder hold as in the previous section. Actually, the intruder has more powerful means to act since the beginning of each time interval.

Now we check if S_0^j and $R_0^{j,q}$, specified in Subsection 2.3, separately satisfy the properties of interest. Let $\mathbf{0'}$ be the process that simply let time pass, $\mathbf{0'} = tick.\mathbf{0'}$. Then, S_0^j enjoys $tGNDC^{\mathbf{0'}}_{\leq_{ttrace}}$ and $R_0^{j,q}$ enjoys $tGNDC^{\alpha_{tInt}(P^q)}_{\leq_{ttrace}}$, that is to say for all $X \in t\mathcal{E}_C^{\phi_0}$ we have $(S_0^j||X)\backslash C \leq_{ttrace} \mathbf{0'}$ and $(R_0^{j,q}||X)\backslash C \leq_{ttrace} \alpha_{tInt}(P^q)$. This may be proved by finding a suitable weak simulation relation between $(S_0^j||X_{\phi_0}) \setminus C$ and $\mathbf{0'}$ and between $(R_0^{j,q}||X_{\phi_0}) \setminus C$ and $tSpec_0$, respectively. The set C of channels over which an intruder is able to communicate is $C = \{c_i \mid i \geq 0\}$.

Lemma 1. *S_0^j and $R_0^{j,q}$ are t. d. stable w.r.t. $\{\phi_i\}$.*

Lemma 2. *$S_0^j \in tGNDC^{\mathbf{0'}}_{\leq_{ttrace}}$ and $R_0^{j,q} \in tGNDC^{\alpha_{tInt}(P^q)}_{\leq_{ttrace}}$.*

The proof of Lemma 2 is in the Appendix.

The following proposition follows by Lemmas 1 and 2 and by Proposition 5, where $r = 1, 2$, $P_1 = S_0^j$, $P_2 = R_0^{j,q}$.

Proposition 6. *$P^q \in tGNDC^{\alpha_{tInt}(P^q)}_{\leq_{ttrace}}$*[10].

[10] Note that $\mathbf{0'}||\alpha_{tInt}(P^q) \leq_{ttrace} \alpha_{tInt}(P^q)$.

The correctness of the multiple receivers version (considering all the receivers belonging to subgroup j), can be also proved using results of Lemmas 1 and 2 and Proposition 5, where index r is not fixed *a priori* and $P_1 = S_0^j$ and $P_r = R_0^{j,q}$ with $1 \leq q \leq n_j$.

Proposition 7. *System* P^j *(in Definition 4)* $\in tGNDC^{\alpha^j_{tInt}(P^j)}_{\leq ttrace}$.

We get into the issue of considering a multiple senders/receivers environment. Let us consider $\Gamma = \Pi_{1 \leq j \leq n} P^j$ and $\alpha_{tInt}(\Gamma) = \Pi_{1 \leq j \leq n} \alpha^j_{tInt}(P^j)$, where n is the cardinality of the senders processes.

Proposition 8. *System* $\Gamma \in tGNDC^{\alpha_{tInt}(\Gamma)}_{\leq ttrace}$.

The result follows by application of Propositions 5 and 7.

We note that, in order to have timed integrity on the messages m_i, μTESLA must ensure timed secrecy on the keys K_i. Indeed, we could also check explicitly timed secrecy on the keys with the same machinery.

3.4 An analysis of the N Root/Leaf pairwise keys protocol: secrecy

A secrecy analysis on the protocol presented in Section 2.4, with respect to an intruder that tries to discover m, is achieved by exploiting the principle on the persistent stability of the parallel composition of stable processes, introduced as part of Prop. []. For the sake of clarity, we report this result as a stand-alone lemma.

Lemma 3. *Given an intruder's initial knowledge* ϕ_X*, assume that* P_1 *and* P_2 *are stable processes w.r.t.* ϕ_X*; then* $P_1||P_2$ *is stable w.r.t.* ϕ_X.

We informally motivate the guidelines of the analysis, before showing its steps. The intruder is provided with an initial knowledge ϕ_X, that can be increased to the set ϕ'_X during the execution of the protocol by the messages the intruder process receives. Accordingly, the intruder's knowledge becomes at most $\mathcal{D}(\phi'_X)$.

Thus, to carry out an analysis on the secrecy of messages, one can act in the following way. We must analyze how the knowledge of the intruder is altered in the course of the protocol execution. If, by increasing its knowledge, message m happens to be in that knowledge, this does mean that the intruder has discovered m. In other words, there is a secrecy attack on the protocol.

Thus, let ϕ_X be the set $\{\{K\}_{K_{IL_1}}|\{K\}_{K_{IL_2}}|\ldots|\{K\}_{K_{IL_N}}\} \cup \{\{m\}_K\}$.

One can easily check that $I_1(k,m)$ is stable w.r.t. ϕ_X and each $L_1^n(K_{IL_n})$, with $1 \leq n \leq N$, is stable w.r.t. ϕ_X. Let C be the set $\{c_1, c_2\}$.

During its computation, the initiator process performs only two output actions, whose corresponding messages exactly correspond to ϕ_X. On the other hand, each leaf in the set of the receivers does not perform any output action, thus not contributing to augmenting the initial knowledge ϕ_X.

By applying Lemma 3, one can conclude that process $P(K,m) \doteq I_1(k,m)||L_1^n(K_{IL_n})$ is stable w.r.t. ϕ_X, meaning that the knowledge of the intruder does not significantly evolve during the computation of the protocol. In particular, this means that the protocol preserves the secrecy of message m (given of course the initial confidentiality of m and the correct choice and delivery of $\{K_{IL_n}\}$ and K).

4 Conclusions

Multicast and wireless security are a fertile field for computer science and engineering researchers and developers. In this paper an attention was focused on methodologies for certifying the correctness of some architectures for authenticating digital streams and giving them data confidentiality.

The modeling and the verification approaches have been presented through some case studies. In particular, the protocols' models have been given by exploiting a process-algebraic framework dealing with cryptographic and timed primitives. Also, the framework is rich enough to describe wireless communications (at the level of details useful for our goals). The verification has been performed using appropriate methods derived from usual process-algebras techniques, such as simulation checking. A key feature is the application of compositional analysis techniques that allowed us to check systems even with an unbounded number of participants.

The choice of the case studies involving the signature of digital streams has not been random. Indeed, the first is considered a pioneering protocol in the field. However, it suffers from the problem of packet loss, in the sense that, if a packet is missing, the authentication chain is broken and the integrity of the subsequent packets cannot be verified. Several protocols were born with the intent of fighting against this problem. In particular, we have chosen EMSS, in order to model also packet loss. We achieve it through a non-deterministic choice performed at the receiver's side. Finally, also timed issues in wireless environments have been considered. To this aim, a process algebra enriched with timed primitives has been used, able to model the passing of time.

An analysis has been also conducted in order to prove that the multicast data are not modified *en-route*, *i.e.*, in their traveling from one sender to the set of receivers. To analyze this sort of robustness against packet modification, also called integrity of packets, a compositional analysis has been applied. The methodology can work both in a timed and in an untimed setting and, for some protocols, it has the advantage of carrying out the analysis over an unbounded number of components.

In the timed case study, the fulfillment of the property of timed integrity is a consequence of the fulfillment of the property of timed secrecy over the keys that are going to be disclosed. We could also have checked explicitly timed secrecy over those keys, with the same proposed machinery. On the contrary, what has been proposed here is a case study dealing with secure group communication. Whereas the modeling of the protocol has been done within the same process-algebraic framework, another principle has been used for the analysis. The aim of the analysis was checking the fulfillment of the secrecy of data exchanged within the group's members. To this aim, the property of secrecy has been mapped into a property over the intruder's knowledge, by checking how it changes during the computation. A possible extension to this kind of analysis could be enlarging the scenario to protocols guaranteeing forward and backward secrecy in dynamic groups, see, *e.g.*, [28].

To sum up, the number of protocols, the different scenarios and the properties we were able to deal with suggest the feasibility of our verification approach. By starting from these results, we are also going to develop techniques to automatize the proofs as well as a more precise modeling of wireless communication. This could allow us to deal with other relevant properties such as denial of service, location-based security properties (as privacy location) and similar issues.

References

1. M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME. In *Proc. WITS'02*, 2002. Informal proceedings.
2. B. Bell and L. La Padula. Secure computer systems—unified exposition and multics interpretation. Technical Report Tech. Rep. ESD-TR-75-306, MITRE MTR-2997, 1976.
3. P. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *Proc. ESORICS'02*, volume LNCS 2502, pages 146–161. Springer, 2002.
4. R. Canetti, J.A. Garay, G. Itkis, D. Micciancio, M. Naor, and B.Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proc. of INFOCOM 1999*, pages 708–716. IEEE, 1999.
5. R. Focardi and R. Gorrieri. A taxonomy of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, 1995.
6. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proc. ICALP'00*, volume LNCS 1853, pages 354–372. Springer, 2000.
7. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non interference. In *ENTCS 32*, 2000.
8. R. Focardi, R. Gorrieri, and F. Martinelli. Classification of security properties—part II: Network security. In *Proc. FOSAD 2001/2002—Tutorial Lectures*, volume LNCS 2946, pages 139–185. Springer, 2004.
9. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proc. FM'99*, volume LNCS 1708, pages 794–813. Springer, 1999.
10. R. Gennaro and P. Rohatgi. How to sign digital streams. *Information and Computation*, 165(1):100–116, 2001.
11. J.A. Goguen and J. Meseguer. Security policy and security models. In *Proc. S&P'82*, pages 11–20. IEEE, 1982.
12. P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proc. NDSS'01*. The Internet Society, 2001.
13. R. Gorrieri and F. Martinelli. A simple framework for real-time cryptographic protocol analysis with compositional proof rules. *Sci. Comput. Program.*, 50(1-3):23–49, 2004.
14. R. Gorrieri, F. Martinelli, M. Petrocchi, and A. Vaccarelli. Compositional verification of integrity for digital stream signature protocols. In *Proc. ACSD'03*, pages 142–149. IEEE, 2003.
15. R. Gorrieri, F. Martinelli, M. Petrocchi, and A. Vaccarelli. Formal analysis of some timed security properties in wireless protocols. In *Proc. FMOODS'03*, volume LNCS 2884, pages 139–154. Springer, 2003.
16. M. Hennessy and T. Regan. A temporal process algebra. *Information and Computation*, 117:222–239, 1995.
17. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. TACAS'96*, volume LNCS 1055, pages 147–166. Springer, 1996.
18. F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.
19. F. Martinelli, M. Petrocchi, and A. Vaccarelli. Compositional verification of secure streamed data: a case study with EMSS. In *Proc. ICTCS'03*, LNCS 2841, pages 383–396. Springer, 2003.
20. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

21. J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In *Proc. S&P'02*, pages 227–240. IEEE, 2002.
22. A. Perrig, R. Canetti, D. X. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proc. NDSS'01*. The Internet Society, 2001.
23. A. Perrig, R. Canetti, D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. S&P'00*, pages 56–73. IEEE, 2000.
24. A. Perrig, R. Szewczyk, D. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless Networks Journal*, 8:521–534, 2002.
25. J. Postel. The User Datagram Protocol - RFC 768, 1980.
26. M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Proc. of FORTE 2003*, volume LNCS 2767, pages 240–256. Springer, 2003.
27. D. Wallner, E. Harder, and R. Agee. RFC 2627: Key management for multicast: issues and architectures, 1999.
28. C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.

APPENDIX

A Crypto-CCS and *tCryptoSPA*

This appendix presents a concise description of the Crypto-CCS and the *tCryptoSPA* syntax and semantics. Some constructs of the languages are here omitted, since they are not of direct interest for the investigated topics. For a complete description, the interested reader is invited to see [18, 13], respectively.

A.1 Crypto-CCS

The model of the language consists of sequential agents able to communicate by exchanging messages.

The data handling part of the language consists of messages and inference systems. Messages are the data manipulated by agents, they form a set *Msgs* of terms possibly containing variables. The set *Msgs* is defined by the grammar:

$$m ::= x \mid b \mid F^1(m_1, \ldots, m_{k_1}) \mid \ldots \mid F^l(m_1, \ldots, m_{k_l})$$

where F^i (for $1 \leq i \leq l$) are the constructors for messages, $x \in V$ is a countable set of variables, $b \in B$ is a collection of basic messages and k_i, for $1 \leq i \leq l$, gives the number of arguments of the constructor F^i. Messages without variables are *closed* messages.

Inference systems model the possible operations on messages. They consist of a set of rules r, *e.g.*, :

$$r = \frac{m_1 \quad \ldots \quad m_n}{m_0}$$

where $\{m_1, \ldots, m_n\}$ is a set of premises (possibly empty) and m_0 is the conclusion. An instance of the application of rule r to closed messages m_i is denoted as $m_1 \quad \ldots \quad m_n \vdash_r m_0$. Given an inference system, a *deduction function* $\mathcal{D}$ is defined such that, if ϕ is a finite set of closed messages, then $\mathcal{D}(\phi)$ is the set of closed messages that can be deduced starting from ϕ by applying instances of the rules in the system. The syntax and semantics of Crypto-CCS are parametric with respect to a given inference system.

The control part of the language consists of compound systems, *i.e.*, sequential agents running in parallel. The language syntax is as follows:

$$\begin{array}{ll} \textsc{Compound systems:} & S ::= (S_1 || S_2) \mid S \setminus C \mid A_\phi \\ \textsc{Sequential agents:} & A ::= \mathbf{0} \mid p.A \mid A_1 + A_2 \mid [m_1 \ldots m_n \vdash_r x]A_1; A_2 \\ & \qquad \mid [m = m']A_1; A_2 \mid E(m_1, \ldots, m_n) \\ \textsc{Prefix constructs:} & p ::= c!m \mid c?x \end{array}$$

where $m, m', m_1, \ldots, m_n$ are *closed* messages or variables, x is a variable, $c \in Ch$ (a finite set of channels) ϕ is a finite set of *closed* messages, C is a subset of Ch.

$\mathbf{0}$ is the process that does nothing.

$p.A$ is the process that can perform an action according to the particular prefix construct p and then behaves as A. In particular,

- $c!m$ denotes a message m sent on channel c;
- $c?x$ denotes the receiving of a message m on channel c. The received message replaces the variable x.

$$(!)\frac{}{(c!m.A)_\phi \xrightarrow{c!m} (A)_\phi} \qquad (||_1)\frac{S \xrightarrow{a} S'}{S||S_1 \xrightarrow{a} S'||S_1}$$

$$(?)\frac{m \in Msgs}{(c?x.A)_\phi \xrightarrow{c?m} (A[m/x])_{\phi\cup\{m\}}} \qquad (||_2)\frac{S \xrightarrow{c!m} S' \quad S_1 \xrightarrow{c?m} S_1'}{S||S_1 \xrightarrow{\tau} S'||S_1'}$$

$$(\mathcal{D})\frac{m_1 \ldots m_n \vdash_r m \quad (A[m/x])_{\phi\cup\{m\}} \xrightarrow{a} (A')_{\phi'}}{([m_1 \ldots m_n \vdash_r x]A; A_1)_\phi \xrightarrow{a} (A')_{\phi'}} \qquad (\backslash_1)\frac{S \xrightarrow{c!m} S' \quad c \notin L}{S \setminus L \xrightarrow{c!m} S' \setminus L}$$

$$(+_2)\frac{S \xrightarrow{a} S'}{S + S_1 \xrightarrow{a} S'}$$

$$(\mathcal{D}_1)\frac{\nexists m \text{ s.t. } m_1 \ldots m_n \vdash_r m \quad (A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{([m_1 \ldots m_n \vdash_r x]A; A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(=)\frac{m = m' \quad (A)_\phi \xrightarrow{a} (A')_{\phi'}}{([m = m']A; A_1)_\phi \xrightarrow{a} (A')_{\phi'}}$$

$$(=_1)\frac{m \neq m' \quad (A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}{([m = m']A; A_1)_\phi \xrightarrow{a} (A_1')_{\phi'}}$$

$$(Const)\frac{E(x_1, \ldots, x_n) =_{def} A \quad A[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} A_1}{E(m_1, \ldots, m_n) \xrightarrow{a} A_1}$$

Fig. 5. Operational semantics of Crypto-CCS.

$A_1 + A_2$ represents the non deterministic choice between A and A_1.

$[m_1 \ldots m_n \vdash_r x]A_1; A_2$ is the inference construct. If, by applying an instance of rule r, with premises $m_1 \ldots m_n$, a message m can be inferred, then the process behaves as A_1 (where m replaces x), otherwise it behaves as A_2.

$[m = m']A_1; A_2$ is the match construct, to check message equality. If $m = m'$ then the system behaves as A_1, otherwise it behaves as A_2.

A compound system $S_1||S_2$ denotes the parallel execution of S_1 and S_2. $S_1||S_2$ performs an action p if one of its sub-components performs p. A synchronization, or internal action, denoted by τ, may take place whenever S_1 and S_2 are able to perform two complementary actions, *i.e.*, send-receive actions on the same channel.

A compound system $S \setminus C$ allows only visible actions whose channels are not in C. (Internal action τ being the invisible action).

The term A_ϕ is a single sequential agent whose knowledge, *i.e.*, the set of messages which occur in its term, is described by ϕ. The knowledge of an agent increases either when it receives messages (see rule (?) in Fig. 5) or it infers new messages from the messages it knows (see rule $\mathcal{D}$ in Fig. 5). For every sequential agent A_ϕ, it is required that all the closed messages that appear in A_ϕ belong to its knowledge ϕ.

The activities of the agents are described by the actions that they can perform. The set *Act* of actions which may be performed by a compound system ranges over by

a and it is defined as: $Act = \{c?m, c!m, \tau \mid c \in C, m \in Msgs, m \text{ closed}\}$. $\mathcal{P}$ is the set of all the Crypto-CCS *closed* terms (*i.e.*, with no free variables). *sort(P)* is the set of all the channels that syntactically occur in the term P.

The operational semantics of a Crypto-CCS term is described by means of the *labeled transition system* (*lts*, for short) $\langle \mathcal{P}, Act, \{\stackrel{a}{\longrightarrow}\}_{a \in Act}\rangle$, where $\{\stackrel{a}{\longrightarrow}\}_{a \in Act}$ is the least relation between Crypto-CCS processes induced by the axioms and inference rules of Fig. 5 (in that figure the symmetric rules for $||_1, ||_2, \backslash_1, +_2$ are omitted).

The expression $S \stackrel{a}{\longrightarrow} S'$ means that the system can move from the state S to the state S' through the action a. The expression $S \Longrightarrow S'$ denotes that S and S' belong to the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$; let $\gamma = a_1 \dots a_n \in (Act \backslash \{\tau\})^*$ be a sequence of actions. Then, $S \stackrel{\gamma}{\Longrightarrow} S'$ if $S \Longrightarrow \stackrel{a_1}{\longrightarrow} \Longrightarrow \dots \Longrightarrow \stackrel{a_n}{\longrightarrow} \Longrightarrow S'$.

As behavioral relations among Crypto-CCS terms, we are interested in trace inclusion (equivalence) and (weak) simulation.

Definition 6. *We say that the traces of P are included in the traces of Q ($P \leq_{trace} Q$) whenever, if $P \stackrel{\gamma}{\Longrightarrow} P_1$ then $Q \stackrel{\gamma}{\Longrightarrow} Q_1$. We write that $P =_{trace} Q$ iff $P \leq_{trace} Q$ and $Q \leq_{trace} P$.*

Definition 7. *We say that a relation $\mathcal{R}$ among processes is a weak simulation, if for every $(P, Q) \in \mathcal{R}$ we have:*

- *If $P \stackrel{a}{\longrightarrow} P', a \neq \tau$, then there exists Q' s.t. $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$.*
- *If $P \stackrel{\tau}{\longrightarrow} P'$ then there exists Q' s.t. $Q \Longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$.*

The union of all weak simulations is a weak simulation and it is denoted by $\prec$. As usual, it holds that if $P \prec Q$ then $P \leq_{trace} Q$.

A.2 *tCryptoSPA*

The real-time extension of the *Cryptographic Security Process Algebra* (for short, $CryptoSPA$) of [9, 6] has been proposed in [13]. The new language, $timedCryptoSPA$ ($tCryptoSPA$ for short), is adopted for describing cryptographic protocols where information about the concrete timing of events is necessary. We remind the reader of the syntax, the operational semantics of the language and some auxiliary notions. The description is not exhaustive, since some constructs are not of direct interest for the investigated topics. Furthermore, some terms of the language are the same as in the Crypto-CCS language. Finally, the interested reader is referred to [13] for a more complete discussion of $tCryptoSPA$.

The set $\mathcal{L}$ of $tCryptoSPA$ processes is defined as:

$$P ::= \mathbf{0} \mid c(x).P \mid \overline{c}m.P \mid \tau.P \mid tick.P \mid P_1 + P_2 \mid P_1||P_2 \mid P\backslash L \mid$$

$$A(m_1, \dots, m_n) \mid [\langle m_1, \dots, m_r\rangle \vdash_{rule} x]P_1; P_2$$

We omit to describe terms whose meaning has been already explained in the previous part of the appendix, subsection A.1. To this aim, note that the $tCryptoSPA$ sequential construct $\overline{c}e.P$ is syntactically and semantically equivalent to the Crypto-CCS sequential construct $c!m.P$. Thus, $\overline{c}m.P$ is the process that can send m on channel c, then behaving like P.

$m, m_1, \dots, m_r, m_n$ are messages or variables and L is a set of channels. Both the operators $c(x).P$ and $[\langle m_1 \dots m_r\rangle \vdash_{rule} x]P_1; P_2$ bind the variable x in P and P_1, respectively.

Let $Def : Const \longrightarrow \mathcal{L}$ be a set of defining equations of the form $A(x_1, \ldots, x_n) \doteq P$, where P may contain no free variables except $x_1, \ldots, x_n$, which must be distinct. Constants permit us to define recursive processes. A term P is *closed* with respect to Def if all the constants occurring in P are defined in Def (and, recursively, for their defining terms). A term P is *guarded* w.r.t. Def if all the constants occurring in P (and, recursively, for their defining terms) occur in a prefix context [20].

The set Act of actions which may be performed by a system is defined as: $Act = \{c(m), \overline{c}m, \tau, tick, \mid c \in \mathcal{I}, \overline{c} \in \mathcal{O}, m \in \mathcal{M}, m \text{ closed}\}$. τ is the internal, invisible action. $tick$ is the special action used to model time elapsing. We let l range over $Act\backslash\{tick\}$. We call $\mathcal{L}$ the set of all the *tCryptoSPA closed* terms (*i.e.*, with no free variables) that are closed and guarded w.r.t. Def. We define *sort(P)* to be the set of all the channels syntactically occurring in the term P.

$\tau.P$ is the process that executes the internal action τ and then behaves like P;

$tick.P$ is a process willing to let one time unit pass and then behaving as P;

$P_1 + P_2$ (*choice*) represents the nondeterministic choice between the two processes P_1 and P_2; with respect to $tick$ actions, time passes when both P_1 and P_2 are able to perform a $tick$ action – and in such a case by performing $tick$ a configuration where both the derivatives of the summands can still be chosen is reached. When only one of the two processes can perform $tick$, say P_1, it could be either that P_1 performs $tick$ – and in such a case P_2 is discarded – or P_2 performs its normal activity – and in such a case P_1 is discarded; moreover, τ prefixed summands have priority over $tick$ prefixed summands;

$P_1||P_2$ (*parallel*) is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize on complementary actions to make a communication, represented by a τ. Both components must agree on performing a $tick$ action, and this can be done even if a communication is possible.

$P\backslash L$ allows only visible actions whose channels are not in L;

$A(m_1, \ldots, m_n)$ behaves like the respective defining term P where all the variables $x_1, \ldots, x_n$ are replaced by the messages $m_1, \ldots, m_n$.

The time model adopted in the language is known as the *fictitious clock* approach of, *e.g.*, [16]. A global clock is supposed to be updated whenever all the processes agree on this, by globally synchronizing on the special action $tick$, representing the passing of a time unit. All the other actions are assumed to take no time.

itted).

The expression $P \stackrel{a}{\Rightarrow} P'$ is a shorthand for $P(\stackrel{\tau}{\longrightarrow})^* P_1 \stackrel{a}{\longrightarrow} P_2(\stackrel{\tau}{\longrightarrow})^* P', a \neq \tau$, where $(\stackrel{\tau}{\longrightarrow})^*$ denotes a (possibly empty) sequence of transitions labeled τ. The expression $P \Rightarrow P'$ is a shorthand for $P(\stackrel{\tau}{\longrightarrow})^* P'$. Let $\gamma = a_1, \ldots, a_n \in (Act\backslash\{\tau\})^*$ be a sequence of actions; then $P \stackrel{\gamma}{\Rightarrow} P'$ iff there exist $P_1, \ldots, P_{n-1} \in \mathcal{P}$ such that $P \stackrel{a_1}{\Rightarrow} P_1 \stackrel{a_2}{\Rightarrow}, \ldots, P_{n-1} \stackrel{a_n}{\Rightarrow} P'$. Let $\mathbf{0'} \doteq tick.\mathbf{0'}$.

For timed behavioural relations among $tCryptoSPA$ processes, we will be mainly interested in *timed trace* inclusions.

Definition 8. *For any $P \in \mathcal{L}$ the set $T(P)$ of timed traces associated with P is defined as follows $T(P) = \{\gamma \in (Act\backslash\{\tau\})^* \mid \exists P'.P \stackrel{\gamma}{\Rightarrow} P'\}$. The timed trace pre-order, denoted by $\leq_{ttrace}$, is defined as follows: $P \leq_{ttrace} Q$* iff *$T(P) \subseteq T(Q)$. P and Q are* timed trace equivalent*, denoted by $P =_{ttrace} Q$, if $T(P) = T(Q)$.*

We define the concept of weak simulation as usual.

Definition 9. *We say that a relation R among processes is a weak simulation, if for every $(P, Q) \in \mathcal{R}$ we have:*

$$(tick)\frac{}{tick.P \xrightarrow{tick} P} \qquad (||_1)\frac{P_1 \xrightarrow{l} P_1'}{P_1||P_2 \xrightarrow{l} P_1'||P_2} \qquad (||_2)\frac{P_1 \xrightarrow{c(x)} P_1' \quad P_2 \xrightarrow{\overline{c}m} P_2'}{P_1||P_2 \xrightarrow{\tau} P_1'||P_2'}$$

$$(||_3)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1||P_2 \xrightarrow{tick} P_1'||P_2'} \qquad (+_1)\frac{P_1 \xrightarrow{l} P_1'}{P_1 + P_2 \xrightarrow{l} P_1'}$$

$$(+_2)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1 + P_2 \xrightarrow{tick} P_1' + P_2'} \qquad (+_3)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \not\xrightarrow{tick} \quad P_2 \not\xrightarrow{\tau}}{P_1 + P_2 \xrightarrow{tick} P_1'}$$

Fig. 6. Semantics of $tCryptoSPA$ involving action $tick$.

- *If* $P \xrightarrow{a} P', a \neq \tau$, *then there exists* Q' *s.t.* $Q \stackrel{a}{\Longrightarrow} Q'$ *and* $(P', Q') \in \mathcal{R}$.
- *If* $P \xrightarrow{\tau} P'$ *then there exists* Q' *s.t.* $Q \Longrightarrow Q'$ *and* $(P', Q') \in \mathcal{R}$.

Let $\prec$ the union of all weak simulations among processes. Then, we have $\prec \subseteq \leq_{ttrace}$.

B GNDC and tGNDC

In this appendix, we present the general schema *Generalized Non Deducibility on Compositions* (GNDC), for the definition of security properties given in [6, 9, 8], and its timed extension tGNDC given in [13].

In the literature, several efforts have been made to prevent the unauthorized information flow in multilevel computer systems [2], *i.e.* systems where processes and objects are bound to a specific security level. An example from military jargon is the fact that documents are generally hierarchized from unclassified to top secret. The seminal idea of *non interference* proposed in [11] aims at assuring that information can only flow from low levels to higher ones. The first taxonomy of non-interference-like properties has been uniformly defined and compared in [5] in the context of a CCS-like process algebra. In particular, processes in the algebra were divided into high and low processes, according to the level of actions that they can perform. To detect whether an incorrect information flow (*i.e.* from high to low) has occurred, a particular non-interference-like property has been defined, the so-called *Non Deducibility on Compositions* (NDC). NDC essentially says that a process is secure with respect to wrong information flows if its low behaviour in isolation appears to be the same as its low behaviour when interacting with any high-level process. NDC can be reformulated from the world of multilevel systems to the one of network security. See [9, 7], where the low-level process becomes a specification of a cryptographic communication protocol and the behaviour of the protocol running in isolation is compared with that of the protocol running in parallel with any possible adversary.

As a further step, a *Generalized* NDC (GNDC) has been formulated in [9], in order to encompass in a uniform way many security properties. The main idea of GNDC is the following: a system P satisfies property $GNDC^{\alpha}_{\lhd}$ if the behavior of P, despite the presence of a hostile environment X that can interact with P only through a fixed set of channels C, *appears* to be same (w.r.t. a behavioral relation $\lhd$ of observational equivalence) to the behavior of a modified version $\alpha(P)$ of P that represents the *expected* (correct) behavior of P.

The analysis of cryptographic protocols involves specifying a set of messages known by the adversary at the beginning of the computation. This *static* (initial) knowledge of the hostile environment must be bound to a specific set of messages. This limitation is needed to avoid a too strong hostile environment that would be able to corrupt any secret (as it would know all cryptographic keys, etc.). Given an adversary X, we call $ID(X)$ the set of closed messages that syntactically appear in X. This set, intuitively, contains all the messages that are initially known by X. Let ϕ_X be a set of messages representing the static, initial knowledge that we would like to give to X. We want $ID(X)$ to be consistent with ϕ_X. This can be obtained by requiring that all the messages in $ID(X)$ are *deducible* from ϕ_X by means of the *deduction function* $\mathcal{D}$.

The set $\mathcal{E}_C^{\phi_X}$ of processes that can communicate on a subset of public channels C and have an initial knowledge bound by ϕ_X can be therefore defined as follows:

$$\mathcal{E}_C^{\phi_X} = \{X \in \mathcal{P} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \ \mathcal{D}(\phi_X)\}$$

We consider as hostile processes only the ones belonging to $\mathcal{E}_C^{\phi_X}$.
We define the property $GNDC_{\lhd}^{\alpha}$ as follows:

Definition 10. *A process P is $GNDC_{\lhd}^{\alpha} \iff \forall X \in \mathcal{E}_C^{\phi_X} : (P||X) \setminus C \lhd \alpha(P)$ where $\lhd : \mathcal{P} \to \mathcal{P}$ is a behavioral relation between processes and $\alpha : \mathcal{P} \to \mathcal{P}$ is a function between processes.*

For the sake of completeness, it is worth noticing that a slightly extended GNDC schema has been recently defined in [8], incorporating the fact that the set of bad behaviours of P may depend on P itself and on the property under scrutiny.

For the analysis of safety properties it is enough to consider the trace inclusion relation $\leq_{trace}$ as behavioral relation among the terms of the algebra. When the $\leq_{trace}$ relation is considered, there exists a sufficient criterion for the static characterization, *i.e.*, not involving the universal predicate $\forall$, of $GNDC_{\lhd}^{\alpha}$ properties. In the following, we give hints to the definition of GNDC without the need of the universal predicate, since some notions will be useful in the rest of the paper. For further details about this static characterization, the interested reader can see [6, 9], where the following statements were first declared and proved.

Informally, the so called most powerful intruder in the trace setting ($(Top_{trace}^C)_\phi$, hereafter, for short Top_ϕ^C) is that intruder whose knowledge is ϕ, that can communicate only over channels in C, that can receive every message passing over these channels (increasing in such a way its knowledge) and, finally, that can send over these channels every message that it can deduce starting from ϕ.

More formally, Top_ϕ^C is defined as follows in [9]:

Definition 11.

$$Top_\phi^C = \sum_{c \in C} c(x).Top_{\phi \cup \{x\}}^C + \sum_{c \in C, m \in \mathcal{D}(\phi)} \overline{c}m.Top_\phi^C$$

It has been proved ([6, 9]) that the general way in which Top_C^ϕ is specified implies that its behaviour includes that of any X belonging to the set $\mathcal{E}_C^{\phi_X}$ of admissible hostile processes.

Corollary 1. *For every function $\alpha : \mathcal{P} \to \mathcal{P}$, a process P is $GNDC_{\leq_{trace}}^{\alpha} \iff (P||Top_\phi^C) \setminus C \leq_{trace} \alpha(P)$.*

The corollary implies that, for the analysis of safety properties in the trace setting, to check if a specification enjoys GNDC w.r.t. all the admissible hostile environments, it is sufficient to check if the same specification enjoys GNDC with respect to the most powerful intruder Top_C^{ϕ}.

By varying the parameter α, the GNDC schema can be used to define and verify many security properties—among which secrecy, integrity, and entity authentication [5–7, 9, 14, 19]. As an example, we remind here how the secrecy and the entity authentication properties have been formalized in [6] (relation $\lhd$ for specifying these properties is trace inclusion $\leq_{trace}$).

The requirements for a secrecy property to be satisfied are quite intuitive: a certain message M, declared to be secret, should not be learnt by unauthorized users. Thus, let us consider the event *learnt(M)*, signaling that M has been learnt by the hostile environment. Then, $\alpha_S(P(m))$ "is the set of processes where the event *learnt(M)* can never occur". For more details, the interested reader can see [7].

On the other hand, entity authentication "should allow the verification of an entity's claimed identity, by another entity" [6]. To formalize this action, the followed approach is the one proposed in [17] and based on a so called *correspondence* between actions. Let us consider two users A and B, participating through a protocol. To assure the property, one would like that, whenever A concludes the protocol apparently with B, B has indeed executed the protocol. This can be tested with the introduction of two events, *commit(A,B)* and *run(B,A)*, representing the fact that A has indeed terminated the protocol apparently with B, (action *commit*), and B has indeed started communicating with A, (action *run*). To fulfill entity authentication means to require that event *commit(A,B)* is always preceded by event *run(B,A)*. In the GNDC definition, $\alpha_{EA}(P)$ is the process where *commit(A,B)* is always preceded by *run(B,A)*.

Along with GNDC, a general schema for the definition of timed security properties, called *timed Generalized Non Deducibility on Compositions* (*tGNDC* for short) has been proposed in [13].

Property *tGNDC* rephrases the analogue GNDC, but in a timed setting. A system S is $tGNDC_{\lhd}^{\alpha} \iff$ for every enemy X the composition of the system with X satisfies the timed specification $\alpha(S)$. Basically, *tGNDC* guarantees that the timed property α is satisfied, with respect to the $\lhd$ timed behavioral relation, even when the system is composed with any possible adversary X.

We give here the set of admissible hostile environments for our timed setting. For a certain enemy X, we call $ID(X)$ the set of closed messages that syntactically appears in X, all the messages initially known by X. Let ϕ_0 be the initial knowledge we would like to give to the enemy at the beginning of the computation. We require that all the messages in $ID(X)$ are deducible from ϕ_0. We consider as hostile processes only the ones belonging to the set $t\mathcal{E}_C^{\phi_0}$ [11]. They can communicate on a subset of public channels C and have an initial knowledge bound by ϕ_0:

$$t\mathcal{E}_C^{\phi_0} = \{X \in \mathcal{L} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_0)\}$$

The property $tGNDC_{\lhd}^{\alpha}$ is defined as follows:

[11] Actually, there is another constraint that imposes that the enemy must eventually let time pass. This is however not useful for safety properties we are going to study in this paper and so it has been omitted for the sake of simplicity.

Definition 12. *S is $tGNDC^{\alpha}_{\lhd}$ $\iff$ $\forall X \in t\mathcal{E}^{\phi_0}_C : (S||X)\backslash C \lhd \alpha(S)$ where $\lhd : \mathcal{L} \to \mathcal{L}$ is a timed behavioral relation between processes and $\alpha : \mathcal{L} \to \mathcal{L}$ is a function between processes defining the property specification for S as the process $\alpha(S)$.*

As for the case of GNDC, it has been shown that ([13]), for the analysis of safety properties in the timed-trace setting, it is possible to prove the existence of a most general intruder $(tTop^C_{ttrace})_\phi$, acting as its companion in the non timed setting. Moreover, $(tTop^C_{ttrace})_\phi$ can let time pass, by performing *tick* actions. Again, the timed traces of $(tTop^C_{ttrace})_\phi$ include those of any X belonging to the set $t\mathcal{E}^{\phi_0}_C$, [13].

Thus, the following corollary holds:

Corollary 2. *For every function $\alpha : \mathcal{L} \to \mathcal{L}$, a process S is $tGNDC^{\alpha}_{\leq_{ttrace}}$ $\iff$ $(S||(tTop^C_{ttrace}) \setminus C \leq_{ttrace} \alpha(S)$.*

C Proofs

Lemma A.5. $S^j_0 \in tGNDC^{\mathbf{0}'}_{\leq_{ttrace}}$ and $R^{j,q}_0 \in tGNDC^{\alpha_{tInt}(P^q)}_{\leq_{ttrace}}$

This may be proved by finding a suitable weak simulation relation between $(S^j_0||X_{\phi_0})\backslash C$ and $\mathbf{0}'$ and between $(R^{j,q}_0||X_{\phi_0}) \setminus C$ and $tSpec_0$, respectively. The set C of channels over which an intruder is able to communicate is $C = \{c_i \mid i \geq 0\}$.

The weak simulation relation for the sender specifications is the following:

$$\begin{aligned}\mathcal{R}_{\mathcal{S}} = &(((S^j_i(...)||X_{\phi_i})\backslash C, \mathbf{0}') \mid \forall i, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup(((B^j_i(...)||X_{\phi_i})\backslash C, \mathbf{0}') \mid \forall i, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup(((\overline{c_i}K^j_{i-1}.B^j_i(\ldots)||X_{\phi_i})\backslash C, \mathbf{0}') \mid i > 1, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\end{aligned}$$

The weak simulation relation we consider for dealing with the receiver specifications is the following (superscript q is omitted for simplicity):

$$\begin{aligned}\mathcal{R} = &(((R^j_0(null)||X_{\phi_0})\backslash C, tSpec_0) \mid X_{\phi_0} \in t\mathcal{E}^{\phi_0}_C)\\ &\cup((tick.(R^j_1(K^j_0)||X_{\phi_0})\backslash C, tSpec_0) \mid X_{\phi_0} \in t\mathcal{E}^{\phi_0}_C)\\ &\cup(((R^j_1(K^j_0)||X_{\phi_1})\backslash C, tSpec_1) \mid X_{\phi_1} \in t\mathcal{E}^{\phi_1}_C)\\ &\cup(((R^j_i(null, K^j_0)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup((tick.(R^j_i(p_{i-1}, K^j_0)||X_{\phi_{i-1}})\backslash C, tSpec_{i-1}) \mid i \geq 2, X_{\phi_{i-1}} \in t\mathcal{E}^{\phi_{i-1}}_C)\\ &\cup(((R^j_i(p_{i-1}, K^j_0)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup(((R^{j'}_i(p_i*, p_{i-1}, K^j_0)||X_{\phi_i})\backslash C, tSpec_i) \mid i \geq 2, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup(((R^j_i(x_{i-1}, K^j_0)||X_{\phi_i})\backslash C, tSpec_i) \mid fst(x_{i-1}) \neq m^j_{i-1}, i \geq 2, X_{\phi_i} \in t\mathcal{E}^{\phi_i}_C)\\ &\cup((tick.(R^j_i(x_{i-1}, K^j_0)||X_{\phi_{i-1}})\backslash C, tSpec_{i-1}) \mid fst(x_{i-1}) \neq m^j_{i-1}, i \geq 2,\\ &X_{\phi_{i-1}} \in t\mathcal{E}^{\phi_{i-1}}_C)\\ &\cup((tick.(R^j_i(p_{i-1}*, K^j_0)||X_{\phi_{i-1}})\backslash C, tick.tSpec_i) \mid i \geq 2, X_{\phi_{i-1}} \in t\mathcal{E}^{\phi_{i-1}}_C)\end{aligned}$$

where $p_1, p_{i-1}, p_i*, p_{i-1}*$ and x_{i-1} are not empty fields. $p_i*, p_{i-1}*$ are shortcuts to denote either authentic packets sent by the sender or others. We omitted to explicitly put in $\mathcal{R}_{\mathcal{S}}$ and $\mathcal{R}$ the pairs in which the first process performs deduction constructs.

Proof. Throughout the proof, we omit to consider the cases in which the sender and the receiver by themselves perform internal actions.

- $S^j_0 \in tGNDC^{\mathbf{0}'}_{\leq_{ttrace}}$. Let us consider relation $\mathcal{R}_{\mathcal{S}}$. $\mathcal{R}_{\mathcal{S}}$ is a weak simulation:

- $((S_i^j||X_{\phi_i})\backslash C, \mathbf{0'})$. S_i^j may
  - ∗ either perform a *tick* action: in this case the whole system on the left performs *tick* and $(S_i^j||X_{\phi_i})\backslash C \xrightarrow{tick} (S_{i+1}^j||X_{\phi_{i+1}})\backslash C$. $\mathbf{0'}$ is able to simulate it and $((S_{i+1}^j||X_{\phi_{i+1}})\backslash C, \mathbf{0'}) \in \mathcal{R}_{\mathcal{S}}$.
  - ∗ or go to intermediate state B_i^j. $\mathbf{0'}$ is able to simulate it and $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0'}) \in \mathcal{R}_{\mathcal{S}}$.
- $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0'})$. B_i^j may perform a sending action, whereas X_{ϕ_i} synchronizes on that action: the whole system performs τ. It may happen:
  - ∗ $(B_i^j||X_{\phi_i})\backslash C \xrightarrow{\tau} (B_i^j||X_{\phi_i})\backslash C, i = 0, 1$. $\mathbf{0'}$ is able to simulate it and $((B_i^j||X_{\phi_i})\backslash C, \mathbf{0'}) \in \mathcal{R}_{\mathcal{S}}$.
  - ∗ $(B_i^j||X_{\phi_i})\backslash C \xrightarrow{\tau} (\overline{c_i}K_{i-1}^j.B_i^j||X_{\phi_i})\backslash C, i \geq 1$. $\mathbf{0'}$ is able to simulate it and $((\overline{c_i}K_{i-1}^j.B_i^j||X_{\phi_i})\backslash C, \mathbf{0'}) \in \mathcal{R}_{\mathcal{S}}$.
- $((\overline{c_i}K_{i-1}^j.B_i(\ldots)||X_{\phi_i})\backslash C, \mathbf{0'})$. The process on the left may perform a τ action, i.e. $(\overline{c_i}K_{i-1}^j.B_i^j(\ldots)||X_{\phi_i})\backslash C \xrightarrow{\tau} (B_i^j(\ldots)||X_{\phi_i})\backslash C$. Similar to the previous item.

– $R_0^{j,q} \in tGNDC_{\leq ttrace}^{\alpha_{tInt}(P^q)}$. Let us consider relation $\mathcal{R}$. $\mathcal{R}$ is a weak simulation:

- $((R_0^j(null)||X_{\phi_0})\backslash C, tSpec_0)$. Suppose $R_0^j(null)$ performs a receiving action and X_{ϕ_0} the corresponding sending action. X_{ϕ_0} could have sent any message $\in \mathcal{D}(\phi_0)$ whereas the only message $R_0^j(null)$ will accept will be the MAC computed with key K_{SM}^j. In this case $(R_0^j(null)||X_{\phi_0})\backslash C \xrightarrow{\tau} tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C$ and $tSpec_0$ is able to simulate τ and $(tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C, tSpec_0) \in \mathcal{R}$. When the received message contains a MAC not computed with K_{SM}^j the system maintains the same configuration and $tSpec_0$ is able to simulate it.
- $(tick.(R_1^j(K_0^j)||X_{\phi_0})\backslash C, tSpec_0)$. The first process may only perform *tick* by reaching the configuration $(R_1^j(K_0^j)||X_{\phi_1})\backslash C$. Note that also $tSpec_0 \xrightarrow{tick} tSpec_1$ and $((R_1^j(K_0^j)||X_{\phi_1})\backslash C, tSpec_1) \in \mathcal{R}$.
- $((R_1^j(K_0^j)||X_{\phi_1})\backslash C, tSpec_1)$.
  - ∗ The first process may perform *tick* and go to $(R_2^j(null, K_0^j)||X_{\phi_2})\backslash C$. Note that also $tSpec_1 \xrightarrow{tick} tSpec_2$ and $((R_2^j(null, K_0^j)||X_{\phi_2})\backslash C, tSpec_2) \in \mathcal{R}$.
  - ∗ If $R_1^j(K_0^j)$ performs a receiving action and X_{ϕ_1} the corresponding sending action (by sending messages $\in \mathcal{D}(\phi_1)$), then $((R_1^j(K_0^j)||X_{\phi_1})\backslash C \xrightarrow{\tau} (tick.R_2^j(p_1*, K_0^j)||X_{\phi_1})$, where p_1* could be either the authentic packet send by the sender p_1 or another one x_1. Note that $((tick.R_2^j(p_1*, K_0^j)||X_{\phi_1}), tSpec_1) \in \mathcal{R}$.
- $(tick.(R_2^j(p_1, K_0^j)||X_{\phi_1})\backslash C, tSpec_1)$. The first process may only perform a tick action reaching the configuration $(R_2^j(p_1, K_0^j)||X_{\phi_2})\backslash C$. Note that also $tSpec_1 \xrightarrow{tick} tSpec_2$ and $((R_2^j(p_1, K_0^j)||X_{\phi_2})\backslash C, tSpec_2) \in \mathcal{R}$.
- $((R_i^j(p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$.
  - ∗ The first process may perform *tick* by reaching $\{(R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C$. Note that also $tSpec_i \xrightarrow{tick} tSpec_{i+1}$ and $((R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C, tSpec_{i+1}) \in \mathcal{R}$.
  - ∗ If $R_i^j(p_{i-1}, K_0^j)$ performs a receiving action then $((R_i^j(p_{i-1}, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\tau} (R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C$. Note that $((R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i) \in \mathcal{R}$.

- $((R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$. If $R_i^{j'}$ outputs a message over channel app, it must be $z = y_{mac}$, $p_{i-1} = snd(y_{mac})$, $x_{K_{i-1}} = K_{i-1}^j$ and $fst(p_{i-1})$ must be replaced with m_{i-1}^j. $R_i^{j'}(p_i*, p_{i-1}, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\overline{app}m_{i-1}^j} tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})\backslash C$ and $tSpec_i \xrightarrow{\overline{app}m_{i-1}^j} tick.tSpec_{i+1}$. Both the derivatives $\in \mathcal{R}$.
- $((tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})\backslash C, tick.tSpec_{i+1})$. Both the processes may perform $tick$ and the derivatives $\in \mathcal{R}$.
- $((R_i^j(null, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$.
  - If $R_i^j(null, K_0^j)$ performs a receiving action and X_{ϕ_i} the corresponding sending action, then $((R_i^j(null, K_0^j)||X_{\phi_i})\backslash C \xrightarrow{\tau}$ $(tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i})$. Note that $((tick.R_{i+1}^j(p_i*, K_0^j)||X_{\phi_i}), tSpec_i) \in \mathcal{R}$.
  - If the first process performs $tick$, it reaches the configuration $(R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C$. Note that also $tSpec_i \xrightarrow{tick} tSpec_{i+1}$ and $((R_{i+1}^j(null, K_0^j)||X_{\phi_{i+1}})\backslash C, tSpec_{i+1}) \in \mathcal{R}$
- $((R_i^j(x_{i-1}, K_0^j)||X_{\phi_i})\backslash C, tSpec_i)$. In this case the equality check among hashes does not succeed and the system gets stuck. $tSpec_i$ is always able to simulate it.