ICALP'07 & LICS'07 Workshop

FCS-ARSPA'07

Joint Workshop on
Foundations of Computer Security
and
Automated Reasoning for Security Protocol Analysis

July 8, 2007

Proceedings

Editors:
Pierpaolo Degano, Ralf Küsters, Luca Viganò, Steve Zdancewic

Contents

Preface

The Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07) was held in Wroclaw (Poland) on July 8, 2007, in association with ICALP'07 and LICS'07.

The workshop FCS-ARSPA'07 was the second edition of the fusion of two workshops: FCS and ARSPA, which joined forces in 2006 for FCS-ARSPA'06, which was affiliated to LICS'06, in the context of FLoC'06. The workshop FCS continues a tradition (initiated with the Workshops on Formal Methods and Security Protocols (FMSP) in 1998 and 1999, then with the Workshop on Formal Methods and Computer Security (FMCS) in 2000, and finally with the LICS satellite Workshop on Foundations of Computer Security (FCS) in 2002 through 2005) of bringing together formal methods and the security community. ARSPA is a series of workshops on Automated Reasoning for Security Protocol Analysis, bringing together researchers and practitioners from both the security and the formal methods communities, from academia and industry, who are working on developing and applying automated reasoning techniques and tools for the formal specification and analysis of security protocols. The first two ARSPA workshops were held as satellite events of the 2nd International Joint Conference on Automated Reasoning (IJCAR'04) and of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), respectively, and their proceedings have been published as volumes 125 and 135 of the Electronic Notes in Theoretical Computer Science. FCS and ARSPA have spawned special issues of the International Journal of Information Security (FCS 2002), of the Journal of Automated Reasoning (ARSPA'04), of Theoretical Computer Science and the International Journal of Information Security (ARSPA'05), and of Information and Computation (FCS-ARSPA'06).

The FCS-ARSPA'07 workshop brought together researchers and practitioners who are working on the foundations of computer security and on the development and application of automated reasoning techniques and tools for the formal specification and analysis of security protocols. There were 13 submissions of high quality, from countries in Asia, Europe, and North America. All the submissions were evaluated by at least three referees and the Program Committee then selected the 9 research contributions that are included in this volume. The workshop program was enriched by an invited talk by Srdjan Capkun, whose abstract is also included.

We would like to thank all the people who contributed to the organization of the FCS-ARSPA'07 Workshop. In particular, we are deeply grateful to:

– the other members of the Program Committee: Alessandro Armando, Anindya Banerjee, Massimo Bartoletti, Michele Boreale, Yannick Chevalier, Veronique Cortier, Cas Cremers, Volkmar Lotz, Cathy Meadows, Sebastian Mödersheim, David Naumann, Mark Ryan, Eijiro Sumii;

- the additional referees, who allowed us to review the papers in a very short time while maintaining a very high standard: Sebastian Bala, Andrea Bracciali, Marzia Buscemi, Stéphanie Delaune, Tomasz Truderung, Max Tuengerthal, Emilio Tuosto, Eugen Zalinescu, Roberto Zunino;
- Andrei Voronkov, who allowed us to use the free conference software system EasyChair, which greatly simplified the work of the Program Committee;
- last but not least, the organizers of ICALP'07 and LICS'07, who made all this possible.

The Program Chairs of the FCS-ARSPA'07 Workshop

Pierpaolo Degano
Dipartimento di Informatica
Università di Pisa (Italy)

Ralf Küsters
Department of Computer Science
ETH Zurich (Switzerland)

Luca Viganò
Dipartimento di Informatica
Università di Verona (Italy)

Steve Zdancewic
Department of Computer and Information Science
University of Pennsylvania (USA)

From Securing Navigation Systems to Securing Wireless Communication Through Location-Awareness[⋆]

Srdjan Capkun

ETH Zurich, Switzerland
`http://www.syssec.ethz.ch/people/capkun`

Recent rapid development of wireless networks of sensors, actuators and identifiers dictates the digitalization of our physical world and the creation of the "internet of things". In this new internet, each wireless device will sense and provide contextual information, of which crucial component are locations of devices and objects. In this talk, we present recent research results in secure computation and verification of locations of wireless devices: we show that current localization systems are highly vulnerable to attacks and we demonstrate that out solutions can prevent these attacks. We further illustrate how location-awareness can help in solving some of the fundamental security challenges of wireless networks, e.g., enabling authenticated and confidential communication without pre-shared keys of credentials.

[⋆] Invited talk of FCS-ARSPA'07 and WCAN'07.

Automated Security Analysis of Ad Hoc Routing Protocols

Todd R. Andel and Alec Yasinsac

Florida State University
Tallahassee, FL, 32306-4530
{andel, yasinsac}@cs.fsu.edu

Abstract. Mainstream evaluation approaches in the mobile ad hoc network routing community do not provide an automated or exhaustive security analysis capability. In this paper we offer an automated process to evaluate security properties in the route discovery phase for on-demand source routing protocols. We use the SPIN model checker to exhaustively evaluate protocol abstractions against an attacker attempting to corrupt the route discovery process.

Key words: Security analysis, model checking, formal methods, secure routing, mobile ad hoc networks.

1 Introduction

Mobile ad hoc networks (MANETs) are comprised of portable wireless nodes that do not use fixed infrastructure. Unlike wired networks that depend on fixed routers for network connectivity and message forwarding, each node in a MANET must provide routing functionality. Ad hoc routing protocols provide the core functionality enabling wireless nodes to communicate with nodes outside their local transmission radius. Ad hoc routing protocols [5, 17] commonly utilize two-phased routing approaches in which a route is first determined using a route discovery phase and data is then forwarded between a given source and destination pair over the identified route. Both the route discovery phase and data forwarding phase must be secured to protect the protocol operation from malicious activity. As secure routing protocols are being developed, they must be analyzed to ensure the protocol's intended security requirements are met. There are various techniques being used to analyze security properties in MANET routing protocols, to include visual inspection, network simulation, analytical proofs, simulatability models, and formal methods.

In this paper, we follow the formal methods approach by providing an automated model checking technique to evaluate the route discovery process against route corruption. For a given network topology, our models exhaustively check all routing combinations to evaluate if an attacker can corrupt the route discovery phase by successfully returning routes that are not consistent within the network topology. Our protocol models utilize the SPIN model checker [16] to provide exhaustive analysis for the specified protocol attackers. While SPIN has been used to evaluate MANET routing protocol operation (e.g., loop freedom) [8, 15, 28], to the best of our knowledge we are the first to use SPIN to evaluate security properties in ad hoc routing protocols.

In the remainder of this paper, we discuss requirements for secure routing and the current analysis techniques being used to evaluate secure ad hoc routing protocols. We provide a brief background on the SPIN model checker. The primary contribution is the secure routing model abstraction, including abstractions over the wireless medium, the routing protocol, and the attacker abstraction. We illustrate the defined automated analysis of routing attacks against the Secure Routing Protocol (SRP) [24] and conclude with future work.

2 Current Secure Routing Evaluation Techniques

As secure routing protocols are being developed, it is vital they receive the appropriate analysis to determine if the security goals are being met and to identify under what adversarial environments the protocols may fail. We first define security requirements for a secure routing protocol and discuss the current techniques used to evaluate security within this context.

2.1 Requirements for Secure Routing Protocols

The term security protocol traditionally refers to *authentication protocols*, or *cryptographic protocols*, where the goal is to securely share information (e.g., a message or a session key) between two nodes. Security analysis for authentication protocols evaluates if it is possible for a third party (i.e., the adversary) to obtain access to the protected key, regardless of intermediate nodes within the communication path [26]. Conversely, security evaluations for MANET secure routing protocols must consider actions taken by intermediate nodes. That is, we must consider whether the intermediate nodes can impact the secure routing protocol's intended goal. More specifically, we must consider route accuracy (securing the route discovery phase) and protocol reliability (securing the data forwarding phase).

A routing protocol is considered to maintain *route accuracy* if it produces routes that exist within the current network topology. Route accuracy is an integrity issue, ensuring that a malicious attacker has not corrupted the path obtained during the route discovery phase. Since the routes obtained during route discovery can fail due to both malicious actions and non-malicious failures (e.g., mobility, hardware failures, etc.), the routing protocols must also provide *reliability*. Once route paths begin to fail, reliability mechanisms identify that the path is no longer operating and initiate a new route discovery process or select an alternate path if multi-path protocols [9, 19] are being utilized. Reliability mechanisms may also attempt to detect and remove malicious nodes via probing protocols [6].

2.2 Current Evaluation Techniques

There are numerous proposed approaches to evaluate MANET routing protocol security properties. These approaches include visual inspection, network simulation, analytical methods, simulatability models, and formal methods.

Visual inspection relies on human intuition to find attacks. While human reasoning is powerful in finding attacks, it cannot provide an exhaustive approach to determine if a protocol is vulnerability free and cannot guarantee a given attack does not exist when analyzing large or complex distributed systems, such as ad hoc routing protocols.

Network simulation packages, such as *ns-2*, *GloMoSim*, and *OPNET*, are primarily used to project average case performance, focusing on packet deliverability ratios, network overhead, and network delay. However, it is difficult to use network simulation to determine if a security property holds in all situations or not, as an attack possibility is not necessarily statistical in nature. While network simulation can identify overhead costs and indicate how a routing protocol may mitigate certain attacks, simulation cannot provide an exhaustive method to claim protocol security since it focuses on the resulting performance effects provided by security enhancements. Just because an attack is not likely or does not show up during a statistical simulation run, does not mean a protocol is secure against the attack.

Analytical proof systems, as used to prove deliverability in single-phased routing protocols [11], use mathematical properties, theorems, and lemmas to prove or disprove security properties. While proof systems can guarantee message deliverability in some cases, proof systems are highly tailored to individual problems, dependent on individual researcher capabilities, and are not easily generalizable nor are they automated.

Simulatability models leverage the rigor provided by analytical proofs by adapting a technique historically used to provide cryptographic proof systems [7]. Simulatability models prove a protocol secure if an attacker has no greater advantage over a *real* protocol than it has over an abstracted *ideal* protocol. The ideal protocol contains complete knowledge over a system through the use of a trusted third party, or oracle. The oracle knowledge renders the attacker powerless in the ideal model.

The simulatability model has been adapted to evaluate route security for MANET routing protocols [1, 2, 12]. If a protocol is not proven secure via the simulatability approach, the results indicate an attack must be possible. The attack itself must then be identified using another technique, such as visual inspection. Additionally, provably secure routing protocols are only secure under the attacker assumptions in which the protocol was analyzed. Attackers with different capabilities may render a provably secure protocol as insecure [3].

Formal methods have been used in isolated cases to evaluate route security in ad hoc routing protocols. When using formal methods, a system (e.g., a protocol) and its desired properties (e.g., a security property) are formally specified in a specified mathematical or semantic description [13]. Once specified, theorem proving or model checking techniques can be used to evaluate the system for failures. Theorem proving is a manual process to prove the system outcome using the formal semantics and axioms defined by the formal method being utilized.

Model checking [14] provides an automated process in which a finite system model is created and then exhaustively searched to determine if the given security property holds or fails within the model abstraction. If the system fails, model checking can provide an event sequence that leads to the failure, thus automatically finding protocol security failures. In the context of security protocol analysis, automated formal methods have shown success in analyzing authentication protocols [20, 22, 27, 30]. In isolated cases [23, 29], automated formal methods have also been used to evaluate MANET route security for ad hoc on-demand distance vector protocols such as AODV and Secure AODV (SAODV), where the attacker's goal is to corrupt a node's next-hop entry for destinations the node may hold in its routing table.

Our research follows the model checking approach to evaluate route accuracy in on-demand source routing protocols. In MANET on-demand source routing protocols, such as the Dynamic Source Routing (DSR) protocol and the Secure Routing Protocol (SRP), the complete path to a destination is explicitly determined during route discovery and subsequently embedded into each packet during the data forwarding phase. We develop our models for security analysis using the SPIN model checker. Our models allows automated evaluation, producing attack sequences when a protocol vulnerability is discovered.

3 The SPIN Model Checker

The SPIN model checker, developed by Holzmann [16], is designed to verify correctness properties for concurrent (or distributed) systems. To use SPIN, a protocol abstraction and its desired goals are specified in the Promela (Process Meta Language) formal modeling language. SPIN generates a finite state automaton (FSA) and verifies the claimed goal through exhaustive reachability analysis. If the protocol analysis encounters a failure according to the specified criteria, a counter-example is produced showing the event sequence leading to the discovered failure.

One of the factors into choosing SPIN for our evaluation is that SPIN has been used to verify security properties in authentication protocols [21] and to formally identify loop-free failures (i.e., correctness in non-malicious environments) in MANET routing protocols [8, 15, 28]. Our research combines these two areas to model MANET routing protocol security properties using SPIN.

Other factors in choosing SPIN include Promela's ability to model message channels and the ability to model distributed processes as independent system threads. These factors are key to modeling routing interaction between asynchronous wireless nodes. Another key feature to the Promela language is the ability to code non-deterministic choices. For instance, the *if* statement below has more than one entry condition, leading to different executable sections:

```
if
:: conditional statement 1 ->
       actions taken when statement 1 is true
:: conditional statement 2 ->
       actions taken when statement 2 is true
fi .
```

If both conditional statements are true, then either true condition is non-deterministically chosen and its corresponding actions are executed. The non-deterministic choice depends on the seed value chosen during SPIN simulation runs. For SPIN exhaustive analysis, all possible non-deterministic choices are examined independently. Using the non-deterministic *if* construct provides the ability to model message deliverability in a wireless environment, where message deliverability is not guaranteed. The possibility of dropped messages enables attackers to inject corrupted routing information into the route discovery process.

If exhaustive analysis is not possible, SPIN can utilize a bit-state hashing method to use two bits to store a system state. Since bit-state hashing does not provide 100% state coverage, a system evaluation showing no errors does not guarantee that an attacker does not exist. However, any failure found via bit-state hashing provides a correct attack sequence. Exhaustive analysis should be performed if possible. SPIN increases its chances for exhaustive analysis by providing a state-space compression option that trades off execution time, and through the use of partial order reduction. The partial order reduction technique refers to a state-space elimination process that can occur if any concurrent process statement interleavings result in the same outcome regardless of their execution order.

4 Model Checking Secure Routing

Our model abstraction focuses on the MANET route discovery phase, analyzing if the protocol can defend against an attacker attempting to inject false routing information into the discovered routes. The model abstracts only the required protocol elements in order to evaluate the desired security property. Our modeling focuses on source routing protocols, where the route is explicitly embedded into the message. As illustrated in Fig. 1, we use three primary process types: the wireless medium, non-malicious node, and attacker node. We also note the global connectivity array which specifies the current network topology for the given evaluation.

4.1 Eliminating Non-Malicious Failures

Before presenting our model development, we first analyze non-malicious route failure impact in our MANET protocol abstraction. Non-malicious failures can occur due mobility or failed nodes, either of which affects the protocol's message deliverability performance. MANET routing protocols alleviate non-malicious failures through reliability mechanisms, such as reinitiating route discovery or using multipath protocols. Our approach is to model the protocol messages and associated security goals rather than performance issues. Incorporating non-malicious failures may not be

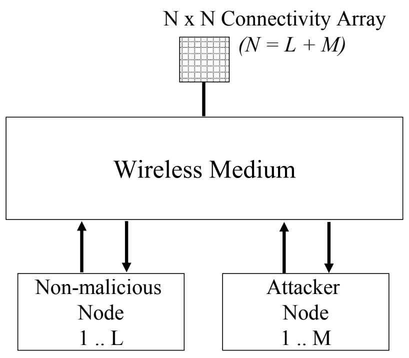

Fig. 1. MANET Model Abstraction

feasible and may inject meaningless results into the analysis. For instance, consider the analysis feasibility of a model checker that includes mobility in the model. Work in [28] to verify protocol operation shows that it is not feasible to model mobility in model checking paradigms due to rapid state-space explosion.

On the other hand, if we assume that we could model non-malicious failures without limitations, the source of any routing inconsistencies would be inconclusive. Consider analyzing a system to determine the ability of a MANET protocol to achieve a route consistent with the current network topology. If the route fails within this model, it is impossible to differentiate whether the failure was natural or malicious. Including non-malicious failures is one of the limiting factors in using network simulation packages to evaluate route security. Routing failures discovered using network simulation as a security evaluation tool cannot be differentiated between malicious or non-malicious activity unless mobility and other non-malicious failure causes are removed.

Based on these observations, we eliminate non-malicious routing failures from our model development. Our abstraction allows us to effectively isolate discovered route failures due to malicious activity. However, we must ensure that removing such items maintains the modeled protocol's security semantics. That is, by removing non-malicious routing failures we must be careful not to change or bias the protocol's security functionality or vulnerabilities.

We justify eliminating non-malicious failures from our abstraction based on the adversary advantage definitions presented in [10]. The *adversary advantage* is the probability that an attacker can make a protocol fail. A protocol can have either *Γ-availability* or *Γ-tolerance*, where Γ is the attacker node(s) within the network. A protocol maintains *Γ-availability* if the adversary advantage is negligible for all *Γ-adversaries*, which is considered perfect security. Perfect security is not attainable in MANETs, since attacks such as the invisible node attack [4] cannot be completely eliminated and undetected compromised insiders can drop packets at any time.

A protocol maintains *Γ-tolerance* if the |*adversary advantage - non-malicious protocol failure*| is negligible. That is, if a protocol works $X\%$ of the time without an adversary, does the presence of an adversary reduce X? Following the reasoning behind *Γ-tolerance*, we can effectively model security features without including non-malicious failures. The failures discovered in our model will therefore be attributable only to malicious activity.

Eliminating non-malicious failures while analyzing route discovery security can also be intuitively justified. Consider node mobility during the route discovery process. If node mobility disrupts an expected route reply for a given path, the source never receives that respective route reply. Therefore, no actual route is returned to be evaluated to determine if it is a valid route according to the network topology. The original route no longer exists due to mobility, not due to a malicious attacker corrupting the route discovery process.

4.2 Modeling the Wireless Medium

The first priority in building a SPIN model for wireless ad hoc networks is to implement message transmission. When a wireless node transmits a message, all nodes within its transmission footprint receive the communication. The wireless recipients must then determine if it must process or simply drop each packet.

Given that SPIN does not natively support broadcast communication, we must implement a way to model the required communication components. Since time is implicitly modeled by evaluating all computational possibilities, we can transmit a broadcast message to all the intended recipients via individual unicast messages. In Ruys' dissertation [25], he discusses implementing simple broadcast communication as a bus, a matrix of channels for each node combination, or a separate process to model the broadcast service. Ruys describes how each of these options can be implemented in SPIN along with their associated features and limitations.

We choose to model MANET communication dedicating the wireless medium as a separate process, thus limiting the state-space and simplifying our communication into a modularized approach for future development. We must also consider reachable neighbors when modeling the wireless broadcast process. We use a two-dimensional array to hold the network topology for the time instance we are evaluating. As an example, Fig. 2 shows the associated array for a four node network topology.

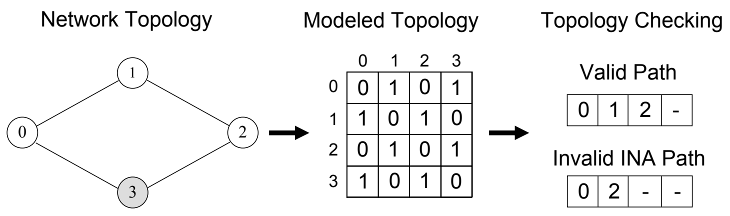

Fig. 2. MANET Model Abstraction

The model's wireless medium process uses the array to determine which neighbors are within the transmission of a given node and sends the messages accordingly. The array is composed of N rows by N columns, where N is the total number of non-malicious (L) nodes plus the total number of malicious (M) nodes. Rows indicate the from node and columns indicate the nodes' local neighbors. Each location holds a Boolean value, with true (or 1) indicating a connection exists between node pairs.

While the general process modeled into the wireless medium can support message transmission in any MANET routing protocol, we tune the wireless medium to model on-demand source routing protocols. By tuning the model abstraction to our research focus, we maintained a reduced state-space.

Our wireless medium server approach allows route request (RREQ) messages to be broadcast to each adjacent node, where each message is then processed locally and sent back to the wireless medium server for its next transmission. The route reply (RREP) process can use this same procedure with the exception that each recipient within the footprint of the transmitted message must determine if it is included in the explicit path before it decides to accept and retransmit the message. Since we are trying to minimize the required state-space, the server only transmits to the intended recipient identified in the unicast RREP message and to all attacker nodes within the current transmission footprint. This approach reduces the state-space yet still captures the protocol's required elements for security analysis.

4.3 Modeling Source Routing Protocols

MANET two-phased routing protocols can be *reactive* or *proactive*. In reactive protocols, a route is only established between a source and destination when required, as opposed to the proactive approach which attempts to maintain current routes between all source-destination pairs at all times. Reactive protocols, also known as on-demand protocols, can utilize distance vector mechanisms or use source routing. On-demand distance vector protocols, such as AODV, use tables in each node to track what the next hop is for a given destination. On-demand source routing protocols, such as DSR, explicitly embed the path into each message.

Our focus is automating the security analysis process for evaluating routing attacks against the route discovery phase for on-demand source routing protocols. Our current modeling environment includes the DSR protocol and the SRP extension to DSR. All abstraction choices are made to simplify the resulting model and guard against state-space explosion. As our security analysis focus is to determine if the route returned from the route request process is valid, we must ensure the abstraction captures all route discovery possibilities. Model checking over the abstraction exhaustively examines all routing possibilities. Our model abstraction does not reflect message timing issues, as messages themselves may be lost. Therefore, we search for possible route violations rather than probable outcomes.

Modeling DSR. We assume the non-malicious nodes following the routing protocol use bi-directional links, thus any target receiving a route request (RREQ) will return the route reply (RREP) over the accumulated path received in the RREQ message. Figure 3 illustrates The Dynamic Source Routing (DSR) protocol [18] route discovery process.

The DSR route discovery message format can be viewed as:

$$< msg\_type, initiator, target, id, accum\_path > .$$

The *msg_type* tags the message as a RREQ or a RREP, with the initiator and target as the source-destination pair for the desired route. The *id* is a unique identifier to ensure a node only forwards a given RREQ the first time it sees the request. As many RREQs occur in an operational setting, each node tracks the $< initiator, id >$ for each RREQ it receives. In our abstraction we focus on a single route discovery iteration at a time; therefore, we remove the id from the message and implement a simple Boolean flag in each node that tracks if a given node has sent a

```
RREQ process:
    • Initiator node
        ◦ Initiate a RREQ to target
    • Intermediate nodes
        ◦ If previously seen RREQ → drop
        ◦ Else
            ⋄ If target → generate RREP
            ⋄ If not target → append id to path and retransmit
RREP process
    • Target node
        ◦ Unicast the RREP
    • Intermediate nodes (along unicast path)
        ◦ If initiator → accept route
        ◦ If not initiator → rebroadcast
```

Fig. 3. DSR Route Discovery

RREQ. The accumulated path (*accum_path*) lists all intermediate nodes in the path and is updated at each node that forwards the RREQ. In our implementation we list all nodes (to include the initiator and target) in the accumulated path, allowing us to use this field to check against the network connectivity graph during analysis. We also add a position value (*accum_pos*) to track the array element that the current node adds its own *id* to. Our abstracted DSR message format follows as:

$$< msg\_type, initiator, target, accum\_path, accum\_pos > .$$

Once the RREQ is delivered to the intended target, a RREP is generated and sent back to the initiator. During the RREP, the accumulated path and the current array position are read by the wireless medium server to determine the next-hop destination node for the unicasted RREP. As previously mentioned, wireless RREP messages are received by all nodes within the current node's transmission footprint and each receiving node determines if it is the intended relay in the unicast packet. To reduce the abstraction state-space, we make the relay decision in the wireless medium process and send a single message to the appropriate node. We also send the transmission to any malicious node that may be within the current transmission footprint.

Modeling SRP. The Secure Routing Protocol (SRP) [24] provides an extension to DSR, attempting to secure the route discovery phase from maliciously corrupted routes. SRP assumes an existing security association between an initiator and target node. Intermediate nodes do not utilize any cryptographic mechanisms during the route discovery process, therefore, they interact following the underlying protocol process for DSR.

The SRP RREQ format is:

$$< RREQ, \mathbf{initiator}, \mathbf{target}, \mathbf{Q_{id}}, \mathbf{Q_{sn}}, MAC_{it}, accum\_path > .$$

The SRP RREP format is:

$$< RREP, initiator, target, Q_{id}, Q_{sn}, MAC_{it}, \mathbf{accum\_path} > .$$

The bold message portions indicate which message parts the message authentication code (MAC), subsequently stored in MAC_{it}, is generated over using the security association between the initiator and target. The route discovery process is similar to DSR, with changes added for the MAC generation and associated checks in the initiator and target nodes.

The query id (Q_{id}) and query sequence number (Q_{sn}) are used to ensure the route request is unique and has not been replayed. Similar to DSR, the intermediate nodes check the Q_{id} to ensure only the first RREQ that is received is forwarded for the given $< initiator, Q_{id}, Q_{sn} >$ value. Following our DSR abstraction for a single protocol round we do not include the Q_{id} and Q_{sn}; however, the intermediate nodes will track if they have already forwarded the single round route request and not respond to more than one RREQ. Since the MAC during the RREQ ensures against replay attacks, we focus on possible attacks that corrupt the path discovered by the routing process. We target our model abstraction for attack analysis against the accumulated path, ensuring the SRP abstraction can properly account for the target MAC over the accumulated path. Our abstracted SRP message format follows as:

$$< msg\_type, initiator, target, accum\_path, accum\_pos, mac\_path > .$$

During the route discovery process, the target node models the MAC over the accumulated path by copying the path received in the RREQ into the *mac_path* variable and adding it to the RREP. The MAC is protected against attacker corruption (other than blind bit manipulation), since it can be accessed only by the initiator-target pair due to their security association. Once the initiator node receives the RREP, it checks the accumulated path against the path contained in the *mac_path* value before accepting the route.

4.4 Modeling the Attacker

Our security analysis model isolates the attacker actions into an individual process. This simplifies later adaptations to the attacker capabilities and goals against the various protocols. When modeling a different attacker, the attacker process is simply replaced. The attacker cannot break any cryptographic mechanism, but is not forced to follow the routing protocol operations. In the attacker modeling that follows, we describe actions within the context of SRP. The network topology used is illustrated in Fig. 4, where Node 0 is the initiator, Node 3 is the target, and Node 4 is the attacker node.

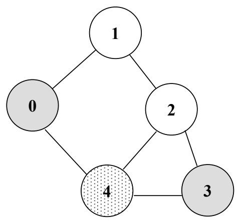

Fig. 4. Five-Node Evaluation Topology

The Invisible Node Attack. The invisible node attack (INA) [4] occurs when an attacker refuses to add itself to the routing path during route discovery. The attacker model relays the forward RREQ message without adding itself to the accumulated path and relays any subsequent RREP. As an example, an INAer using the network topology in Figure 4 results in the path *0-3* being returned instead of the actual path *0-4-3*. Even though the attacker is not part of the unicast RREP, our model ensures the attacker receives any wireless message transmitted within its reception range. The attacker does not expect to be within the accumulated path and simply relays the RREP to the next upstream node in the accumulated path.

The Node Drop Attack. The INA may be intuitively described as a relay attack. Conversely, the node drop attack (NDA) occurs when a node corrupts a route by proactively dropping a node from the accumulated routing path embedded into the route discovery messages. Relating the NDA to our SRP model development, recall the actions SRP takes to protect the accumulated path against routing attacks is for the target to calculate a MAC over the path extracted from the RREQ and include the MAC in the RREP. Attacks corrupting the RREP will be detected by SRP since the MAC value will not match the corrupted path. Therefore, the goal of the SRP attacker is to corrupt the RREQ phase to trick the target into computing and returning a MAC for a corrupt route. The target will compute a MAC over any path that is delivered with the RREQ, to include paths that have been corrupted and do not contain valid routes.

Our modeled NDAer provides complete automation without requiring any *a priori* knowledge of the network configuration and does not rely on any intervention from the security analyst. During the RREQ process the attacker removes the previous node from the accumulated path and replaces itself in the previous node's place, as long as the previous node is not the initiator. Once the corrupted route reaches the target, it is included in the MAC calculation. During the RREP process the attacker relays the unicast RREP to all upstream nodes from the embedded path, attempting to deliver the corrupted packet to the initiator or a node in the upstream path that can eventually reach the initiator. The attacker cannot use the original stripped node to relay the RREP on the way back to the initiator, as the stripped node is no longer part of the embedded unicast path.

The only requirement for the NDA attack to be successful against SRP is that the attacker must be able to successfully transmit the corrupted path during the RREP to any upstream node from the accumulated path. The ability to reach the initiator without using the stripped node depends on the current network topology.

As an example, a NDA occurs when Node 4 in Figure 4 receives a RREQ packet from Node 2 containing the current accumulated path *0-1-2*. Node 4 strips off Node 2 and adds itself to the accumulated path, transmitting a RREQ to the target node containing the path *0-1-4*, resulting in *0-1-4-3* as the complete path between the initiator and target node. While Node 3 may have already serviced a RREQ for the path *0-1-2-3*, RREQ messages are not guaranteed deliverability in a wireless environment, forcing us to consider all possible attack path sequences during the security analysis. During the route reply process, Node 4 simply attempts to relay the RREP to all upstream nodes contained in the accumulated path. That is, Node 4 will attempt to send a message to Node 0 and Node 1. The message intended for Node 0, the initiator, can successfully be delivered, returning the corrupted path *0-1-4-3* to the initiator. The SRP initiator accepts the corrupted path as valid, since the corrupted path matches the MAC value computed by the target node.

5 Automated Analysis

In addition to modeling the protocol and attacker, we must also specify our desired security property. For our analysis purposes, ϕ is defined as:

$$\phi = \text{returned routes must exist in the current topology} .$$

Once the protocol model and desired properties are specified, formally stating that the system model M models the given security property ϕ over all σ computational paths is captured by the following equation:

$$(M \models \phi) \leftrightarrow \forall \sigma, (\sigma \in M \rightarrow \sigma \models \phi). \tag{1}$$

To evaluate if $M \models \phi$ in SPIN we add an analysis check for the path once it is received and accepted by the node that initiated the route discovery process. The node takes the returned path and checks each link to ensure that link exists in the model's connectivity array. If any link check fails, an assertion violation is raised and SPIN halts execution, creating a trail file that lists the event sequence leading to the failure. For example, Fig. 2 shows how an invisible node attack sequence produces a path that is not consistent with the topology check, since the row for Node 0 does not indicate a link exists to Node 2.

During SPIN simulations, the chosen process interleaving sequence and the resulting chosen path is determined by the initial seed value. Depending on the seed, a given simulation run may or may not produce an error. Once SPIN compiles the model into an exhaustive verifier, the verifier searches all possibilities and finds the violation (i.e., a successful attack) if one exists within the evaluated network topology.

To highlight the differences between a simulator and an exhaustive model checker, we evaluate the SRP model development against the network topology specified in Fig. 4. When using SPIN as a simulator, the outcome is directly related to the initial seed value and simulator outcomes cannot guarantee an attack does not exist. The seed determines the process interleaving at each step, since each node is a concurrent process and can operate independently. Additionally, the seed determines whether a node receives or drops any incoming message, which allows the possibility for any path to be exercised during the route discovery round.

During analysis over the NDA against the forward routing process our analysis should detect an error when SRP accepts a corrupt route. SPIN simulations found non-attacked valid routes for the paths *0-4-3* (with seed set to 17) and *0-4-2-3* (with seed set to 173). After performing 300 simulations using different seed values, we did not find a case where SRP accepts a corrupt route. Seed dependent simulation may therefore lead us to believe that SRP is secure against this attacker, however, the attack indeed exists.

SPIN also allows interactive simulations. Performing an interactive SPIN simulation allows the analyst to choose the outcome for all non-deterministic decision points, showing that the attack is possible. While interactive simulation requiring human intervention for each decision point can produce the attack, it is a grueling process in which the analyst could miss an attack.

The power in our analysis model lies in SPIN's exhaustive verification method. During verification, SPIN automatically checks all possible message orderings until it finds that an attack is successful. The SPIN verifier automatically generates a trail file that can then be read into the SPIN simulator to view the successful attack, without having to guess the seed to simulate the attack or interactively walk through all the simulation choices. Evaluating SRP against the NDA attack using SPIN's exhaustive verifier identifies the corrupted path *0-1-4-3*, which SRP inappropriately accepts as a valid path.

6 Conclusion

In this paper we utilize the SPIN model checker to provide an automated analysis technique to evaluate route corruption attacks against the route discovery phase for on-demand source routing protocols. The current security analysis techniques used to evaluate security properties in MANET routing protocols are not automated (e.g., visual inspection, analytical proofs, and simulatability models) or do not provide exhaustive attacker analysis (e.g., current network simulation packages evaluate performances characteristics).

While SPIN has been previously used to evaluate security in authentication protocols and evaluate loop-freedom criteria in MANET routing protocols, our work combines these two areas to provide security analysis for MANET routing protocols. Our modeling automatically identifies route discovery attacks against SRP. We are currently developing SPIN models to study route discovery attacks against the Ariadne protocol.

An additional automated process we are currently developing is the ability to search all possible network topology configurations for a given number of nodes. One of the biggest impediments to manual analysis methods (e.g., visual inspection) is the intuition to choose a network configuration in which the attack exists. We also encounter this limitation in our own current automated attack modeling. For instance, the attacks we discovered automatically with SPIN are dependent on the attacker's ability to relay the corresponding RREP to a neighbor in the upstream source routing path. The attacks were possible to detect within the topology we chose to evaluate, corresponding to the network configuration in Fig. 4. If that network topology did not contain a link between Node 0 and Node 4, the automated evaluation process would not have found the NDA against SRP, indicating that SRP was secure against the NDAer. To solve this problem and automate the entire process, we are developing Perl scripts to produce a separate SPIN model file for each possible network configuration for the desired node population. Each file is subsequently analyzed with SPIN, identifying any configuration producing attacks along with the attack sequence.

Acknowledgments. Todd Andel is supported by the U.S. Air Force. Alec Yasinsac's work is in part supported by U.S. Army Research Laboratory and the U.S. Army Research Office under grant W91NF-04-1-0415. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

References

1. Ács, G., Buttyán, L., Vajda, I.: Provable security of on-demand distance vector routing in ad hoc networks. European Workshop on Security and Privacy, Vol. 3813. Springer-Verlag (2005) 113-127
2. Ács, G., Buttyán, L., Vajda, I.: Provably secure on-demand source routing in mobile ad hoc networks. IEEE Transactions on Mobile Computing **5** (2006) 1533-1546
3. Andel, T.R.: Can ad hoc routing protocols be shown provably secure. Computer Science Department. Florida State University, Tallahassee, FL (2006)
4. Andel, T.R., Yasinsac, A.: The Invisible Node Attack Revisited. 2007 IEEE SoutheastCon, Richmond, VA (2007) 686-691
5. Argyroudis, P.G., O'Mahony, D.: Secure routing for mobile ad hoc networks. IEEE Communications Surveys & Tutorials **7** (2005) 2-21
6. Awerbuch, B., Holmer, D., Nita-Rotaru, C., Rubens, H.: An on-demand secure routing protocol resilient to byzantine failures. 3rd ACM Workshop on Wireless Security. ACM Press, Atlanta, GA, USA (2002) 21-30
7. Beaver, D.: Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. Journal of Cryptology **4** (1991) 75-122
8. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. Journal of the ACM **49** (2002) 538-576
9. Burmester, M., Van Le, T.: Secure multipath communication in mobile ad hoc networks. International Conference on Information Technology: Coding and Computing (ITCC '04), Vol. 2 (2004) 405-409

10. Burmester, M., Van Le, T.: Provably secure routing for MANETs. in-submission (2006)
11. Burmester, M., Van Le, T., Yasinsac, A.: Adaptive gossip protocols: Managing security and redundancy in dense ad hoc networks. Ad Hoc Networks **5** (2007) 313-323
12. Buttyán, L., Vajda, I.: Towards provable security for ad hoc routing protocols. 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks, Washington DC, USA (2004) 94-105
13. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Computing Surveys **28** (1996) 626-643
14. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA (1999)
15. de Renesse, F., Aghvami, A.H.: Formal verification of ad-hoc routing protocols using SPIN model checker. 12th IEEE Mediterranean Electrotechnical Conference, Vol. 3 (2004) 1177-1182
16. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering **23** (1997) 279-295
17. Hu, Y.C., Perrig, A.: A survey of secure wireless ad hoc routing. IEEE Security & Privacy **2** (2004) 28-39
18. Johnson, D., Maltz, D.: Dynamic source routing in ad hoc wireless networks. In: Imielinski, T., Korth, H. (eds.): Mobile Computing. Kluwer (1996) 153-181
19. Kotzanikolaou, P., Mavropodi, R., Douligeris, C.: Secure Multipath Routing for Mobile Ad Hoc Networks. Second Annual Conference on Wireless On-demand Network Systems and Services (WONS '05) (2005) 89-96
20. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Tools and Algorithms for the Construction and Analysis of Systems, Vol. 1055. LNCS (1996) 147-166
21. Maggi, P., Sisto, R.: Using SPIN to Verify Security Properties of Cryptographic Protocols. 9th international SPIN Workshop on Model Checking of Software, Vol. 2318. Springer-Verlag (2002) 187-204
22. Meadows, C.: The NRL Protocol Analyzer: An Overview. The Journal of Logic Programming **26** (1996) 113-131
23. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theoretical Computer Science **367** (2006) 203-227
24. Papadimitratos, P., Haas, Z.J.: Secure routing for mobile ad hoc networks. SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), San Antonio, TX (2002)
25. Ruys, T.C.: Towards effective model checking. Department of Computer Science. University of Twente, Deventer, The Netherlands (2001)
26. Ryan, P., Schneider, S.: Modelling and Analysis of Security Protocols. Addison-Wesley, Harlow, England (2001)
27. Song, D., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. Journal of Computer Security **9** (2001) 47-74
28. Wibling, O., Parrow, J., Pears, A.: Automatized Verification of Ad Hoc Routing Protocols. Formal Techniques for Networked and Distributed Systems (FORTE 2004), Vol. 3235. Springer-Verlag (2004) 343-358
29. Yang, S., Baras, J.S.: Modeling vulnerabilities of ad hoc routing protocols. 1st ACM Workshop on Security of Ad hoc and Sensor Networks (2003) 12-20
30. Yasinsac, A., Wulf, W.A.: A Framework for A Cryptographic Protocol Evaluation Workbench. The International Journal of Reliability, Quality and Safety Engineering (IJRQSE) **8** (2001) 373-389.

Verifying Regular Trace Properties of Security Protocols with Explicit Destructors and Implicit Induction[⋆]

Adel Bouhoula[1] and Florent Jacquemard[2]

[1] École Supérieure des Communications de Tunis, Tunisia. bouhoula@planet.tn
[2] INRIA Futurs & LSV UMR CNRS–ENSC, France. florent.jacquemard@inria.fr

Abstract. We present a procedure for the verification of cryptographic protocols based on a new method for automatic implicit induction theorem proving for specifications made of conditional and constrained rewrite rules. The method handles axioms between constructor terms which are used to introduce explicit destructor symbols for the specification of cryptographic operators. Moreover, it can deal with non-confluent rewrite systems. This is required in the context of the verification of security protocols because of the non-deterministic behavior of attackers. Our induction method makes an intensive use of constrained tree grammars, which are used in proofs both as induction schemes and as oracles for checking validity and redundancy criteria by reduction to an emptiness problem. The grammars make possible the development of a generic framework for the specification and verification of protocols, where the specifications can be parametrized with (possibly infinite) regular sets of user names or attacker's initial knowledge and complex security properties can be expressed, referring to some fixed regular sets of bad traces representing potential vulnerabilities. We present some case studies giving very promising results, for the detection of attacks (our procedure is complete for refutation), and also for the validation of protocols.

1 Introduction

Inductive theorem proving techniques and tools have been successfully applied in last years to the verification of security protocols, both for proving security properties and for identifying attacks on faulty protocols.

Paulson's inductive approach [15] has been applied to many case studies. In this method, protocols are formalized in typed higher-order logic and the Isabelle/HOL interactive theorem prover is used to prove security properties. Paulson's technique handles infinite state protocols and does not assume any restriction on the number of protocol participants. However, it is not automatic and requires interaction with the user and also a good expertise (if a proof fails with Isabelle, it is difficult to conclude whether the proof attempt failed or the conjecture to be proved is not valid).

[⋆] This work has been partially supported by the grant INRIA–DGRSRT 06/I09.

Bundy and Steel [8] derive attacks on faulty protocols specified in first-order logic using a *proof by consistency* technique. Such a technique is sometimes also called *inductionless induction* [11] since it does not construct an induction proof following an induction schema but rather tries to automatically derive an inconsistency using first-order theorem proving techniques. This technique is hence fully automatic but its outcome may be difficult to analyze and convergence is difficult to achieve.

In this paper we present a new method for the formal verification of security protocols based on an *implicit induction* procedure. The protocol, the insecure communication network (attackers) and the security assumptions are modeled with an equational specification which is passed to an inductive theorem prover in order to validate the protocol or to derive an attack. The advantage of this procedure is that it is automatic and returns readable proofs or counter-examples.

The specifications are strongly typed and follow a constructor discipline: we distinguish in the signature the *constructor symbols*, used to build terms representing the values of the computation, in our case the list (trace) of messages exchanged. The other symbols, called *defined symbols*, represent functions defined on these values (for instance, we use below a predicate trace characterising the protocol traces) and are specified by Horn clauses.

Equational axioms between terms made of constructors are very difficult to deal with in automated induction and are generally not allowed. Our procedure however allows such axioms, and we use them in order to specify cryptographic operators like decryption. This approach with *explicit destructors* is the base of a uniform framework for the verification of security protocols in an insecure communication environment [1]. Explicit destructors both simplify the specification of attacker's capabilities and increase the expressiveness of specification as models with explicit destructors are strictly more expressive than models based on free algebra, in the sense that they captures more attacks [14].

Our induction procedure introduces another important novelty, compared to other implicit induction techniques, since it handles specifications which are *not confluent*. The property of ground confluence (any two divergent reduction sequences starting from the same ground term converge ultimately) is usually required for induction procedures. For the application to protocol verification, we consider a model with an active attacker which interferes non-deterministically with the communications of honest users. Such a model, relevant in the context of security, can not be expressed with a ground-confluent specification.

The axioms of the specification contain constraints such as equations, disequations and membership to fixed term languages (characterized by tree grammars). The membership constraints come almost for free as our induction procedure is based on constrained tree grammars. This feature appeared however extremely useful in a setting of protocol specification. It permits to parametrize the specifications of protocols and attackers with (possibly infinite) regular sets of user names or attacker's initial knowledge. Moreover, and more important, it also allows to deal with a rich language of security properties, not limited to the confidentiality of some fixed piece of data as it is the case in many approaches.

Indeed, assume given a set $\mathcal{B}$ of *bad traces* (lists of messages exchanged corresponding to an attack) which is a regular tree language (of constructor terms), characterized by a tree grammar. We can express as a conjecture to be proved the property that every protocol trace (as defined by the predicate trace) does not belong to $\mathcal{B}$. This allows to write a wide range of security properties, like for instance variants of authenticity.

In contrast to the technique of [8], implicit induction is a goal directed proof technique, and we believe that it is therefore quite efficient for automatically finding attacks on faulty protocols. The use of tree automata techniques permits in particular to focus on traces of events in normal form, and consequently to minimize the set of traces to be checked. Since our procedure is refutationally complete (under some conditions for the specification) its application on any flawed protocol will return a readable attack in finite (and typically very small) time and in a completely automatic way, as illustrated by the examples of Section 3. Moreover, it can also help in protocol validation (proof that there is no attack), though the interactive addition of lemmas may be required for that purpose, see the example in Section 4.

2 Preliminaries

We consider a many-sorted signature $\mathcal{F}$. As explained in introduction, $\mathcal{F}$ is partitioned into a subset $\mathcal{C}$ of *constructor symbols* and a subset $\mathcal{D}$ of *defined symbols*. Each symbol f is given with a profile $f : S_1 \times \ldots \times S_n \to S$ where $S_1, \ldots, S_n, S$ are sorts and n is the *arity* of f. We note $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C}, \mathcal{X})$) the set of well-sorted terms over $\mathcal{F}$ (resp. constructor well-sorted terms) with variables in $\mathcal{X}$ and $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$) and the subsets of variable-free terms, or *ground* terms. The *subterm* of t at position p is denoted by $t|_p$. The result of replacing $t|_p$ with s at position p in t is denoted by $t[s]_p$. This notation is also used to indicate that s is a subterm of t, in which case p may be omitted. A term t is *linear* if every variable occurs at most once in t. A *substitution* is a finite mapping from variables to terms, extended as usual as a morphism from terms to terms, written in postfix notation.

Conditional constrained rewriting. We shall consider below *constraints* which are Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where P belongs to a fixed language of constraints predicates interpreted over terms of $\mathcal{T}(\mathcal{C})$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. The *solutions* of a constraint c, whose set is denoted $sol(c)$, are (constructor) substitutions σ grounding for all terms in c and such that $c\sigma$ is interpreted to true. *Constrained terms* have the form $t \llbracket c \rrbracket$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and c is a constraint and *conditional constrained rewrite rules* are constrained Horn clauses such as:

$$u_1 = v_1, \ldots, u_n = v_n \Rightarrow \ell \to r \llbracket c \rrbracket$$

where $u_1, v_1, \ldots, u_n, v_n, \ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, the terms ℓ and r (called resp. left- and right-hand side of the rule, $\ell \to r$ is an oriented equation) are linear[3] and have the same sort, and c is a constraint. Our procedure takes as input a constrained constructor rewrite system (CTRS) $\mathcal{R}_\mathcal{C}$, which is a set of rules such that $n = 0$ (no conditions) and $\ell, r \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and a conditional constrained rewrite system (CCTRS) $\mathcal{R}_\mathcal{D}$, with rules such that $n \geq 0$, $\ell = f(\ell_1, \ldots, \ell_k)$ where $f \in \mathcal{D}$ and $\ell_1, \ldots, \ell_n, r \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. We note $\mathcal{R} = \mathcal{R}_\mathcal{C} \uplus \mathcal{R}_\mathcal{D}$.

A term $t\,[\![d]\!]$ rewrites to $s\,[\![d]\!]$ by the above rule, denoted by $t\,[\![d]\!] \xrightarrow[\mathcal{R}]{} s\,[\![d]\!]$, if $t|_p = \ell\sigma$ for some position p and substitution σ, $s = t[r\sigma]_p$, the substitution σ is such that $d \wedge \neg c\sigma$ is unsatisfiable and $u_i\sigma \downarrow_\mathcal{R} v_i\sigma$ for all $i \in [1..n]$; where $\downarrow_\mathcal{R}$ denotes $\xrightarrow[\mathcal{R}]{*} \circ \xleftarrow[\mathcal{R}]{*}$ and $\xrightarrow[\mathcal{R}]{*}$ is the reflexive transitive closure of $\xrightarrow[\mathcal{R}]{}$. If there exists such a term s, then $t\,[\![d]\!]$ is called *reducible*, otherwise it is called a *normal form*. A constrained term $t\,[\![c]\!]$ is *ground reducible* (resp. *ground irreducible*) by $\mathcal{R}$ if for every irreducible substitution $\sigma \in sol(c)$ grounding for t, $t\sigma$ is reducible (resp. irreducible) by $\mathcal{R}$.

The CCTRS $\mathcal{R}$ is *terminating* if there is no infinite sequence $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \ldots$, and $\mathcal{R}$ is *ground-confluent* if for any ground terms $u, v, w \in \mathcal{T}(\mathcal{F})$, $v \xleftarrow[\mathcal{R}]{*} u \xrightarrow[\mathcal{R}]{*} w$, implies that $v \downarrow_\mathcal{R} w$, and $\mathcal{R}$ is *ground convergent* if $\mathcal{R}$ is both ground-confluent and terminating.

Inductive theorems, tautologies Let $\mathcal{R}$ be a terminating CCTRS. A constrained equation $a = b\,[\![c]\!]$ is called an *inductive theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{ind} a = b\,[\![c]\!]$) if for all substitution $\sigma \in sol(c)$ grounding for a and b, $a\sigma \xleftrightarrow[\mathcal{R}]{*} b\sigma$, and it is called a *joinable inductive theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{jind} a = b\,[\![c]\!]$) if for all substitution $\sigma \in sol(c)$ grounding for $a = b$, and all $\mathcal{R}$-normal forms n_a, n_b respectively of $a\sigma, b\sigma$, we have $n_a = n_b$. These notions are extended to clauses as expected. The definition of joinable inductive theorems is motivated by the applications presented in Sections 3 and 4.

Note that the two notions of inductive and joinable inductive theorem coincide when $\mathcal{R}$ is ground-confluent. However, they can differ otherwise. Consider for instance $\mathcal{R} = \{c \to a, c \to b\}$. The conjecture $a = b$ is an inductive theorem (since $a \xleftrightarrow[\mathcal{R}]{*} b$) but it is not a joinable inductive theorem (as a and b are $\mathcal{R}$-normal forms). On the other hand, $a = b \Rightarrow f(x) = c$ is a joinable inductive theorem but not an inductive theorem (for the same reasons).

We call *tautology* of $\mathcal{R}$ a constrained clause of the form $a = a \vee L\,[\![c]\!]$ such that a is ground irreducible by $\mathcal{R}$ or of the form $a = b \vee a \neq b \vee L\,[\![c]\!]$ such that a and b are ground irreducible by $\mathcal{R}$. Note that every tautology is both an inductive and a joinable inductive theorem of $\mathcal{R}$.

Constrained tree grammars A *constrained tree grammar* $\mathcal{G} = (Q, \Delta)$ is given by a finite set Q of *non-terminals* of the form $\llcorner u \lrcorner$, where u is a linear term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and a finite set Δ of *production* rules of the form $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$ where $f \in \mathcal{F}$, $\llcorner t \lrcorner$, $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q$ and c is a constraint.

[3] Note that assuming that l and r are linear is not restrictive since non linearities may be expressed as equalities between variables in c.

$$\begin{array}{lll} \mathsf{fst}\big(\mathsf{pair}(x_1,x_2)\big) \to x_1 & \mathsf{snd}\big(\mathsf{pair}(x_1,x_2)\big) \to x_2 & \mathsf{inv}\big(\mathsf{inv}(y)\big) \to y \\ \mathsf{dec}\big(\mathsf{enc}(x,y),y\big) \to x & \mathsf{adec}\big(\mathsf{aenc}(x,y),\mathsf{inv}(y)\big) \to x & \mathsf{adec}\big(\mathsf{aenc}(x,\mathsf{inv}(y)),y\big) \to x \end{array}$$

Figure 1: Constructor rules

The non-terminals are always considered modulo variable renaming. In particular, we assume that the term $f(u_1,\dots,u_n)$ is linear. The production relation $\vdash^x_{\mathcal{G}}$ on constrained terms is defined by:

$$t[x]\,[\![x{:}\llcorner u\lrcorner \wedge d]\!] \vdash^x_{\mathcal{G}} t[f(x_1,\dots,x_n)]\,[\![x_1{:}\llcorner u_1\lrcorner \wedge \dots \wedge x_n{:}\llcorner u_n\lrcorner \wedge c \wedge d\sigma]\!]$$

if there exists $\llcorner u\lrcorner := f(\llcorner u_1\lrcorner,\dots,\llcorner u_n\lrcorner)\,[\![c]\!] \in \Delta$ such that $f(u_1,\dots,u_n) = u\sigma$ (we assume that the variables of $u_1,\dots,u_n$ and c do not occur in the constrained term $t[x]\,[\![x{:}\llcorner u\lrcorner \wedge d]\!]$) and $x_1,\dots,x_n$ are fresh variables. The variable x, constrained to be in the language defined by the non-terminal $\llcorner u\lrcorner$, is replaced by $f(x_1,\dots,x_n)$ where $x_1,\dots,x_n$ are constrained to the respective languages of $\llcorner u_1\lrcorner,\dots,\llcorner u_n\lrcorner$. The union of the relations $\vdash^x_{\mathcal{G}}$ for all x is denoted $\vdash_{\mathcal{G}}$ and the reflexive transitive and transitive closures of the relation $\vdash_{\mathcal{G}}$ are respectively denoted by $\vdash^*_{\mathcal{G}}$ and $\vdash^+_{\mathcal{G}}$.

The language $L\big(\mathcal{G},\llcorner u\lrcorner\big)$ is the set of ground terms t *generated* by a constrained tree grammar $\mathcal{G}$ starting with the non-terminal $\llcorner u\lrcorner$, *i.e.* such that $x\,[\![x{:}\llcorner u\lrcorner]\!] \vdash^*_{\mathcal{G}} t\,[\![c]\!]$ where c is satisfiable. The above membership constraints $t{:}\llcorner u\lrcorner$, with $\llcorner u\lrcorner \in Q$, are interpreted by: $sol(t{:}\llcorner u\lrcorner) = \{\sigma \mid t\sigma \in L(\mathcal{G},\llcorner u\lrcorner)\}$. Note that we shall use below such membership constraints in order to restrict a term to a given sort or a given regular tree language.

3 Verification of a Key Distribution Protocol

In this section, we describe in an example how to specify a protocol, its environment and security properties with conditional constrained rewrite rules, and how the implicit induction procedure of Section 5 can be applied to the verification of the security properties, expressed as a joinable inductive conjectures.

Signature. Assume some sorts Nat, Bool, Name, Id, Key, Msg, MsgList, with the subsort relations: Name $\subseteq$ Msg and Key $\subseteq$ Msg. The messages exchanged during the protocol execution are abstracted by well sorted terms built with constructor symbols $\mathsf{pair} : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$, and projections $\mathsf{fst}, \mathsf{snd} : \mathsf{Msg} \to \mathsf{Msg}$, encryption and decryption in symmetric and asymmetric key cryptography enc, aenc, dec, adec, all with profile $\mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ and which follow the rules in Figure 1. The variables x represents the encrypted plaintext and the y is a symmetric or a public encryption key. The idempotent operator $\mathsf{inv} : \mathsf{Key} \to \mathsf{Key}$ associates to a public key its corresponding private key (for decryption), and conversely; We assume moreover an operator $\mathsf{pub} : \mathsf{Name} \to \mathsf{Key}$ which associates to the identity of a user its public key. The symbol inv is called *secret* and all the others symbols are called *public.*

Let us also consider a public constructor $\mathsf{sent} : \mathsf{Id} \times \mathsf{Name} \times \mathsf{Name} \times \mathsf{Msg} \to \mathsf{Msg}$ used to encapsulate messages with a header. Its first argument is a message identifier, the second and third arguments are respectively the names of sender and receiver of the message and the last argument is the message itself. The public constructor symbol $\mathsf{body} : \mathsf{Msg} \to \mathsf{Msg}$ can be used for removing the header, with the rule: $\mathsf{body}\big(\mathsf{sent}(x_i, x_a, x_b, x)\big) \to x$.

We assume moreover some additional secret constructors for Boolean: $\mathsf{true}, \mathsf{false} : \mathsf{Bool}$, for natural numbers $0 : \mathsf{Nat}$, $s : \mathsf{Nat} \to \mathsf{Nat}$, for lists of messages, $\mathsf{nil} : \mathsf{MsgList}$, $:: : \mathsf{Msg} \times \mathsf{MsgList} \to \mathsf{MsgList}$ and constant values used in the protocol messages: $K : \mathsf{Key}$, $S : \mathsf{Msg}$. Finally, we assume that the set of names of honest users (*i.e.* the set of terms of sort Name) is a (possibly infinite) regular tree set[4] whose terms are made only of public constructor symbols.

Let us denote $\mathcal{R}_\mathcal{C}$ the set of rewrite rules given above, which are sometimes referred as *explicit destructors* rules in the protocol verification literature. We propose in Appendix A a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ which generates the constructor normal forms. It contains in particular the non terminals $\llcorner y_\lrcorner^{\mathsf{List}}$ and $\llcorner x_\lrcorner^{\mathsf{Nat}}$ generating normal forms of respective sorts List and Nat.

Protocol. We consider a simplification (without certificates and timestamps) of a key distribution protocol of Denning & Sacco [13] for a symmetric key exchange in an asymmetric cryptosystem. Following the approach of [15], we consider traces of messages modelled as lists (built with nil and $::$) and characterized by the defined symbol $\mathsf{trace} : \mathsf{Int} \times \mathsf{MsgList} \to \mathsf{Msg}$. In $\mathsf{trace}(n, \ell)$, ℓ is a list of messages exchanged (protocol trace) and n can be seen as a resource consumed by each operation executed by an honest or dishonest agent. As we shall see below (in the description of the conjectures proved), the principle of our verification method is to perform an induction on the initial value of this resource (at the beginning of the protocol).

The symbol trace is defined recursively by extension with messages sent by the users participating to the protocol (honest or not). In the case of the Denning & Sacco protocol, the conditional rule (DS-A) of $\mathcal{R}_\mathcal{D}$ describes the user x_a sending to user x_b a message with identifier 1, which contains a freshly chosen symmetric key K for further secure communications:

$$\mathsf{trace}(s(n), y) \to \mathsf{trace}\big(n, \mathsf{sent}(1, x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(K, \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b)))) :: y\big) \; [\![x_a : \mathsf{Name}, x_b : \mathsf{Name}, x_a \not\approx x_b]\!] \quad (\mathsf{DS\text{-}A})$$

This key K is encrypted, for authentication purpose, using the asymmetric encryption function aenc and the secret key of x_a, represented as the inverse $\mathsf{inv}(\mathsf{pub}(x_a))$ of its public key $\mathsf{pub}(x_a)$. The result of this encryption is then encrypted with x_b's public key $\mathsf{pub}(x_b)$ so that only x_b shall be able to learn K. Moreover, x_a appends its name at the beginning of the message (using pair) so that the receiver x_b knows which public key to use in order to recover K.

[4] We will not define explicitly a tree grammar for Name here, we just assume that it contains the constants A, B and I.

In the second conditional rule (DS-B) of $\mathcal{R}_\mathcal{D}$, the honest user x_b, while reading a message x_m, expects that x_m has the above form (though he does not check this) and extracts the symmetric key K, applying twice the asymmetric decryption function adec to the second component of x_m, obtained by application of the projection function snd. This key K is then used by x_b to encrypt (with the function enc) a secret code S that he wants to communicate to the user x_a, and this ciphertext is sent in a message with identifier 2.

$$\begin{aligned}\mathsf{sent}(1, x'_a, x_b, x_m) \in y = \mathsf{true} \Rightarrow \mathsf{trace}\big(s(n), y\big) \to \mathsf{trace}(n, \mathsf{sent}(2, x_b, \mathsf{fst}(x_m)),\\ \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x_m), \mathsf{inv}(\mathsf{pub}(x_b))), \mathsf{pub}(\mathsf{fst}(x_m))))) :: y\big) \quad (\mathsf{DS\text{-}B})\end{aligned}$$

Attacker. We assume asynchronous communication of the messages through an insecure public network controlled by a dishonest user called *attacker*. The attacker is able to read and analyse any message sent to the network and to resend new messages composed from the information collected. Both the extraction of information from the messages read and the composition of new messages are modeled by the application of public constructor symbols and the reduction using the rules of $\mathcal{R}_\mathcal{C}$. This makes the framework with explicit destructors more uniform that others (often called "Dolev-Yao" models) where information extraction is modeled with extra ad-hoc inference rules. Note that we do not need here the extra predicates analyze and synthesis of [15] for the specification of trace. Besides uniformity, the addition of explicit destructor rules makes the model strictly more expressive, in the sense that it captures strictly more attacks, like for instance the attack described below. The operations of the attacker are specified by the following rules of $\mathcal{R}_\mathcal{D}$ for the specification of trace:

$$\mathsf{trace}\big(s(n), y\big) \to \mathsf{trace}(n, x :: y)\ [\![x : \mathsf{Init}]\!] \qquad (\mathsf{att\text{-}init})$$

$$x_1 \in y, \dots, x_k \in y \Rightarrow \mathsf{trace}\big(s(n), y\big) \to \mathsf{trace}\big(n, f(x_1, \dots, x_k) :: y\big) \qquad (\mathsf{att\text{-}anlz})$$

In the rule (att-init), Init is an extra non-terminal of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ which generates a regular tree language representing the initial knowledge of the attacker. In this example, we assume that this language contains all the terms of sort Name and Id. In the rule (att-anlz), $x_i \in y$ has to be read as $x_i \in y = \mathsf{true}$. This rule actually represents one conditional rule for each public constructor symbol f of arity k.

The function $\in$: Msg $\times$ MsgList $\to$ Bool is defined by the three rules of $\mathcal{R}_\mathcal{D}$:

$$x \in \mathsf{nil} \to \mathsf{false},\ \ x_1 \in x_2 :: y \to \mathsf{true}\ [\![x_1 \approx x_2]\!],\ \ x_1 \in x_2 :: y \to x_1 \in y\ [\![x_1 \not\approx x_2]\!]$$

Note that the set of the above rules form a CCTRS sufficiently complete and terminating, but not ground-confluent.

Security properties. We construct a tree grammar $\mathcal{G}$ by intersection of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and a regular tree grammar $\mathcal{B}ad$ which generates bad traces. The grammar $\mathcal{G}$ is built with a product construction and for the sake of readability, every non-terminal of $\mathcal{G}$ will be denoted below $N_1 \cap N_2$ where N_1 is a non-terminal of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and N_2 is a non-terminal of $\mathcal{B}ad$. We assume two particular non-terminal in the grammar $\mathcal{B}ad$:

– a non-terminal $B^{\mathsf{DS}}_{\mathsf{auth}}$ which generates the set of lists (built with nil, $::$ and the other constructor symbols) containing a message of the form $\mathsf{sent}(2, B, A, \ldots)$ not preceded by a message of the form $\mathsf{sent}(1, A, B, \ldots)$. Such a list corresponds to an *authentication* flaw. Note that this set is a regular tree language.
– a non-terminal B_{sec} which generates the set of lists containing the constant S. Such a list corresponds to a *secrecy* flaw: it indicates that the secret value S is publicly revealed.

The two conjectures (C_{auth}), (C_{sec}) express that every bad trace y cannot be a trace of the protocol obtained in n steps, for any n:

$$\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)\ [\![y : \llcorner y^{\mathsf{List}}_{\lrcorner} \cap B^{\mathsf{DS}}_{\mathsf{auth}}, n : \llcorner x^{\mathsf{Nat}}_{\lrcorner}]\!] \qquad (C_{\mathsf{auth}})$$

$$\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)\ [\![y : \llcorner y^{\mathsf{List}}_{\lrcorner} \cap B_{\mathsf{sec}}, n : \llcorner x^{\mathsf{Nat}}_{\lrcorner}]\!] \qquad (C_{\mathsf{sec}})$$

More precisely, (C_{auth}) expresses that no authentication flaw (man-in-the-middle attack) occurs during protocol executions, and (C_{sec}) expresses that the constant S remains secret to the attacker. The negation $\mathsf{trace}(n, \mathsf{nil}) \neq \mathsf{trace}(0, y)$ should be understood as $\mathsf{trace}(n, \mathsf{nil}) = \mathsf{trace}(0, y) \Rightarrow \mathsf{true} = \mathsf{false}$. Note that the above variables y and n are constrained to be instantiated by terms generated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ starting respectively with the non-terminals $\llcorner y^{\mathsf{List}}_{\lrcorner}$ and $\llcorner x^{\mathsf{Nat}}_{\lrcorner}$, and y is moreover constrained to be instantiated by terms generated by $\mathcal{B}ad$.

Note that (C_{auth}) and (C_{sec}) have the same form, only the regular tree grammar $\mathcal{B}ad$ differs. In general, with this approach, we can express that there is no intersection between protocol traces and bad traces for any regular set of bad traces, which makes quite a rich language of security properties.

Disproofs. The application of our procedure shows that none of the conjectures is a joinable inductive theorems of $\mathcal{R}$, by induction on traces, revealing attacks on the protocol.
Among the instances of the conjecture (C_{auth}) generated[5] by application of the production rules of $\mathcal{G}$, we have the instance where the variable y is replaced by nil and n is replaced by $s^8(0)$, denoted 8 below. We can show that this instance is a counterexample for the conjecture (C_{auth}), with the normalization with $\mathcal{R}$ presented in Figure 2, where $m_I = \mathsf{pair}(A, \mathsf{aenc}(A, \mathsf{pub}(B))))$ and $m_B = \mathsf{sent}(2, B, A, \mathsf{enc}(S, \mathsf{adec}(A, \mathsf{pub}(A))))$ (A, B and I are arbitrary distinct constructor terms of sort Name). The normalization of Figure 2 indicates quite legibly an authentication attack on the protocol. In the first steps of the reduction, the attacker builds a message $\mathsf{sent}(1, I, B, m_I)$ using its initial knowledge (generated by $\mathcal{G}$ from the no-terminal Init), with rule ($\mathsf{att\text{-}init}$) and some public constructor symbols, with rule ($\mathsf{att\text{-}anlz}$). Then B reads this message and believes that it originated from A. He therefore sends to A an answer which is reduced by $\mathcal{R}_{\mathcal{C}}$ into: $\mathsf{sent}(2, B, A, m_B)$. Hence, the list obtained after the reduction in Figure 2 belongs to $L(\mathcal{G}, B^{\mathsf{DS}}_{\mathsf{auth}})$, since this message $\mathsf{sent}(2, B, A, m_B)$ is not preceded in the list by a message of the form $\mathsf{sent}(1, A, B, \ldots)$ (this indicates the

[5] The procedure generates all the instances which are smaller than $d(\mathcal{R})$ (the maximum depth of the left-hand sides of rules of $\mathcal{R}$), see Section 5.

$$
\begin{aligned}
&\mathsf{trace}\big(8, \mathsf{nil}\big) \xrightarrow[\text{(att-init)}]{*} \mathsf{trace}\big(5, I :: A :: B :: 1 :: \mathsf{nil}\big) \xrightarrow[\text{(att-anlz)}]{*} \mathsf{trace}\big(2, m_I :: I :: A :: B :: 1 :: \mathsf{nil}\big) \\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(1, \underbrace{\mathsf{sent}(1, I, B, m_I) :: m_I :: I :: A :: B :: 1 :: \mathsf{nil}}_{\ell}\big) \xrightarrow[\text{(DS-B)}]{} \\
&\mathsf{trace}\big(0, \mathsf{sent}(2, B, \mathsf{fst}(m_I), \mathsf{enc}(S, \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(m_I), \mathsf{inv}(\mathsf{pub}(B))), \mathsf{pub}(\mathsf{fst}(m_I))))) :: \ell\big) \\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{*} \mathsf{trace}\big(0, \mathsf{sent}(2, B, A, m_B) :: \ell\big)
\end{aligned}
$$

Figure 2: An authentication attack on DS protocol

$$
\begin{aligned}
&\mathsf{trace}(12, \mathsf{nil}) \xrightarrow{*} \mathsf{trace}\big(4, \mathsf{sent}(2, B, A, m_B) :: \ell\big) \\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(3, \mathsf{body}(\mathsf{sent}(2, B, A, m_B)) :: \mathsf{sent}(2, B, A, m_B) :: \ell\big) \\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(3, m_B :: \mathsf{sent}(2, B, A, m_B) :: \ell\big) \ \text{ let } \ell' = m_B :: \mathsf{sent}(2, B, A, m_B) :: \ell \\
&\xrightarrow[\text{(att-anlz)}]{*} \mathsf{trace}\big(1, \mathsf{adec}(A, \mathsf{pub}(A)) :: \ell'\big) \\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(0, \mathsf{dec}(m_B, \mathsf{adec}(A, \mathsf{pub}(A))) :: \mathsf{adec}(A, \mathsf{pub}(A)) :: \ell'\big) \\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{*} \mathsf{trace}\big(0, S :: \mathsf{dec}(m_B, \mathsf{adec}(A, \mathsf{pub}(A))) :: \mathsf{adec}(A, \mathsf{pub}(A)) :: \ell'\big) = \mathsf{trace}(0, \ell'')
\end{aligned}
$$

Figure 3: An attack on the secrecy of S for DS protocol

authentication flaw). It means that the instance of conjecture (C_{auth}) is reduced to a clause of the form $\mathsf{sent}(2, B, A, m_B) :: \ell \neq \mathsf{sent}(2, B, A, m_B) :: \ell$, which leads to a case of **disproof**.

For Conjecture (C_{sec}), we consider now $n = s^{12}(0)$ and the reduction in Figure 3. The first steps of this figure are the same as in Figure 2. In the next steps, the attacker builds (with rules (att-init) and (att-anlz)) a fake key $\mathsf{adec}(A, \mathsf{pub}(A))$ which he uses latter in order to decipher the message m_B from B and recover S. Hence, the lists obtained in Figure 3 belongs to $L(\mathcal{G}, B_{\mathsf{sec}})$. It means that the instance of conjecture (C_{sec}) are reduced to a clause $\mathsf{trace}(0, \ell'') \neq \mathsf{trace}(0, \ell'')$, which leads to a case of **disproof**.

Shamir-Rivest-Adleman Three Pass Protocol. Another example of derivation of an attack is proposed in [6] (see also Appendix B and Figure 6). We won't reproduce it in details here. We would just like to outline one interesting use of constraints in explicit destructor axioms in this example. Indeed, the protocol RSA 3-pass relies on a commutativity-like property for the encryption operator. In our model, it is expressed by the following rule of $\mathcal{R}_\mathcal{C}$:

$$\mathsf{aenc}(\mathsf{aenc}(x, k_1), k_2) = \mathsf{aenc}(\mathsf{aenc}(x, k_2), k_1)\ [\![k_1 > k_2]\!] \qquad (1)$$

Note the addition of the ordering constraint, for termination purposes.

4 Towards Joinable Inductive Validation of Protocols

Let us modify the protocol rules of Section 3 in order to fix the above attacks. We add a $\mathsf{pair}(x_a, x_b)$ along with the key K in the first message:

$$\mathsf{trace}(s(n), y) \to \mathsf{trace}\big(n, \mathsf{sent}(1, x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(\mathsf{aenc}(\mathsf{pair}(\mathsf{pair}(x_a, x_b), K), \mathsf{inv}(\mathsf{pub}(x_a))), \mathsf{pub}(x_b)))) :: y\big) \; [\![x_a : \mathsf{Name}, x_b : \mathsf{Name}, x_a \neq x_b]\!] \quad (\mathsf{DS\text{-}A'})$$

Before sending the second message, x_b checks first the pair $\mathsf{pair}(x_a, x_b)$ sent in the ciphertext (we let $k = \mathsf{adec}(\mathsf{adec}(\mathsf{snd}(x_m), \mathsf{inv}(\mathsf{pub}(x_b))), \mathsf{pub}(\mathsf{fst}(x_m)))$):

$$\mathsf{sent}(1, x'_a, x_b, x_m) \in y = \mathsf{true}, \mathsf{snd}(\mathsf{fst}(k)) = x_b, \mathsf{fst}(\mathsf{fst}(k)) = \mathsf{fst}(x_m) \Rightarrow \mathsf{trace}(s(n), y) \to \mathsf{trace}(n, \mathsf{sent}(2, x_b, \mathsf{fst}(x_m), \mathsf{enc}(S, k))) :: y) \quad (\mathsf{DS\text{-}B'})$$

We present below some parts of the validation of the amended version of the protocol with our procedure, i.e. the proof that the conjecture C_{sec} is a joinable inductive theorem of the above specification. The proof is much more difficult than in the previous sections. Indeed, we need here to verify all the execution traces in order to validate the protocol (by definition of *joinable* inductive theorems), since $\mathcal{R}$ is not ground-confluent. In comparison, it is sufficient to find one erroneous trace in order to show that the protocol is flawed.

The application of the procedure generates several subgoals, amongst them:

$$y \neq \mathsf{nil} \, [\![y : \llcorner y \lrcorner^{\mathsf{List}} \cap B\mathsf{sec}]\!]$$
$$y \neq \mathsf{trace}(n, x :: \mathsf{nil}) \, [\![y : \llcorner y \lrcorner^{\mathsf{List}} \cap B_{\mathsf{sec}}, n : \llcorner x \lrcorner^{\mathsf{Nat}}, x : \mathsf{Init}]\!]$$
$$x_1 \in y = \mathsf{true}, \ldots, x_k \in y = \mathsf{true} \Rightarrow y \neq \mathsf{trace}(n, f(x_1, \ldots, x_k) :: \mathsf{nil}) \; [\![y : \llcorner y \lrcorner^{\mathsf{List}} \cap B^{\mathrm{DS}}_{\mathsf{sec}}, n : \llcorner x \lrcorner^{Nat}]\!]$$

Let us recall that Init is a non-terminal of a regular tree grammar generating the language of the initial knowledge of the attacker. This language contains all the ground constructor terms of sort Name and Id in our example. In the third subgoal, f denotes any public constructor symbol of arity k. In our example, $k = 1$ and f is pub, fst, snd, body or $k = 2$ and and f is pair, enc, dec, aenc, adec.

The proof of the first subgoal is immediate, but the other subgoals need more developments and the interactive addition of some lemmas in order to derive a proof. We are working on an extension of our inference system with new simplification rules in order to avoid the divergence during the validation of correct authentication protocols.

5 Implicit Inductive Theorem Proving procedure

We present in this section a goal-directed inductive theorem proving procedure for conditional and constrained specifications.

This procedure belongs to the family of *implicit induction* (in the lines of [7]) and combines the power of two classical methods for automatic induction: *explicit* *induction* and *proof by consistency* [11]. As outlined above, the procedure supports features which are generally not found in former inductive theorem proving approaches, like handling non ground-confluent rewrite systems, axioms between constructors (used here for specifying explicit destructors) or the parametrization of the specification and conjectures with given regular set of terms. A key for these characteristics is that the whole procedure is based on a constrained tree grammar, which is computed automatically from the given specification. It is used for several purposes: (i) as an *induction scheme*. Using a constrained tree grammar instead of a test-set like in the former procedures [7, 4] permits precisely to handle constrained rewrite rules between constructors, (ii) as an *oracle* for checking validity and redundancy at each induction steps, by reduction to an emptiness problem, (iii) in order to characterize *regular sets* of terms representing specific values or traces, see Section 3.

5.1 Constrained Tree Grammar for Induction

Constrained tree grammars permit an exact representation of the set of ground constructor terms irreducible by a given CTRS. For this reason, such formalisms have been studied in many works related to inductive theorem proving, see *e.g.* [11]. Indeed, under some assumptions like sufficient completeness and termination for constructor axioms, they provide a finite description of the minimal Herbrand model (a set of representatives of the minimal Herbrand model is the language of ground constructor $\mathcal{R}_\mathcal{C}$-normal forms in this case).

For every constructor CTRS $\mathcal{R}_\mathcal{C}$, we can construct a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ which generates the language of ground $\mathcal{R}_\mathcal{C}$-normal forms. This construction, presented in [5] and exemplified in the Section 3, intuitively corresponds to the complementation and completion of a tree grammar for $\mathcal{R}_\mathcal{C}$-reducible terms, where every subset of non-terminals (for the complementation) is represented by the most general unifier of its elements. In a first step of our induction procedure, we construct a constrained tree grammar $\mathcal{G} = (Q, \Delta)$. This grammar is assumed fixed in the rest of the section. For the construction of $\mathcal{G}$, we start with $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and possibly make an intersection with one or several regular tree grammars, (see Section 3). The intersection between a constrained tree grammar and a regular tree grammar is a constrained tree grammar.

We call a constrained term $t\,[\![c]\!]$ *decorated* if $c = x_1 \colon \llcorner u_1 \lrcorner \wedge \ldots \wedge x_n \colon \llcorner u_n \lrcorner \wedge d$, $\{x_1, \ldots, x_n\} = \mathit{var}(t)$, $\llcorner u_i \lrcorner \in Q$ and $\mathit{sort}(u_i) = \mathit{sort}(x_i)$ for all $i \in [1..n]$.

Some of the following inference rules invoke tests for (a) satisfiability of constraints in clauses, (b) ground irreducibility of constructor clauses and (c) joinable inductive validity of ground irreducible of constructor clauses. It is shown in [5] that the properties (a) and (b) are reducible to emptiness decision for constrained tree grammars which slightly extend $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. A similar reduction is also possible for (c), following the idea that when a clause is ground irreducible, testing joinable inductive validity amounts, by definition, at testing syntactic equality. In other terms, when a and b are ground irreducible $\mathcal{R} \models_{jind} a = b\,[\![c]\!]$

iff the constraint $c \wedge a \not\approx b$ is unsatisfiable, and (c) is reducible to (a), hence to emptiness decision.

Based on former decision results for tree automata with equality and disequality constraints [9], some restrictions on $\mathcal{R}_\mathcal{C}$ are given [5] which ensure the decidability of the three above problem. The rewrite systems described in Sections 3 and 4 fulfill these restrictions.

5.2 Simplification Rules

We present in Figure 4 the system $\mathcal{S}$ of simplification rules for constructor clauses. Rewriting simplifies goals with axioms. Since $\mathcal{R}$ may not be ground-confluent, we consider all the (one step) reductions with $\mathcal{R}$. Rewrite Splitting simplifies a constrained clause which contains a subterm matching some left member of rule of $\mathcal{R}_\mathcal{D}$. The inference checks moreover that all cases are covered for the application of such rules of $\mathcal{R}_\mathcal{D}$, *i.e.* that for each ground substitution τ, the conditions and the constraints of at least one rule is true wrt τ. Partial Splitting eliminates ground reducible terms in a constrained clause $C\,[\![c]\!]$ by adding to $C\,[\![c]\!]$ the negation of constraint of some rules of $\mathcal{R}_\mathcal{C}$. Therefore, the saturated application of Partial splitting and Rewriting will always lead to Deletion or to ground irreducible constructor clauses. Finally, Deletion and Validity remove respectively tautologies and clauses with unsatisfiable constraints, and ground irreducible constructor joinable inductive theorems of $\mathcal{R}$. As explained in Section 5.1, the tests in the rules Deletion and Validity are discharged to a decision procedure for the emptiness of constrained tree grammars.

Rewriting $\quad C\,[\![c]\!] \vdash_\mathcal{S} \{D_1\,[\![c]\!], \dots, D_k\,[\![c]\!]\}$

if for all $i \leq k$, $D_i\,[\![c]\!] \ll C\,[\![c]\!]$ where $\{D_1\,[\![c]\!], \dots, D_k\,[\![c]\!]\}$ are all the clauses obtained by one-step rewriting with $\mathcal{R}$ from $C\,[\![c]\!]$.

Rewrite Splitting $\quad C\,[\![c]\!] \vdash_\mathcal{S} \left\{\Gamma_i\sigma_i \Rightarrow C[r_i\sigma_i]_{p_i}\,[\![c \wedge c_i\sigma_i]\!] \;\middle|\; p_i \text{ pos. of } C\right\}_{i\in[1..n]}$

if $\mathcal{R} \models_{jind} \Gamma_1\sigma_1\,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c \wedge c_n\sigma_n]\!]$, $C|_{p_i} > r_i\sigma_i$ and $\{C|_{p_i}\} >^{mul} \Gamma_i\sigma_i$
where the $\Gamma_i\sigma_i \Rightarrow l_i\sigma_i \to r_i\sigma_i\,[\![c_i\sigma_i]\!]$, $i \in [1..n]$, are all the instances of rules $\Gamma_i \Rightarrow l_i \to r_i\,[\![c_i]\!] \in \mathcal{R}_\mathcal{D}$ such that $l_i\sigma_i = C|_{p_i}$

Partial Splitting $\quad C[l\sigma]_p\,[\![c]\!] \vdash_\mathcal{S} \left\{C[r\sigma]_p\,[\![c \wedge c'\sigma]\!], C[l\sigma]_p\,[\![c \wedge \neg c'\sigma]\!]\right\}$

if $l \to r\,[\![c']\!] \in \mathcal{R}_\mathcal{C}$, $l\sigma > r\sigma$, and neither $c'\sigma$ nor $\neg c'\sigma$ is a subformula of c where $C\,[\![c]\!]$ is a constructor clause.

Deletion $\quad C\,[\![c]\!] \vdash_\mathcal{S} \emptyset$

if $C\,[\![c]\!]$ is a tautology or c is unsatisfiable.

Validity $\quad C\,[\![c]\!] \vdash_\mathcal{S} \emptyset$

if $C\,[\![c]\!]$ is a ground irreducible constructor clause and $\mathcal{R} \models_{jind} C\,[\![c]\!]$.

Figure 4: System $\mathcal{S}$: simplification rules

Simplification $\dfrac{(\mathcal{E} \cup \{C \llbracket c \rrbracket\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$

if $\mathcal{I}ndvar(C \llbracket c \rrbracket) = \emptyset$ and $C \llbracket c \rrbracket \vdash_{\mathcal{S}} \mathcal{E}'$

Inductive Narrowing $\dfrac{(\mathcal{E} \cup \{C \llbracket c \rrbracket\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \{C \llbracket c \rrbracket\})}$

if for all i in $[1..n]$, $d(C_i) - d(C) \leq d(\mathcal{R}) - 1$ and $C_i \llbracket c_i \rrbracket \vdash_{\mathcal{S}} \mathcal{E}_i$

where $\{C_1 \llbracket c_1 \rrbracket, \ldots, C_n \llbracket c_n \rrbracket\}$ is the set of clauses s. t. $C \llbracket c \rrbracket \vdash^{+}_{\mathcal{G}} C_i \llbracket c_i \rrbracket$

Subsumption $\dfrac{(\mathcal{E} \cup \{C \llbracket c \rrbracket\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$

if $C \llbracket c \rrbracket$ is subsumed by another clause of $\mathcal{R} \cup \mathcal{E} \cup \mathcal{H}$

Disproof $\dfrac{(\mathcal{E} \cup \{C \llbracket c \rrbracket\}, \mathcal{H})}{(\mathsf{Disproof}, \mathcal{H})}$

if $C \llbracket c \rrbracket$ is a constructor clause and no other rule applies to $C \llbracket c \rrbracket$

Failure $\dfrac{(\mathcal{E} \cup \{C \llbracket c \rrbracket\}, \mathcal{H})}{(\mathsf{Failure}, \mathcal{H})}$

if $C \llbracket c \rrbracket$ is not a constructor clause and no other rule applies to the clause $C \llbracket c \rrbracket$

Figure 5: System $\mathcal{I}$: inference rules for joinable-induction

5.3 Main inference system

The main inference system $\mathcal{I}$ is displayed in Figure 5. Its rules apply to pairs $(\mathcal{E}, \mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures and $\mathcal{H}$ is the *set* of inductive hypotheses (constrained clauses). The inference rules of $\mathcal{I}$ use the constrained tree grammar $\mathcal{G}$ in order to instantiate variables. The replacements are limited to variables, called *induction variables*, whose instantiation is needed in order to trigger a rewrite step.

Definition 1. *The set $\mathcal{I}ndpos(f, \mathcal{R})$ of* induction positions *of $f \in \mathcal{D}$ is the set of non-root and non-variable positions of left-hand sides of rules of $\mathcal{R}_{\mathcal{D}}$ with the symbol f at the root position. The set $\mathcal{I}ndvar(t)$ of* induction variables *of $t = f(t_1, \ldots, t_n)$, with $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, is the subset of variables of $var(t)$ occurring in t at positions of $\mathcal{I}ndpos(f, \mathcal{R})$.*

Let us now describe the inference rules of $\mathcal{I}$. Simplification reduces a conjecture which does not contain any induction variable using the rules of System $\mathcal{S}$ (Figure 4). Inductive Narrowing generates new subgoals by application of the production rules of the constrained grammar $\mathcal{G}$ until the obtained clause is deep enough to cover left hand side of rules of $\mathcal{R}_{\mathcal{D}}$. Each obtained clause must be simplified by one of the rules of $\mathcal{S}$ (if one instance cannot be simplified, then the rule Inductive Narrowing cannot be applied). Subsumption deletes clauses redundant with axioms of $\mathcal{R}$, induction hypotheses of $\mathcal{H}$ and other conjectures not yet proved (in $\mathcal{E}$).

5.4 Soundness and Completeness

Our inference system is sound, and refutationally complete.

Definition 2. *We call* derivation *a sequence of inference steps generated by a pair of the form* $(\mathcal{E}_0, \emptyset)$*, using the inference rules in* $\mathcal{I}$*, written* $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots (\mathcal{E}_n, \mathcal{H}_n) \vdash_{\mathcal{I}} \cdots$. *We say that a derivation is* fair *if the set of persistent constrained clauses* $(\cup_i \cap_{j \geq i} \mathcal{E}_j)$ *is empty or equal to* $\mathsf{Disproof}$ *or* $\mathsf{Failure}$*. The derivation is said to be a* disproof *or* failure*, respectively, in the two last cases, and a* success *in the first case.*

Finite success is obtained when the set of conjectures to be proved is exhausted. Infinite success is obtained when the procedure diverges, assuming fairness. When it happens, the clue is to guess some lemmas which are used to subsume or simplify the generated infinite family of subgoals, therefore stopping the divergence. This is possible in our approach, since lemmas can be used in the same way as axioms are. The proof of the following theorems can be found in the long version [6] of this extended abstract.

Theorem 1 (Soundness of successful derivations). *Let* $\mathcal{E}_0$ *be a set of decorated constrained clauses. If there exists a successful derivation* $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ *then* $\mathcal{R} \models_{jind} \mathcal{E}_0$.

The following theorem states that the derivation of $\mathsf{Disproof}$ by our inference system is a correct refutation of the conjecture.

Theorem 2 (Soundness of disproof). *If a derivation starting from* $(\mathcal{E}_0, \emptyset)$ *returns the pair* $(\mathsf{Disproof}, \mathcal{H})$*, then* $\mathcal{R} \not\models_{jind} \mathcal{E}_0$.

The derivation of $\mathsf{Failure}$ means that we cannot conclude, however, this never happens providing the property of strongly completeness for $\mathcal{R}$. A function symbol $f \in \mathcal{D}$ is *sufficiently complete* wrt $\mathcal{R}$ iff for all $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$, there exists t in $\mathcal{T}(\mathcal{C})$ such that $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{R}]{+} t$. We say that the system $\mathcal{R}$ is sufficiently complete iff every defined operator $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$.

Definition 3. *Let* $f \in \mathcal{D}$ *and let:* $\big\{\Gamma_1 \Rightarrow f(t_1^1, \ldots, t_k^1) \rightarrow r_1 \llbracket c_1 \rrbracket, \ldots, \Gamma_n \Rightarrow f(t_1^n, \ldots, t_k^n) \rightarrow r_n \llbracket c_n \rrbracket\big\}$ *be a maximal subset of rules of* $\mathcal{R}_{\mathcal{D}}$ *whose left-hand sides are identical up to variable renaming* $\mu_1, \ldots, \mu_n$ i.e. $f(t_1^1, \ldots, t_k^1)\mu_1 = \ldots = f(t_1^n, \ldots, t_k^n)\mu_n$. *We say that* f *is* strongly complete *wrt* $\mathcal{R}$ *(see [4]) if* f *is sufficiently complete wrt* $\mathcal{R}$ *and* $\mathcal{R} \models_{jind} \Gamma_1\mu_1 \llbracket c_1\mu_1 \rrbracket \vee \ldots \vee \Gamma_n\mu_n \llbracket c_n\mu_n \rrbracket$ *for every subset of* $\mathcal{R}$ *as above. The system* $\mathcal{R}$ *is said strongly complete if every* $f \in \mathcal{D}$ *is strongly complete wrt* $\mathcal{R}$.

Theorem 3 (Refutational completeness). *Assume that* $\mathcal{R}$ *is strongly complete and let* $\mathcal{E}_0$ *be a set of decorated constrained clauses. If* $\mathcal{R} \not\models_{jind} \mathcal{E}_0$*, then all fair derivations starting from* $(\mathcal{E}_0, \emptyset)$ *end up with* $(\mathsf{Disproof}, \mathcal{H})$.

Conclusion

We have developed a procedure for proving joinable inductive theorems of conditional and constrained constructor based specifications which may be non confluent. This procedure is shown correct and refutationally complete, and has been applied to the verification of security properties of cryptographic protocols, both for the research of attacks or protocol validation.

A closely related first order model, also based on trace, was proposed in [10]. This exact model was defined in order to prove a theoretical result on the minimal number of user names required in order to prove security properties. To our knowledge, this model has not been applied in practice for protocol verification.

We are planing several development of this method for protocol verification. First, a natural case study for induction are group protocols, see *e.g.* [8], with some induction on the number of participants.

Several procedures permit automatic validation of protocols described by first order specifications, e.g. [2], but they generally rely on over-approximating models, and are not suitable for the research of attacks, as they generate false positives. Trace based models, like the one presented here, are not approximated and hence appropriate for the search of attack, but automatic protocol validation in such models is considered as a difficult problem. The main difficulty is to generate invariants about the set of data that the attacker can not deduce. Systems like Securify [12] or Hermes [3] are based on some fixed generic invariants. We would like to study the problem of the automatic generation of appropriate ad-hoc invariants, based on the theoretical framework proposed in this paper.

Acknowledgments. The authors wish to thank Hubert Comon-Lundh for the fruitful discussions they have had about this method and the reviewers for their useful remarks.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, 2001.
2. B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols In IEEE Symp. on Security and Privacy, pages 86-100, 2004.
3. L. Bozga, Y.'Lakhnech, M. Périn. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In Proc. of the 15th Computer-Aided Verification conf. (CAV'03), vol. 2725 of Spinger LNCS, 2003.
4. A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.
5. A. Bouhoula and F. Jacquemard. Automated induction for complex data structures. Research Report LSV-05-11, Laboratoire Spécification et Vérification, 2005.
6. A. Bouhoula and F. Jacquemard. Tree Automata, Implicit Induction and Explicit Destructors for Security Protocol Verification. Research Report LSV-07-10, 2007.
7. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 1995.

8. A. Bundy and G. Steel. Attacking group protocols by refuting incorrect inductive conjectures. *Journal of Automated Reasoning*, vol. 36, numbers 1-2, pages 149-176. January 2006.
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`, 2002.
10. H. Comon and V. Cortier. Security properties: two agents are sufficient. Science of Computer Programming 50(1-3), pages 51-71, Elsevier, 2004.
11. H. Comon-Lundh. *Handbook of Automated Reasoning*, chapter Inductionless Induction. Elsevier, 2001.
12. V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), pages 97-110. IEEE Comp. Soc. Press, 2001.
13. D. E. Denning and G. M. Sacco. Timestamps in Key Distribution Protocols. In *Communications of the ACM*, 1981.
14. C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryption. *Electr. Notes Theor. Comput. Sci.*, 125(1):43–54, 2005.
15. L. C. Paulson. The inductive approach to verifying cryptographic protocol. *Journal of Computer Security*, 6:85–128, 1998.

Appendix

A Normal form constrained tree grammar for Section 3

The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ for the generation of constructor normal forms contains the following sorted non terminals: $\llcorner \mathsf{pair}(x_1, x_2) \lrcorner$, $\llcorner \mathsf{enc}(x, y) \lrcorner$, $\llcorner \mathsf{aenc}(x, y) \lrcorner$, $\llcorner \mathsf{inv}(v) \lrcorner$, $\llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner$, $\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner$, Name, $\llcorner x \lrcorner^{\mathsf{Key}}$, $\llcorner x \lrcorner^{\mathsf{Msg}}$, $\llcorner x \lrcorner^{\mathsf{List}}$, $\llcorner x \lrcorner^{\mathsf{Bool}}$, $\llcorner x \lrcorner^{\mathsf{Nat}}$ and $\llcorner x \lrcorner^{\mathsf{red}}$. We assume that Name is the initial non-terminal of a regular tree grammar generating the constructor terms of sort Name. The constrained production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ are (M represents below any non-terminal of sort Msg):

$$\llcorner x \lrcorner^{\mathsf{Nat}} := 0 \mid s(\llcorner x \lrcorner^{\mathsf{Nat}}) \quad \llcorner x \lrcorner^{\mathsf{List}} := \mathsf{nil} \mid M :: \llcorner x \lrcorner^{List} \quad \llcorner x \lrcorner^{\mathsf{Key}} := K \mid \mathsf{pub}(\mathsf{Name})$$

$$\llcorner \mathsf{enc}(x, y) \lrcorner := \mathsf{enc}(M_1, M_2) \qquad \llcorner \mathsf{inv}(y) \lrcorner := \mathsf{inv}(\llcorner x \lrcorner^{\mathsf{Key}})$$

$$\llcorner \mathsf{aenc}(x, y) \lrcorner := \mathsf{aenc}(M_1, M_2) \quad \llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner := \mathsf{aenc}(M, \llcorner \mathsf{inv}(y) \lrcorner)$$

$$\llcorner \mathsf{pair}(x_1, x_2) \lrcorner := \mathsf{pair}(M_1, M_2) \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{fst}(\llcorner \mathsf{pair}(x_1, x_2) \lrcorner) \mid \mathsf{snd}(\llcorner \mathsf{pair}(x_1, x_2) \lrcorner)$$

$$\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner := \mathsf{sent}(\llcorner x \lrcorner^{\mathsf{Id}}, \llcorner x \lrcorner^{\mathsf{Name}}, \llcorner x \lrcorner^{\mathsf{Name}}, M) \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{body}(\llcorner \mathsf{sent}(x_i, x_a, x_b, x) \lrcorner)$$

$$\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{dec}(\llcorner \mathsf{enc}(x, y) \lrcorner, M)\, [\![y \not\approx M]\!] \quad \llcorner x \lrcorner^{\mathsf{red}} := \mathsf{dec}(\llcorner \mathsf{enc}(x, y) \lrcorner, M)\, [\![y \approx M]\!]$$

$$\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{adec}(\llcorner \mathsf{aenc}(x, y_1) \lrcorner, \llcorner \mathsf{inv}(y_2) \lrcorner)\, [\![y_1 \not\approx y_2]\!] \quad \llcorner x \lrcorner^{\mathsf{red}} := \ldots\, [\![y_1 \approx y_2]\!]$$

$$\llcorner x \lrcorner^{\mathsf{Msg}} := \mathsf{adec}(\llcorner \mathsf{aenc}(x, \mathsf{inv}(y)) \lrcorner, M)\, [\![y \not\approx M]\!] \qquad \llcorner x \lrcorner^{\mathsf{red}} := \ldots\, [\![y \approx M]\!]$$

The non terminal $\llcorner x \lrcorner^{\mathsf{red}}$ generates all $\mathcal{R}_\mathcal{C}$-reducible ground constructor terms, and the other n.t. generate all the ground constructor $\mathcal{R}_\mathcal{C}$-normal forms.

B Shamir-Rivest-Adleman Three Pass Protocol

We consider the same signature as in Section 3 and also the same constructor CTRS $\mathcal{R}_\mathcal{C}$ extended with the following commutativity-like rule for the encryption operator:

$$\mathsf{aenc}(\mathsf{aenc}(x, k_1), k_2) = \mathsf{aenc}(\mathsf{aenc}(x, k_2), k_1)\ [\![k_1 > k_2]\!] \qquad (2)$$

The protocol runs between two users x_a and x_b in 3 pass, which are described by the following 3 CCTRS rules of $\mathcal{R}_\mathcal{D}$.

$$\mathsf{trace}(s(n), y) = \mathsf{trace}\big(n, \mathsf{sent}(1, x_a, x_b, \mathsf{pair}(x_a, \mathsf{aenc}(S, \mathsf{key}(x_a)))) :: y\big)$$
$$[\![x_a, x_b : \mathsf{Name}, x_a \neq x_b]\!] \qquad (\mathsf{RSA\text{-}A1})$$

Initially, the honest user x_a sends to x_b a message containing a secret value S encrypted with its own public key $\mathsf{pub}(x_a)$. The ciphertext is sent in a pair, along with the identity of x_a.

$$\mathsf{sent}(x'_a, x_b, x) \in y = \mathsf{true} \Rightarrow$$
$$\mathsf{trace}(s(n), y) = \mathsf{trace}\big(n, \mathsf{sent}(2, x_b, \mathsf{fst}(x), \mathsf{aenc}(\mathsf{snd}(x), \mathsf{key}(x_b))) :: y\big) \qquad (\mathsf{RSA\text{-}B})$$

While reading a message x from x'_a, x_b is not able to decipher it and recover S, because he does not know the private key of A. Instead, he sends an answer obtained by encrypting again the message read with its public key $\mathsf{key}(x_b)$. The message sent by x_b has hence the form: $\mathsf{aenc}\big(\mathsf{aenc}(S, \mathsf{key}(x_a)), \mathsf{key}(x_b)\big)$ which, by the rule (2) of $\mathcal{R}_\mathcal{C}$ is equivalent to $\mathsf{aenc}\big(\mathsf{aenc}(S, \mathsf{key}(x_b)), \mathsf{key}(x_a)\big)$.

$$\mathsf{sent}(x_a, x_b, x) \in y = \mathsf{true}, \mathsf{sent}(x'_b, x_a, x) \in y = \mathsf{true} \Rightarrow$$
$$\mathsf{trace}(s(n), y) = \mathsf{trace}\big(n, \mathsf{sent}(3, x_a, x_b, \mathsf{adec}(x, \mathsf{inv}(\mathsf{key}(x_a)))) :: y\big) \qquad (\mathsf{RSA\text{-}A2})$$

After reading the message from x_b, x_a decrypts it with its private key $\mathsf{inv}(\mathsf{key}(x_a))$, and send the result $\mathsf{aenc}(S, \mathsf{key}(x_b))$. Then, B is able to decipher this message and recover S.

We consider the same rules of $\mathcal{R}_\mathcal{D}$ for $\in$, trace and for the attacker (att-init and att-anlz) as in Section 3. We assume moreover here that the language generated by the non-terminal Init (initial knowledge of the attacker) contains the private key $\mathsf{inv}(\mathsf{key}(I))$.

Like in Section 3, we construct the constrained tree grammar $\mathcal{G}$ by intersection of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ with a regular tree grammar $\mathcal{B}ad$. We consider again the non-terminal B_{sec} which generate the set of lists containing the constant S (like in Section 3) and another non-terminal $B^{\mathrm{RSA}}_{\mathsf{auth}}$ for authentication flaws, which generates the lists containing a message of the form $\mathsf{sent}(1, A, B, \ldots)$ followed by a message $\mathsf{sent}(3, A, B, \ldots)$, without a message of the form $\mathsf{sent}(2, B, A, \ldots)$

$$
\begin{aligned}
\mathsf{trace}(11, \mathsf{nil}) &\xrightarrow[\text{(RSA-A1)}]{} \mathsf{trace}\big(10, \mathsf{sent}(1, A, B, \mathsf{pair}(A, \mathsf{aenc}(S, \mathsf{key}(A)))) :: \mathsf{nil}\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(9, \mathsf{body}(\mathsf{sent}(1, A, B, \mathsf{pair}(A, \mathsf{aenc}(S, \mathsf{key}(A))))) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(9, \mathsf{pair}(A, \mathsf{aenc}(S, \mathsf{key}(A))) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(8, \mathsf{fst}(\mathsf{pair}(A, \mathsf{aenc}(S, \mathsf{key}(A)))) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(8, \mathsf{aenc}(S, \mathsf{key}(A)) :: \ldots\big)\\
&\xrightarrow[\text{(att-init)}]{} \mathsf{trace}\big(7, I :: \mathsf{aenc}(S, \mathsf{key}(A)) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(6, \mathsf{key}(I) :: I :: \mathsf{aenc}(S, \mathsf{key}(A)) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(5, \mathsf{aenc}(\mathsf{aenc}(S, \mathsf{key}(A)), \mathsf{key}(I)) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(5, \mathsf{aenc}(\mathsf{aenc}(S, \mathsf{key}(I)), \mathsf{key}(A)) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(4, \mathsf{sent}(2, I, A, \mathsf{aenc}(\mathsf{aenc}(S, \mathsf{key}(I)), \mathsf{key}(A))) :: \ldots\big)\\
&\xrightarrow[\text{(RSA-A2)}]{} \mathsf{trace}\big(3, \mathsf{sent}(3, A, B,\\
&\qquad \mathsf{adec}(\mathsf{aenc}(\mathsf{aenc}(S, \mathsf{key}(I)), \mathsf{key}(A)), \mathsf{inv}(\mathsf{key}(A)))) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(3, \mathsf{sent}(3, A, B, \mathsf{aenc}(S, \mathsf{key}(I))) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(2, \mathsf{body}(\mathsf{sent}(3, A, B, \mathsf{aenc}(S, \mathsf{key}(I)))) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(2, \mathsf{aenc}(S, \mathsf{key}(I)) :: \ldots\big)\\
&\xrightarrow[\text{(att-init)}]{} \mathsf{trace}\big(1, \mathsf{inv}(\mathsf{key}(I)) :: \mathsf{aenc}(S, \mathsf{key}(I)) :: \ldots\big)\\
&\xrightarrow[\text{(att-anlz)}]{} \mathsf{trace}\big(0, \mathsf{adec}(\mathsf{aenc}(S, \mathsf{key}(I)), \mathsf{inv}(\mathsf{key}(I))) :: \ldots\big)\\
&\xrightarrow[\mathcal{R}_\mathcal{C}]{} \mathsf{trace}\big(0, S :: \ldots\big)
\end{aligned}
$$

Figure 6: An attack on the secrecy of S for RSA protocol

in between. The conjectures are the same as in Section 3, (C_{auth} and C_{sec}), with $B^{\mathrm{RSA}}_{\mathsf{auth}}$ instead of $B^{\mathrm{DS}}_{\mathsf{auth}}$ in (C_{auth}).

Again, we show with our procedure that these conjectures are not joinable inductive theorems of $\mathcal{R}$ (by induction on traces), revealing known authentication and secrecy flaws of the protocol.

Let us consider an instance of Conjecture (C_{sec}) where y is replaced by nil and n is replaced by 11. The reductions in Figure 3 shows that it is a counter example and hence that the protocol has a secrecy flaw. The first part of the reduction suggests also a counter-example for Conjecture (C_{auth}), demonstrating an authentication flaw of the protocol.

Formal Analysis of Authentication in Bluetooth Device Pairing

Richard Chang and Vitaly Shmatikov

The University of Texas at Austin

Abstract. Bluetooth is a popular standard for short-range wireless communications. Bluetooth device pairing enables two mobile devices to authenticate each other and establish a secure wireless connection.

We present a formal analysis of authentication properties of Bluetooth device pairing. Using the ProVerif cryptographic protocol verifier, we first analyze the standard device pairing protocol specified in the Bluetooth Core Specification, which relies on short, low-entropy PINs for authentication. Our analysis confirms a previously known attack guessing attack. We then analyze a recently proposed Simple Pairing protocol. Simple Pairing involves Diffie-Hellman-based key establishment, in which authentication relies on a *human visual channel*: owner(s) of the mobile devices confirm the established keys by manually comparing the respective hash values of the parameters used to generate each key, as displayed on the devices' screens. This form of authentication presents an interesting modeling challenge. We demonstrate how to formalize it in ProVerif. Our analysis shows that authentication can fail when the same device is involved in concurrent Simple Pairing sessions. We discuss the implications of this authentication failure for typical Bluetooth usage scenarios. We then refine our model to incorporate session identifiers, and prove that the authentication properties of Simple Pairing hold in the new model.

Out-of-band human verification based on image- or audio-matching is increasingly used for authentication of mobile devices. This study is the first step towards automated analysis of formal models of human-authenticated protocols.

1 Introduction

Bluetooth radios are becoming ubiquitous in mobile devices such as cell phones, laptops, and even many modern cars. Over one billion Bluetooth-enabled devices have been shipped to date [5]. These devices are often used to store users' private data. For example, a user may enjoy the convenience of being able to wirelessly transfer contact data between his or her laptop and a mobile phone, but probably does not want that contact data to be publicly available to all Bluetooth devices in range. The Bluetooth specification [4] supports the establishment of pairwise symmetric keys to allow two devices to securely communicate with each other. The process by which a pair of devices establishes the initial symmetric key is called *device pairing*. The device pairing process comprises authentication, generation of the initialization key, and generation of the link key.

Our main contribution is a formal, tool-supported security analysis of two Bluetooth device pairing protocols. The Simple Pairing protocol presents an interesting challenge to formal verification methods because it relies on out-of-band, *human* authentication (explained in more detail below). Human-verifiable protocols based on image- or audio-matching are becoming increasingly popular for mobile device authentication [12, 13, 7]. Although cryptographic security proofs have been manually derived for similar protocols [9, 15], we view our work as the initial step in applying formal methods to this class of protocols.

The first protocol we analyze is the device pairing protocol defined by the Bluetooth Specification [4]. We will refer to this protocol as *standard pairing*. The second is a recently proposed protocol called *Simple Pairing* [6]. Informally, the security properties we verify for both pairing protocols are (1) key secrecy for the initialization key, and (2) authentication of session participants. Intuitively, our key secrecy property states that upon successfully completing a device pairing between devices A and B, the initialization key is known only to A and B. Our authentication property states that, if A has completed a device pairing in which A believes that it has successfully paired with B, then it has indeed paired with B (and vice versa). To support our analysis, we use ProVerif, an automated verifier for cryptographic protocols [3].

As a warm-up exercise, intended to demonstrate the capabilities of formal verification tools as applied to the security analysis of wireless protocols, we use ProVerif to re-discover a known vulnerability in the standard pairing protocol which allows an attacker to impersonate a Bluetooth device after eavesdropping on a successful pairing session. The attack is a guessing attack against the low-entropy, human-memorable shared secret which is used to generate the initialization key [8].

The main part of this paper is devoted to the analysis of the Simple Pairing protocol, which was designed to rectify vulnerabilities caused by the use of low-entropy secrets in the standard pairing protocol. For key establishment, Simple Pairing uses plain, unauthenticated Diffie-Hellman key exchange. Authentication relies on an interesting out-of-band mechanism. Each device computes a short cryptographic hash of the established key and displays it on the device's screen. The two devices' owner(s) visually compares the displayed values and manually confirms that they match. In other words, authentication is done via key confirmation on a secure human channel, *i.e.*, a human "equality oracle." In this paper, we present the first formal model for this type of authentication amenable to automated analysis. While there are other protocols in the literature that employ similar human authentication mechanisms [12, 13], to the best of our knowledge none of them have been formally analyzed.

When analyzing our model of human key confirmation, ProVerif was unable to prove that the authentication conditions hold. Further manual analysis revealed that the failure is due to an interaction between concurrent protocol sessions. Authentication requires that the keys established by the two participants be equal in any successfully completed protocol session. The hash value displayed by a Bluetooth device does not indicate in *which* session of the protocol the underlying key was established. As our formal analysis shows, even if the human "equality oracle" confirms that two keys (or, more precisely, their hashes) are equal, these keys might not have been established in the same session.

We emphasize that the specification of the Simple Pairing protocol does not appear to allow any input from the user other than a confirmation that two hash values are equal. In this sense, our symbolic model is an appropriate abstraction of user interaction in Simple Pairing. At the same time, the "attack" in the symbolic model is fundamentally concurrent, and only works if the same device is simultaneously engaged in multiple Simple Pairing sessions. Many real Bluetooth implementations can only conduct a single pairing session at any given time—even though this is *not* required by the actual specification, which, by default, appears to permit concurrent executions.

The failure of authentication in our formal model indicates the lack of precision in the Simple Pairing specification. Either the specification should rule out concurrent sessions outright, or it should explicitly require that the hash values presented to the user be accompanied by session identifiers or other means of distinguishing the values established in different sessions. We discuss this further in section 4.3. This may have implications for other wireless protocols, which are secure when multiple instances are executed sequentially, but insecure when they are executed concurrently.

We modify our model to incorporate explicit session identifiers into the equality tests, *i.e.*, in the modified protocol the human oracle is asked to verify not only that the key hashes match, but also that they correspond to the same protocol session. ProVerif has been able to verify both key secrecy and authentication in the modified protocol.

Organization of the paper. Section 2 gives a brief overview of ProVerif. Section 3 describes standard Bluetooth device pairing and our analysis thereof. Section 4 describes the new Simple Pairing protocol based on a human visual channel, and our analysis. Conclusions are in section 5.

2 Analysis Methodology

For our formal verification of Bluetooth device pairing, we use ProVerif, a cryptographic protocol verifier developed by Bruno Blanchet (see [2] for a detailed overview of the syntax and semantics of ProVerif). ProVerif uses a resolution-based solving algorithm which is sound, but not complete. The general approach employed by ProVerif is as follows. The protocol to be verified is specified in a process calculus, explained in more detail below. Each protocol role is represented by a separate process. The complete protocol is modeled by the parallel composition of an unbounded number of copies of the individual protocol role processes.

ProVerif automatically translates these processes into a set of first-order logic formulas (Horn clauses) which abstractly represent the protocol. The solving algorithm takes these clauses as input and determines the set of facts that an attacker can learn from protocol executions. Properties to be verified are modeled as derivability queries. For example, secrecy of a key k can be modeled as a derivability query for the attacker's knowledge k: if ProVerif's solving algorithm terminates and the set of attacker knowledge does not include k, then k has been proved to be secret. Such proofs are sound even with an unbounded number of sessions and without an *a priori* bound on the structural size of messages sent by the attacker.

The process calculus used to specify protocols is an extension of the π-calculus. Among the extensions are function symbols that model cryptographic operations as symbolic "black boxes." Messages are represented by abstract terms. Terms are names, variables, or constructor applications. Constructors are functional symbols used to build terms. For example, encryption is modeled by a constructor: given term m representing a message, and term k representing a symmetric key, the term $encrypt(m, k)$ represents the encryption of m under k generated by the application of the *encrypt* constructor. The process calculus also includes destructors, which are functional symbols that manipulate terms. The decryption destructor, *decrypt*, is modeled by the following reduction rule: $decrypt(encrypt(m, k), k) \rightarrow m$. Other cryptographic primitives are modeled symbolically in a very similar fashion. Communication is modeled by named “channels,” and sending and receiving messages is modeled by inputing and outputting terms over channels.

ProVerif uses the so called Dolev-Yao model of an attacker. The attacker can eavesdrop or intercept any message sent over the network, compute new messages, and send them to protocol participants. In ProVerif’s formalism, the attacker’s knowledge is represented by a set of terms. Let us call this set A. If a process P sends a term M over channel c, and $c \in A$ (*i.e.*, the channel is public and readable by the attacker, modeled symbolically by the fact that the attacker knows its name), then M is added to A. The attacker can also apply constructors to elements of A, generating new elements for A, and send elements of A over channels in A.

After a protocol has been specified as a set of processes using the process calculus described above, ProVerif automatically translates these processes into a set of Horn clauses. Essentially, these Horn clauses represent the attacker’s potential knowledge from protocol executions as a set of implications. A special predicate $\mathit{attacker}(M)$ is used in these clauses to represent the fact that an attacker knows the term M. Another predicate $\mathit{mess}(c, M)$ is used to represent the fact that a message M has been sent by a process over the channel c. The ability of the attacker to receive message M' from the network is represented by the following Horn clause: $\mathit{mess}(c, M') \wedge \mathit{attacker}(c) \Rightarrow \mathit{attacker}(M')$, which basically encodes part of the attacker model above. If a message is sent over a channel that the attacker can read, then the attacker receives that message.

To see how constructors and destructors are translated into Horn clauses, recall the encryption/decryption example above. The *encrypt* constructor is translated into the following clause:

$\mathit{attacker}(m) \wedge \mathit{attacker}(k) \Rightarrow \mathit{attacker}(\mathit{encrypt}(m, k))$.

The *decrypt* destructor is translated into the following clause:

$\mathit{attacker}(\mathit{decrypt}(m, k)) \wedge \mathit{attacker}(k) \Rightarrow \mathit{attacker}(m)$

The solving algorithm employed by ProVerif applies resolution-based theorem proving to this set of clauses to determine the derivability of facts (not unlike logic programming in Prolog). Modeling and verifying basic secrecy properties is relatively straightforward. If one wishes to verify that a private symmetric key, k_s, used in a protocol’s specification is secret, the derivation query $\mathit{attacker}(k_s)$ is given as input to the solving algorithm. The solving algorithm applies resolution with free selection to the set of Horn clauses obtained by translating the process calculus specification. If the algorithm terminates and the attacker’s knowledge set does not include $\mathit{attacker}(k_s)$, then secrecy of k_s is preserved by the protocol.

ProVerif supports verification of security properties other than basic secrecy. Most of these properties are based on the notion of *equivalence* between processes. For example, consider *strong secrecy.* Intuitively, a value is strongly secret if the attacker cannot observe any difference between a process where this value is used, and an "ideal" process where a completely different value (*e.g.*, a fresh random nonce) is used in the same position. (This definition of secrecy is similar in spirit to cryptographic definitions of secrecy, which are based on real-or-random indistinguishability.) If, in fact, there is an observable difference between the real and ideal processes—for example, some destructor application succeeds for one and fails for the other—then a special "bad" fact is derivable. Therefore, strong secrecy properties are modeled as derivability of this fact. ProVerif also supports verification of weak secrecy (useful for reasoning about low-entropy secrets, as we show in our model for standard pairing) and the so called "correspondence assertions" (used for modeling authentication, and also employed in our models).

While ProVerif's solving algorithm is not guaranteed to terminate, if it does terminate, the resulting proof is valid for an unbounded number of sessions. Also, as our analysis of Simple Pairing shows, failure to prove a protocol secure can be used to identify attack traces associated with potential vulnerabilities.

3 Standard Pairing

This section presents the results of our analysis of the Bluetooth standard pairing protocol. We provide an overview of the protocol, discuss the interesting aspects of formal modeling, and present the results of our ProVerif-based analysis, including the guessing attack found by ProVerif.

3.1 Overview of Standard Pairing

The standard pairing protocol aims to enable a pair of Bluetooth devices, A and B, to generate a symmetric initialization key, K_{init}, mutually authenticate each other, and generate a link key. The link key that is generated during a standard pairing session is derived using the initialization key. The protocol specification provides a number of ways for the link key to be generated. One device's unit key can be used as the link key by sending it to the other device under encryption of the initialization key, or each device can generate a random number, send this number under encryption of the initialization key to the other device, after which

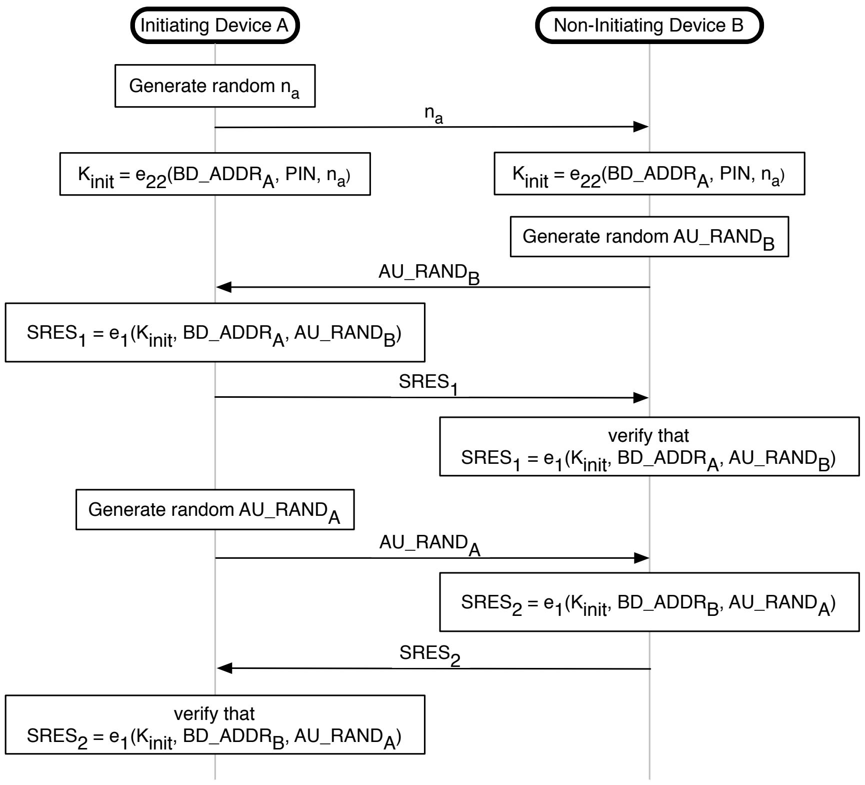
Initiating Device A
Non-Initiating Device B
Generate random n_a
n_a
$K_{init} = e_{22}(BD\_ADDR_A, PIN, n_a)$
$K_{init} = e_{22}(BD\_ADDR_A, PIN, n_a)$
Generate random AU_RAND_B
AU_RAND_B
$SRES_1 = e_1(K_{init}, BD\_ADDR_A, AU\_RAND_B)$
$SRES_1$
verify that $SRES_1 = e_1(K_{init}, BD\_ADDR_A, AU\_RAND_B)$
Generate random AU_RAND_A
AU_RAND_A
$SRES_2 = e_1(K_{init}, BD\_ADDR_B, AU\_RAND_A)$
$SRES_2$
verify that $SRES_2 = e_1(K_{init}, BD\_ADDR_B, AU\_RAND_A)$

Fig. 1. Sequence diagram for Bluetooth standard pairing

the pair of random numbers are used to generate the link key. In some scenarios introduced in newer versions of the specification the link key is generated before mutual authentication is performed, while in others the link key is generated after mutual authentication is performed using the initialization key. Each of these scenarios suffers from the same problem. An offline guessing attack against the low-entropy secret used to generate the initialization key allows an attacker to impersonate a Bluetooth device. We illustrate this attack using the simplest version, which matches the paper [8] where this attack was first discovered.

The sequence diagram for this protocol appears in Figure 1. Initially, both devices share a low-entropy, human-memorable secret value, PIN. A and B also have access to the each other's Bluetooth device addresses. We call these addresses BD_ADDR_A and BD_ADDR_B for A and B, respectively. The device that initiates the standard pairing session is called the initiating device. Assume, without loss of generality, that A is the initiating device. B is then the "non-initiating" device (this awkward terminology is borrowed from the Bluetooth product literature).

A begins the protocol by generating a random nonce n_a and sending it to B. Both devices then compute the initialization key K_{init} as a function of n_a, BD_ADDR_A, and PIN (we omit the details of the key generation function, as it has been extensively studied—*e.g.*, see [14, 11, 10]). A and B then execute a two-way challenge-response for authentication. First, A authenticates itself to B. B generates a new random value AU_RAND_B and sends it to A. Upon receiving the new random value, A computes the response as a function of K_{init}, BD_ADDR_A, and AU_RAND_B. A sends this value to B, which verifies the response. The process is repeated with roles reversed to provide mutual authentication.

It is clear from the protocol specification that if an attacker knows the shared PIN value, then both secrecy and authentication will be violated. Key secrecy is violated because all of the parameters used to compute K_{init} are known to the attacker. Additionally, to successfully impersonate a device it is sufficient to know K_{init} and the device's (public) Bluetooth address.

3.2 Analysis of Standard Pairing

As described in section 3.1, breaking secrecy of PIN is sufficient to violate both secrecy and authentication of standard pairing. Therefore, our analysis focuses on secrecy of PIN.

Modeling secrecy of PIN is fairly subtle. In the standard pairing protocol, PIN is intended to be human-memorable, and therefore has low entropy. In practice, 4-digit PINs are used. The main property of low-entropy secrets that we need to model is that they are *guessable.* If an attacker is able to guess a PIN value and verify his or her guess offline, this could constitute a real attack because exhaustively attempting to verify all 4-digit numbers is computationally feasible.

ProVerif supports reasoning about guessing attacks [3]. Modeling this requires declaring term t in question as a *weaksecret.* While executing its solving algorithm, ProVerif attempts to prove an observational equivalence between the attacker's knowledge given a correctly guessed value for t and the attacker's knowledge given a fresh value. Intuitively, should be no observable difference between a process in which the guessed secret is used, and a process in which a different value is used instead. If observational equivalence holds, then there is no guessing attack. In our

ProVerif model for standard pairing, however, observational equivalence fails.

Msg 1. $A \rightarrow B$: n_a
Msg 2. $B \rightarrow A$: AU_RAND_B
Msg 3. $A \rightarrow B$: $e_1(e_{22}(BD\_ADDR_A, PIN, n_a), BD\_ADDR_A, AU\_RAND_B)$

Fig. 2. Standard Paring Guessing Attack Messages

The attack found by ProVerif is essentially the same offline guessing attack as described in [8]. Figure 2 shows a transcript of messages that an attacker can use to perform a guessing attack on the shared secret PIN value. An attacker can eavesdrop on a successful standard pairing session between A and B and use these three messages to verify a guessed PIN value offline. Verification works as follows. First, the attacker guesses a value for PIN, say, PIN_g. Then the attacker tries to recompute the third message using PIN_g. In order to compute this message, he or she must know BD_ADDR_A, n_a, and AU_RAND_B. BD_ADDR_A is A's Bluetooth address, which is public. The values of n_a and AU_RAND_B are learned by the attacker from the first and second messages, respectively. To verify whether his guess PIN_g is correct, the attacker uses these values to compute $e_1(e_{22}(BD\_ADDR_A, PIN_g, n_a), BD\_ADDR_A, AU\_RAND_B)$ and compares the result to the third message.

4 Simple Pairing

This section presents of the results of our analysis of the Simple Pairing protocol. We give an overview of the protocol, discuss the authentication scheme used in Simple Pairing, and present the results of our ProVerif-based analysis.

The most interesting aspect of our analysis is our formal model of the "human visual channel," which is essentially a secure equality oracle which enables two Bluetooth devices (or, more precisely, their human owner) to test the established keys for equality. Because equality testing is based on the visual comparison of two numbers, a network attacker cannot interfere with it. Intuitively, a visual channel provides a secure "ideal functionality" for comparing two values for equality. This ideal functionality is then leveraged into complete authentication of the key establishment protocol. Similar mechanisms are used for device authentication in other protocols [12, 13].

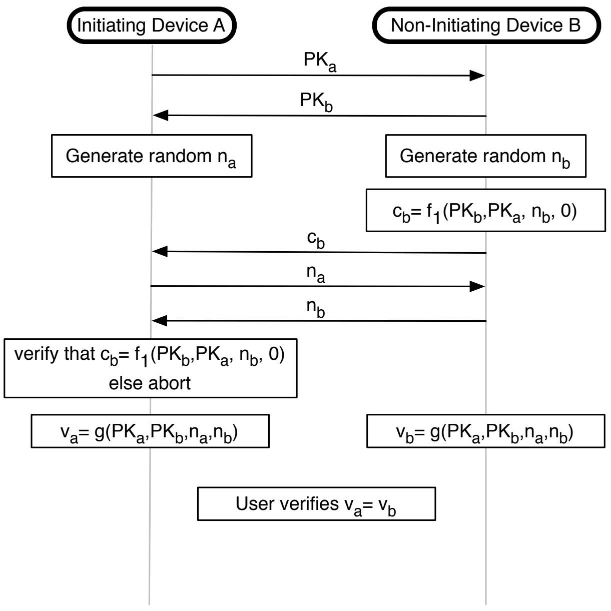
Initiating Device A
Non-Initiating Device B
PK$_a$
PK$_b$
Generate random n$_a$
Generate random n$_b$
c$_b$= f$_1$(PK$_b$,PK$_a$, n$_b$, 0)
c$_b$
n$_a$
n$_b$
verify that c$_b$= f$_1$(PK$_b$,PK$_a$, n$_b$, 0) else abort
v$_a$= g(PK$_a$,PK$_b$,n$_a$,n$_b$)
v$_b$= g(PK$_a$,PK$_b$,n$_a$,n$_b$)
User verifies v$_a$= v$_b$

Fig. 3. Sequence diagram for Bluetooth Simple Pairing

4.1 Overview of Simple Pairing

One of the goals of the Simple Pairing protocol is shared with standard pairing: to enable a pair of Bluetooth devices, A and B, to authenticate each other and generate a symmetric initialization key K_{init}. The differences are as follows. First, Simple Pairing has several distinct association models based on the device's hardware capabilities. Second, under most of these association models Simple Pairing does not use any shared secrets at all, let alone low-entropy PINs. The protocol we analyze in this paper uses the *Numeric Comparison* association model [6]. This association model is designed for pairing Bluetooth devices that have displays, such as a cell phone and a laptop. Authentication involves a human oracle: the user is asked to examine the screens of both devices and confirm that they display the same number.

As before, assume that A is the initiating device. First, A and B exchange Elliptic Curve Diffie-Hellman public values PK_A and PK_B. Then A and B generate random values n_a and n_b, respectively. B computes a commitment value c_b which is a collision-resistant function of both Diffie-Hellman values and n_b, and sends c_b to A. A sends n_a to B. B sends its nonce n_b to A. A re-computes the commitment value c_b and checks whether it is equal to the commitment value previously received from B. Note that there is no authentication so far, *i.e.*, an attacker who controls the communication medium can easily substitute different values into A's and B's messages.

To verify that the parties' views of the conversation match, each device computes a cryptographic hash $H(PK_A, PK_B, n_a, n_b)$ of the two Diffie-Hellman public values PK_A and PK_B, and the two nonces n_a and n_b. The hash value is truncated to a 6 decimal digits, which are displayed on the device's screen. The user is asked to check whether the numbers displayed on the two screens are equal. If the user confirms equality, the devices are considered authenticated, and the symmetric key K_{init} is derived from the joint Diffie-Hellman value and the nonces in the usual way.

4.2 Analysis of Simple Pairing

Our formal model of the Simple Pairing protocol includes a model of the human equality oracle. We formalize it as a special process with two *private channels* which the attacker cannot read or write. Privacy of the channels between the devices and the equality testing process is very important: it models the fact that the user is looking directly at the screen of the Bluetooth device, and the network-based attacker cannot force him to see a number which is different from what the device is displaying. The oracle process reads a value from each channel and outputs a special constant on each channel if the values are equal.

To model secrecy of each device's key, we use standard secrecy in ProVerif, as opposed to weak secrecy. Secrecy of the key values is handled as a derivation query, as mentioned previously.

Modeling authentication is more interesting, and requires *correspondence assertions*. In [1], the ProVerif process calculus is extended to allow processes to emit special `begin` and `end` events that are parameterized by protocol terms. To express authentication of A to B, we modify B's process so that it emits an `end(`B`)` event upon successful completion of a Simple Pairing session, modeling the fact that B has completed a Simple Pairing session. Similarly, A's process is modified to emit a `begin(`B`)` event immediately after public keys are exchanged, modeling the fact that A has started a simple pairing session in which it believes it is pairing with B.

Authentication properties can be expressed as an implication of the following form: if `end(`B`)` is output, then `begin(`B`)` is output. ProVerif supports two types of implication properties, injective and non-injective.

The non-injective version of this property says that if `end(`B`)` is output, then at least one `begin(`B`)` event is output. The injective version is stronger and says that for every `end(`B`)` event, there should be a matching `begin(`B`)` event.

In our analysis, we model both the injective and non-injective versions of two types of authentication properties, which we call weak and strong authentication. *Weak authentication* is modeled by the implication conditions in which the `begin` and `end` events are parameterized only by the participants' identities. If weak authentication holds, then each participant is not mistaken about the identity of the other participant in the protocol, but other session parameters do not necessarily match. *Strong authentication*, on the other hand, is modeled by the implication conditions in which the events are parameterized by all data values used in computing the challenge-response values.

We consider all four types of authentication properties for both A-to-B and B-to-A authentication, modeled modeled as eight correspondence assertion queries. The results of running the ProVerif tool on our Simple Pairing model to check these properties appear in Table 1.

Table 1. Simple Pairing Authentication Properties

Property	Result
A-to-B Injective Weak Authentication	FAILS
B-to-A Injective Weak Authentication	FAILS
A-to-B Non-Injective Weak Authentication	HOLDS
B-to-A Non-Injective Weak Authentication	HOLDS
A-to-B Injective Strong Authentication	FAILS
B-to-A Injective Strong Authentication	FAILS
A-to-B Non-Injective Strong Authentication	FAILS
B-to-A Non-Injective Strong Authentication	FAILS

In our Simple Pairing model, the non-injective weak authentication properties were proved by ProVerif, but all other authentication properties failed, including injective weak authentication and all strong authentication properties. Manually examination of ProVerif's output reveals that the violating derivations involve concurrent execution of multiple Simple Pairing sessions between A and B, and correspond to the situation where the human user's key confirmation is interpreted as successful authentication in a *different* protocol session.

An example of a protocol trace which violates all four strong authentication properties appears in Figure 4. This trace contains messages from two concurrent sessions in which A and B attempt to pair. O represents the equality oracle in our model. Events used by ProVerif to handle correspondence assertions also appear in the trace. They do *not* correspond to actual messages sent in the protocol; we show them to aid the reader in understanding how the properties are violated.

The first part of the trace consists of messages 1 through 7, in which A initiates a Simple Pairing session with B. This session pauses execution immediately after both A and B send their hashes, v_a and v_b respectively, to O. Because this is an uncorrupted session, these values will be the same, and the oracle will eventually return *ok* to both devices. Messages 1' through 7' represent a different session, running concurrently, where again A initiates a Simple Pairing Session with B. In this session, message 4' is actually a corrupted message from an attacker impersonating A, represented by $I(A)$. Instead of receiving A's real nonce for this session, n'_a, B receives n_i. Because A and B's view of A's nonce in the second session differs, the pairing should fail in this session. Specifically, the hashes generated for the human verification step will differ; the keys generated for this session will not agree; and therefore, O should not return *ok* to A and B. Messages 6' and 7' consist of A and B sending their hashes to O for equality verification. At this point in the execution of both sessions, all participants are expecting to receive a reply from O over their respective private channels. Messages 8 and 9 represents a pair of equality confirmation messages to A and B from the first session. In this trace, both instances of A receive message 8 from O and output `end` events corresponding to a successful Simple Pairing session completion. Similarly, both instances of B receive message 9 from O and output the appropriate `end` events. The strong authentication correspondence assertions for the second session are violated by this trace. At the end of this transcript, the event `endBconfirm`(PK_a, PK_b, n_i, n'_b) appears but there is no corresponding `beginBconfirm`(PK_a, PK_b, n_i, n'_b) message. This represents a violation of both the injective and non-injective versions of B-to-A strong authentication. A similar unmatched `endAconfirm`(PK_a, PK_b, n'_a, n'_b) event from the second session appears in the transcript which violates both versions of A-to-B strong authentication.

4.3 Interpreting the Results of Formal Analysis

What do the results of our formal analysis imply about the security of Bluetooth Simple Pairing, as deployed on real-world mobile devices?

Msg 1.	$A \rightarrow B : PK_a$
event beginAparam(PK_a).	
Msg 2.	$B \rightarrow A : PK_b$
event beginBparam(PK_b).	
Msg 3.	$B \rightarrow A : f_1(PK_b, PK_a, n_b, 0)$
Msg 4.	$A \rightarrow B : n_a$
event beginAconfirm(PK_a, PK_b, n_a, n_b).	
Msg 5.	$B \rightarrow A : n_b$
event beginBconfirm(PK_a, PK_b, n_a, n_b).	
Msg 6.	$A \rightarrow O : v_a$
Msg 7.	$B \rightarrow O : v_b$
Msg 1'.	$A \rightarrow B : PK_a$
event beginAparam(PK_a).	
Msg 2'.	$B \rightarrow A : PK_b$
event beginBparam(PK_b).	
Msg 3'.	$B \rightarrow A : f_1(PK_b, PK_a, n'_b, 0)$
Msg 4'.	$I(A) \rightarrow B : n_i$
event beginAconfirm(PK_a, PK_b, n_i, n'_b).	
Msg 5'.	$B \rightarrow A : n'_b$
event beginBconfirm(PK_a, PK_b, n'_a, n'_b).	
Msg 6'.	$A \rightarrow O : v'_a$
Msg 7'.	$B \rightarrow O : v_i$
Msg 8.	$O \rightarrow A : ok$
event endAparam(PK_a).	
event endAparam(PK_a).	
event endAconfirm(PK_a, PK_b, n_a, n_b).	
event endAconfirm(PK_a, PK_b, n'_a, n'_b).	
Msg 9.	$O \rightarrow B : ok$
event endBparam(PK_b).	
event endBparam(PK_b).	
event endBconfirm(PK_a, PK_b, n_a, n_b).	
event endBconfirm(PK_a, PK_b, n_i, n'_b).	

Fig. 4. Simple Paring trace in which strong authentication is violated

Formal verification has been very successful in analyzing (and discovering bugs in) a broad class of network protocols such as SSL/TLS and IKE, which are intended to operate in Internet-like environments, where concurrent execution of multiple instances of the same protocol is a fact of life (*e.g.*, a single Web server may be carrying out hundreds of concurrent TLS sessions). Therefore, protocol analysis tools such as ProVerif are designed to prove that secrecy and authentication hold for an *unbounded* number of concurrent and sequential sessions. In ProVerif, this is modeled by allowing unbounded replication of the authenticating processes.

This strong notion of security is certainly appropriate for Internet security protocols. Bluetooth, on the other hand, is designed for more constrained environments. In particular, it is not clear whether any Bluetooth device ever has the need to carry out concurrent pairing sessions, or is even capable of doing it. The Simple Pairing white paper does not address concurrent executions at all, nor does it forbid the same device from being engaged in concurrent sessions.

The failure of authentication discovered by ProVerif and illustrated by the trace in Figure 4 represents a genuine attack *when the same device is engaged in concurrent sessions.* In this scenario, the number displayed on the screen for human verification may not have been computed in the session which the user is trying to authenticate.

Effectively, our analysis shows that if the same device is allowed to execute multiple concurrent instances of Simple Pairing, then its user interaction component must associate session identifiers with the hash values it displays to the user (and thus deviate from the Simple Pairing specification). If concurrent executions are not permitted, then this restriction should be clearly spelled out in the protocol specification. Otherwise, Simple Pairing or a similar protocol may end up being deployed in a setting where concurrent executions are a possibility, *e.g.*, when the same mobile phone is trying to pair simultaneously with a music player and a laptop.

One may argue that no "reasonable" implementation would support concurrent sessions without giving the user a means of distinguishing hash values produced in different sessions. This may or may not be true, since such an implementation would *not* be compliant with the current Simple Pairing specification, in which user interaction is limited to displaying two values and asking whether they are equal. It is hard to guess, let alone analyze formally, under what circumstances a "reasonable" implementor might deviate from the protocol specification. It is better to explicitly include all restrictions in the specification.

Normally user interaction is not considered as part of the specification, but in protocols where authentication relies on *human* input (*e.g.*, all protocols authenticated via short authenticated string), security depends on correctness of user interaction, that is, representing the right state of the right instance of the protocol to the user. Our analysis points out not so much an inherent flaw of Simple Pairing, but how important it is that the user interaction piece of the system match the underlying protocol precisely.

4.4 Fixing the Simple Pairing Protocol

The vulnerability discovered in our model involves the human user's key confirmation from one session being interpreted as successful authentication in a different session. Therefore, we propose a minor modification to the Simple Pairing protocol, in which *session identifiers* are included in the messages to and from the human oracle. These identifiers allow each "instance" of a device in a given session to correctly identify which confirmation messages correspond to its session.

We built a process calculus model for the modified Simple Pairing protocol, in which each instance of the same device (*i.e.*, when the same device is participating in multiple instances of the protocol) is assigned a unique session identifier. We also added a check to ensure that each session instance only accepts human oracle responses associated this instance's session identifier. Therefore, there is no possibility that an instance accepts an oracle response intended for a different instance (recall that the channel between the human oracle and the instance is the device's physical interface, presumed to be secure). In the new model, ProVerif was able to prove both the injective and non-injective versions of weak and strong authentication, even in the case of an unbounded number of concurrent sessions.

5 Conclusions

We have presented an automated, tool-supported analysis of two Bluetooth authentication protocols—one based on low-entropy PINs, the other based on human verification of equality. For the standard protocol, our analysis confirms a previously known guessing attack. For the new Simple Pairing protocol, we discover a potential vulnerability caused concurrent executions of authentication, and show how to fix the problem by adding explicit session identifiers to the protocol.

From the verification perspective, our main contribution is a formal model for a non-standard form of authentication, which is based not on cryptography, but on access to a "human oracle" who visually checks whether the key confirmation values derived by the two devices match. This is an increasingly popular device authentication technique [12, 13]. We expect that our work will serve as the first step towards richer formal models of human authentication.

An interesting topic for future work is to develop a proper cryptographic (rather than symbolic) proof of security for Simple Pairing. In such a proof, human oracles can be modeled as a secure ideal functionality for equality testing, and the protocol can be proved secure relative to this functionality.

References

1. B. Blanchet. From secrecy to authenticity in security protocols. In *Proc. SAS*, 2002.
2. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. IEEE S&P*, 2004.
3. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *Proc. LICS*, 2005.
4. Bluetooth Special Interest Group. Specification of the Bluetooth system. `http://www.bluetooth.com/NR/rdonlyres/1F6469BA-6AE7-42B6-B5A1-65148B9DB238/840/Core_v210_EDR.zip`, 2004.
5. Bluetooth Special Interest Group. Bluetooth wireless technology surpasses one billion devices. `http://www.bluetooth.com/Bluetooth/Press/SIG/BLUETOOTH_WIRELESS_TECHNOLOGY_SURPASSES_ONE_BILLION_DEVICES.htm`, 2006.
6. Bluetooth Special Interest Group. Simple pairing whitepaper. `http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`, 2006.
7. M. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Urzun. Human-verifiable authentication based on audio. In *Proc. ICDCS*, 2006.
8. M. Jakobsson and S. Wetzel. Security weaknesses in Bluetooth. In *Proc. CT-RSA*, 2001.
9. S. Laur and K. Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Proc. CANS*, 2006.
10. O. Levy and A. Wool. A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. In *Proc. SecureComm*, 2005.
11. Y. Lu and S. Vaudenay. Cryptanalysis of Bluetooth keystream generator two-level E0. In *Proc. ASIACRYPT*, 2004.
12. J. McCune, A. Perrig, and M. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proc. IEEE S&P*, 2005.
13. N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Proc. IEEE S&P*, 2006.
14. J. Vainio. Bluetooth security. `http://www.niksula.cs.hut.fi/~jiitv/bluesec.html`, 2000.
15. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Proc. CRYPTO*, 2005.

Deciding knowledge in security protocols for monoidal equational theories [⋆]

Véronique Cortier and Stéphanie Delaune

LORIA, CNRS & INRIA project Cassis, Nancy, France

Abstract. In formal approaches, messages sent over a network are usually modeled by terms together with an equational theory, axiomatizing the properties of the cryptographic functions (encryption, exclusive or, ...). The analysis of cryptographic protocols requires a precise understanding of the attacker knowledge. Two standard notions are usually used: deducibility and indistinguishability. Only few results have been obtained (in an ad-hoc way) for equational theories with associative and commutative properties, especially in the case of static equivalence. The main contribution of this paper is to propose a general setting for solving deducibility and indistinguishability for an important class (called *monoidal*) of these theories. Our setting relies on the correspondence between a monoidal theory E and a semiring $\mathcal{S}_{\mathsf{E}}$ which allows us to give an algebraic characterization of the deducibility and indistinguishability problems. As a consequence we recover easily existing decidability results and obtain several new ones.

1 Introduction

Security protocols are paramount in today's secure transactions through public channels. It is therefore essential to obtain as much confidence as possible in their correctness. Formal methods have proved their usefulness for precisely analyzing the security of protocols. Understanding security protocols often requires reasoning about knowledge of the attacker. In formal approaches, two main kind of definitions have been given in the literature for this knowledge. They are known as message deducibility and indistinguishability relations.

Most often, the knowledge of the attacker is described in terms of message deducibility [13, 16, 14]. Given some set of messages ϕ representing the knowledge of the attacker and another message M, intuitively the secret, one can ask whether an attacker is able to compute M from ϕ. To obtain such a message he uses his deduction capabilities. For instance, he may encrypt and decrypt using keys that he knows.

[⋆] This work has been partly supported by the RNTL project POSÉ and the ARA SSIA Formacrypt

This concept of deducibility does not always suffice for expressing the knowledge of an attacker. For example, if we consider a protocol that transmits an encrypted Boolean value (e.g. the value of a vote), we may ask whether an attacker can learn this value by eavesdropping the protocol. Of course, it seems to be completely unrealistic to say that the Boolean true and false are not deducible. We need to express the fact that the two transcripts of the protocol, one running with the Boolean value true and the other one with false are *indistinguishable.*

In both cases, deduction and indistinguishability apply to observations on messages at a particular point in time. They do not take into account the dynamic behavior of the protocol. For this reason the indistinguishability relation is called *static equivalence.* Nevertheless those relations are quite useful to reason about the dynamic behavior of a protocol. For instance, the deducibility relation is often used as a subroutine of many decision procedures [17, 7, 8]. In the applied-pi calculus framework [2], it has been shown that observational equivalence (relation which takes into account the dynamic behavior) coincides with labeled bisimulation which corresponds to checking static equivalences and some standard bisimulation conditions.

Both of these relations rely on an underlying equational theory axiomatizing the properties of the cryptographic functions (encryption, exclusive or, . . .). Many decision procedures have been provided to decide these relations under a variety of equational theories. For instance algorithms for deduction are provided for exclusive or [8], homomorphic operators [9] and subterm theories [1]. These theories allow basic equations for functions such as encryption, decryption and digital signature. There are also results for static equivalence. For instance, a general decidability result for the class of subterm convergent equational theories is given in [1]. This class contains classical cryptographic primitives like encryption, signatures and hashes. Also in [1] some abstract conditions on the underlying equational theory are proposed to ensure decidability of deduction and static equivalence. Note that the use of this result requires checking some assumptions, which might be difficult to prove. Regarding theories with associative and commutative properties (AC), they only obtain decidability for pure AC and exclusive or. A weakness of most of these approaches is their lack of generality since each new theory requires a new proof. Homomorphic properties occur in many protocols, alone or in combination with other operators, and cannot be dealt with by a simple adaptation of the techniques that have been developed so far.

In this paper, we consider the axioms of Associativity-Commutativity (AC), Unit element (U), Nilpotency (N), Idempotency (I), homomorphism (h), and more especially the combinations of these axioms that constitute monoidal theories. We propose a general approach to handle *monoidal* theories that covers several cases already studied, and furthermore includes some new decidability and complexity results on homomorphic operators. Monoidal theories have been extensively studied by F. Baader and W. Nutt [15, 4, 5] who have provided a complete survey of unification in these theories. More recently, these theories have been studied in the context of security protocols. S. Delaune *et al.* have shown that deduction is decidable for a subclass of monoidal equational theories, also considering active attacks [10]. However, they do not address static equivalence.

Studying monoidal theories might seem very restricted since they do not contain the equational theories for classical operators like encryption or signatures. However, it has been shown in [3] that equational theories can easily be combined for both deduction and static equivalence, provided the signatures are disjoint. That is why it is sufficient to focus on the important case of monoidal theories. As a consequence of our general approach, we recover many existing results and we obtain several new ones (10 new decidability or complexity results) for static equivalence or deduction.

Outline of the paper. In Section 2 we recall some basic notation and the central notion of monoidal theory. Then, in Section 3, we define the two notions of knowledge we are interested in. In Section 4 we show how to represent terms and substitutions by means of vectors and matrices over semirings. Then Sections 5 and 6 are devoted to the study of deduction and static equivalence respectively. In Section 7, we sum up our results and provide new results obtained as a consequence of our main theorems.

2 Preliminaries

2.1 Terms

A *signature* Σ consists of a finite set of function symbols, each with an arity. A function symbol with arity 0 is a constant symbol. We assume given a signature Σ, an infinite set of names $\mathcal{N}$, and an infinite set of variables $\mathcal{X}$. Let $\mathcal{M}$ be a set of names and variables, we denote by $\mathcal{T}(\Sigma, \mathcal{M})$ the set of *terms* over $\Sigma \cup \mathcal{M}$. $\mathcal{T}(\Sigma, \mathcal{N})$ is called the set of *ground* terms while $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ is simply called the set of terms. We write $fn(M)$ (resp. $fv(M)$) for the set of names (resp. variables) that occur in the term M. A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ called its domain and written $dom(\sigma)$ to $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. Substitutions are extended to endomorphisms of $\mathcal{T}(\Sigma, \mathcal{X})$ as usual. We use a postfix notation for their application. Given two terms u and v, the *replacement* of u by v, denoted by $[u \mapsto v]$, maps every term t to the term $t[u \mapsto v]$ which is obtained by replacing all occurrences of u in t by v.

2.2 Monoidal theories

Equational theories are very useful for modeling the algebraic properties of the cryptographic primitives. Given a signature Σ, an equational theory E is a set of equations (*i.e.* a set of unordered pairs of terms in $\mathcal{T}(\Sigma, \mathcal{X})$). Given two terms M and N such that $M, N \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$, we write $M =_{\mathsf{E}} N$ if the equation $M = N$ is a consequence of E. In this paper, we are particularly interested in the class of monoidal theories introduced by W. Nutt [15]. It captures many theories with AC properties, which are known to be difficult to deal with.

Definition 1 (monoidal theory). *A theory* E *over* Σ *is called* monoidal *if it satisfies the following properties:*

1. *The signature* Σ *contains a binary function symbol* $+$ *and a constant symbol* 0*, and all other function symbols in* Σ *are unary.*
2. *The symbol* $+$ *is associative-commutative with unit* 0, i.e. *the equations* $x + (y + z) = (x + y) + z$, $x + y = y + x$ *and* $x + 0 = x$ *are in* E.
3. *Every unary function symbol* $\mathsf{h} \in \Sigma$ *is an endomorphism for* $+$ *and* 0, *i.e.* $\mathsf{h}(x + y) = \mathsf{h}(x) + \mathsf{h}(y)$ *and* $\mathsf{h}(0) = 0$.

Example 1. Suppose $+$ is a binary function symbol and 0 is nullary. Moreover assume that the others symbols, *i.e* $-$, h, are unary symbols. The equational theories below are monoidal.

– The theory ACU over $\Sigma = \{+, 0\}$ which consists of the axioms of associativity and commutativity with unit 0.
– The theories ACUI and ACUN (*exclusive or*) over $\Sigma = \{+, 0\}$ which consist of the axioms (AC) and (U) with in addition Idempotency (I) $x + x = x$, or Nilpotency (N) $\quad x + x = 0$.
– The theory AG (*Abelian groups*) over $\Sigma = \{+, -, 0\}$ which is generated by the axioms (AC), (U) and $x + -(x) = 0$ (Inv). Indeed, the equations $-(x + y) = -(x) + -(y)$ and $-0 = 0$ are consequences of the others.
– The theories ACUh, ACUIh, ACUNh over $\Sigma = \{+, \mathsf{h}, 0\}$ and AGh over $\Sigma = \{+, -, \mathsf{h}, 0\}$: these theories correspond to the ones described

above extended by the homomorphism laws (h) for the symbol h, *i.e.* $\mathsf{h}(x+y) = \mathsf{h}(x) + \mathsf{h}(y)$ and $\mathsf{h}(0) = 0$ (if it is not a consequence of the others equations).

Note that there are two homomorphisms in the theory AGh, namely $-$ and h. These two homomorphisms commute: $\mathsf{h}(-x) = -(\mathsf{h}(x))$ is a consequence of the others. Other examples of monoidal theories can be found in [15].

3 Deduction and static equivalence

We now describe our two notions of knowledge for an intruder.

3.1 Assembling terms into frames

At a particular point in time, while engaging in one or more sessions of one or more protocols, an attacker may know a sequence of messages $M_1, \dots, M_\ell$. This means that he knows each message but he also knows in which order he obtained the messages. So it is not enough for us to say that the attacker knows the set of terms $\{M_1, \dots, M_\ell\}$ since the information about the order is lost. Furthermore, we should distinguish those names that the attacker knows from those that were freshly generated by others and which are *a priori* secret from the attacker; both kinds of names may appear in the terms. In the applied pi calculus [2], such a sequence of messages is organized into a *frame* $\phi = \nu\tilde{n}.\sigma$, where $\tilde{n}$ is a finite set of *restricted* names (intuitively the fresh ones), and σ is a substitution of the form:

$$\{^{M_1}/_{x_1}, \dots, {}^{M_\ell}/_{x_\ell}\} \quad \text{with} \quad \mathit{dom}(\sigma) = \{x_1, \dots, x_\ell\}.$$

The variables enable us to refer to each M_i and we always assume that the terms M_i are ground. The names $\tilde{n}$ are bound to ϕ and can be renamed. Moreover names that do not appear in ϕ can be added or removed from $\tilde{n}$. In particular, we can always assume that two frames share the same set of restricted names.

3.2 Deduction

Given a frame ϕ that represents the information available to an attacker, we may ask whether a given ground term M may be deduced from ϕ.

Given a theory E over Σ, this relation is written $\phi \vdash_{\mathsf{E}} M$ and is axiomatized by the rules:

$$\frac{}{\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M} \text{ if } \exists x \in dom(\sigma) \text{ s.t. } x\sigma = M \qquad \frac{}{\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} s} \; s \in \mathcal{N} \smallsetminus \tilde{n}$$

$$\frac{\phi \vdash_{\mathsf{E}} M_1 \;\; \dots \;\; \phi \vdash_{\mathsf{E}} M_\ell}{\phi \vdash_{\mathsf{E}} f(M_1, \dots, M_\ell)} \; f \in \Sigma \qquad \frac{\phi \vdash_{\mathsf{E}} M}{\phi \vdash_{\mathsf{E}} M'} \; M =_{\mathsf{E}} M'$$

Intuitively, the deducible messages are the messages of ϕ and the names that are not protected in ϕ, closed by equality in E and closed by application of function symbols. Since the deducible messages depend on the underlying equational theory, we write $\vdash_{\mathsf{E}}$ and simply $\vdash$ when E is clear from the context. When $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$, any occurrence of names from $\tilde{n}$ in M is bound by $\nu\tilde{n}$. So $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$ could be formally written $\nu\tilde{n}.(\sigma \vdash_{\mathsf{E}} M)$. It is easy to prove by induction the following characterization of deduction.

Lemma 1 (characterization of deduction). *Let M be a ground term and $\nu\tilde{n}.\sigma$ be a frame. Then $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$ iff there exists $\zeta \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ such that $fn(\zeta) \cap \tilde{n} = \emptyset$ and $\zeta\sigma =_{\mathsf{E}} M$. Such a term ζ is a* recipe *of the term M.*

Example 2. Consider $\Sigma = \{+, 0\}$ and the equational theory ACUN given in Example 1. Let $\phi = \nu n_1, n_2, n_3.\{^{n_1+n_2+n_3}/_{x_1}, {}^{n_1+n_2}/_{x_2}, {}^{n_2+n_3}/_{x_3}\}$. We have that $\phi \vdash n_2 + n_4$. Indeed the term $x_1 + x_2 + x_3 + n_4$ is a recipe of the term $n_2 + n_4$.

Deduction problem for the equational theory E built over Σ.
Entries: A frame ϕ and a term M (both built over Σ)
Question: $\phi \vdash_{\mathsf{E}} M$?

3.3 Static equivalence

Deduction does not always suffice for expressing the knowledge of an attacker. Sometimes, the attacker can deduce exactly the same set of terms from two different frames but he could still be able to tell the difference between these two frames. Static equivalence is particularly important when defining for example the confidentiality of a vote or anonymity-like properties.

Definition 2 (static equivalence). *Let ϕ be a frame and M,N be two terms. We say that M and N are* equal in ϕ *under the theory E, and* *write* $(M =_{\mathsf{E}} N)\phi$*, if there exists* $\tilde{n}$ *such that* $(fn(M) \cup fn(N)) \cap \tilde{n} = \emptyset$*,* $\phi = \nu\tilde{n}.\sigma$ *and* $M\sigma =_{\mathsf{E}} N\sigma$*. We say that two frames* $\phi_1 = \nu\tilde{n}.\sigma_1$ *and* $\phi_2 = \nu\tilde{n}.\sigma_2$ *are* statically equivalent *w.r.t.* E*, and write* $\phi_1 \approx_{\mathsf{E}} \phi_2$ *when* $dom(\phi_1) = dom(\phi_2)$*, and*

$$\forall M, N \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X}) \text{ we have that } (M =_{\mathsf{E}} N)\phi_1 \Leftrightarrow (M =_{\mathsf{E}} N)\phi_2.$$

Example 3. Consider the equational theory ACU given in Example 1 and let $\phi = \nu n_1, n_2, n_3.\{^{3n_1+2n_2+3n_3}/_{x_1}, {}^{n_2+3n_3}/_{x_2}, {}^{3n_2+n_3}/_{x_3}, {}^{3n_1+n_2+4n_3}/_{x_4}\}$ where the notation kn with $k \in \mathbb{N}$ denotes $n + \cdots + n$ (k times). Let $M = 2x_1 + x_2$ and $N = x_3 + 2x_4$. We have that $(M =_{\mathsf{E}} N)\phi$.

Static equivalence problem for the theory E built over Σ.
Entries: Two frames ϕ_1 and ϕ_2 (both built over Σ)
Question: $\phi_1 \approx_{\mathsf{E}} \phi_2$?

In what follows, we consider decidability and complexity issues for deduction and static equivalence for monoidal theories.

4 Monoidal theories

It has been shown that the deduction problem for ACU amounts to solving linear equations over the semiring $\mathbb{N}$ whereas for AGh this problem amounts to solving linear equations over the ring $\mathbb{Z}[\mathsf{h}]$, the ring of polynomials in one indeterminate with coefficients over $\mathbb{Z}$ [9]. Some results of this kind also exist in the case of static equivalence. For instance, static equivalence has been shown decidable for the equational theories ACUN and AC [1]. By using an algebraic characterization of the problem, we will generalize these results by associating to every monoidal theory E a semiring $\mathcal{S}_{\mathsf{E}}$, that will be used to solve the deduction and the static equivalence problems in E.

4.1 Monoidal theories define semirings

Monoidal theories have an algebraic structure close to rings except that elements might not have an inverse. Such a structure is called *semiring.*

Definition 3 (semiring). *A* semiring *is a set* $\mathcal{S}$ *(called the* universe *of the semiring) with distinct elements* 0 *and* 1 *that is equipped with two binary operations* $+$ *and* $\cdot$ *such that* $(\mathcal{S}, +, 0)$ *is a commutative monoid,* $(\mathcal{S}, \cdot, 1)$ *is a monoid, and the following identities hold for all* $\alpha, \beta, \gamma \in \mathcal{S}$*:*

– $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$ *(right distributivity)*

- $\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$ *(left distributivity)*
- $0 \cdot \alpha = \alpha \cdot 0 = 0$ *(zero laws).*

We call the binary operations $+$ and $\cdot$ respectively the *addition* and the *multiplication* of the semiring. The elements 0 and 1 are called respectively *zero* and *unit.* In the sequel we will often omit the $\cdot$ sign and write $\alpha\beta$ instead of $\alpha \cdot \beta$. A semiring is *commutative* if its multiplication is commutative. Semirings are different from rings in that they need not be groups with respect to addition. Every ring is a semiring. In a ring, we will denote by $-\alpha$ the additive inverse of α, and we write $\alpha - \beta$ as an abbreviation of $\alpha + (-\beta)$.

It has been shown in [15] that for any monoidal theory E there exists a corresponding semiring $\mathcal{S}_{\mathsf{E}}$. We can rephrase the definition of $\mathcal{S}_{\mathsf{E}}$ as follows. Let $\mathbf{1}$ be a free constant ($\mathbf{1} \notin \Sigma$), the universe of $\mathcal{S}_{\mathsf{E}}$ is $\mathcal{T}(\Sigma, \{\mathbf{1}\})/\mathsf{E}$, that is the set of equivalence classes of terms built over Σ and $\mathbf{1}$ under equivalence by the equational axioms E. The constant 0 and the sum $+$ of the semiring are defined as in the algebra $\mathcal{T}(\Sigma, \{\mathbf{1}\})/\mathsf{E}$. The multiplication in the semiring is defined by $s \cdot t := s[\mathbf{1} \mapsto t]$. As a consequence, $\mathbf{1}$ acts as a neutral element of multiplication in $\mathcal{S}_{\mathsf{E}}$. This is the reason why we call this new generator $\mathbf{1}$ instead of, say, x, as it is often done in the literature. It can be shown [15] that $\mathcal{S}_{\mathsf{E}}$ is a ring if, and only if, E is a group theory, and also that $\mathcal{S}_{\mathsf{E}}$ is commutative if, and only if, E has commuting homomorphisms, *i.e.* $\mathsf{h}_1(\mathsf{h}_2(x)) =_{\mathsf{E}} \mathsf{h}_2(\mathsf{h}_1(x))$ for any two homomorphisms h_1 and h_2. For instance, we have that

1. The semiring $\mathcal{S}_{\mathsf{ACU}}$ is isomorphic to $\mathbb{N}$, the semiring of natural numbers.
2. The semiring $\mathcal{S}_{\mathsf{ACUN}}$ consists of the two elements 0 and $\mathbf{1}$ and we have $0 + \mathbf{1} = \mathbf{1} + 0 = \mathbf{1}$, $0 + 0 = \mathbf{1} + \mathbf{1} = 0$, $0 \cdot 0 = \mathbf{1} \cdot 0 = 0 \cdot \mathbf{1} = 0$, and $\mathbf{1} \cdot \mathbf{1} = \mathbf{1}$. Hence, $\mathcal{S}_{\mathsf{ACUN}}$ is isomorphic to the commutative ring (field) $\mathbb{Z}/2\mathbb{Z}$.
3. The semiring $\mathcal{S}_{\mathsf{AGh}}$ is isomorphic to $\mathbb{Z}[\mathsf{h}]$ which is a commutative ring.

Let b be a free symbol (name or variable). We denote by $\psi_b \colon \mathcal{T}(\Sigma, \{b\}) \to \mathcal{S}_{\mathsf{E}}$ the function which maps any term $M \in \mathcal{T}(\Sigma, \{b\})$ to $M[b \mapsto \mathbf{1}]$ considered as an element of the semiring $\mathcal{S}_{\mathsf{E}}$.

Example 4. Let $\mathsf{E} = \mathsf{ACUN}$ and $t = b+b+b$. We have $\psi_b(t) = \mathbf{1}+\mathbf{1}+\mathbf{1} = \mathbf{1}$.

4.2 Representation of terms and frames

A *base* $\mathcal{B}$ is a sequence $[b_1, \ldots, b_m]$ of free symbols (names or variables), We say that $\mathcal{B}$ is a *base of names* when $b_1, \ldots, b_m$ are names.

Definition 4 (decomposable in a base). *A term $M \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ is* decomposable *in $\mathcal{B}$ if $fn(M) \cup fv(M) \subseteq \mathcal{B}$. Let $\phi = \nu\tilde{n}.\{^{M_1}/_{x_1}, \dots, ^{M_\ell}/_{x_\ell}\}$ be a frame. We say that ϕ is* decomposable *in $\mathcal{B}$ if each M_i is decomposable in $\mathcal{B}$.*

Let $\mathcal{B} = [b_1, \dots, b_m]$. We generalize the construction of the previous section and obtain a function which assigns to any term in $\mathcal{T}(\Sigma, \mathcal{B})$ a tuple in $\mathcal{S}_{\mathsf{E}}^m$, that is a tuple of m elements over $\mathcal{S}_{\mathsf{E}}$. The function $\psi_{\mathcal{B}}$ of domain $\mathcal{T}(\Sigma, \{b_1, \dots, b_m\})$ is defined as follows: Any term $M \in \mathcal{T}(\Sigma, \{b_1, \dots, b_m\})$ has a unique decomposition $M_1, \dots, M_m$ such that $M = M_1 + \dots + M_m$ with $M_i \in \mathcal{T}(\Sigma, \{b_i\})$ [15]. Let $\psi_{\mathcal{B}}(M) = (\psi_{b_1}(M_1), \dots, \psi_{b_m}(M_m)) \in \mathcal{S}_{\mathsf{E}}^m$. Given a vector $X \in \mathcal{S}_{\mathsf{E}}^m$ of size m, $\psi_{\mathcal{B}}^{-1}(X)$ is a term $M \in \mathcal{T}(\Sigma, \mathcal{B})$ such that $\psi_{\mathcal{B}}(M) = X$. This term is uniquely defined modulo E.

Example 5. Taking into account that the semiring $\mathcal{S}_{\mathsf{AGh}}$ is (isomorphic to) $\mathbb{Z}[\mathsf{h}]$, we have $\psi_{[b_1,b_2,b_3]}(b_1 + b_1 + \mathsf{h}(b_3) + \mathsf{h}(\mathsf{h}(\mathsf{h}(b_3)))) = (2, 0, \mathsf{h} + \mathsf{h}^3)$. Indeed, we have that $\psi_{b_1}(b_1 + b_1) = 2$, $\psi_{b_2}(0) = 0$ and $\psi_{b_3}(\mathsf{h}(b_3) + \mathsf{h}(\mathsf{h}(\mathsf{h}(b_3)))) = \mathsf{h} + \mathsf{h}^3$.

A term can be uniquely decomposed on a base $\mathcal{B}$. This can be extended to associate a (unique) matrix to a frame. Let $\phi = \nu\tilde{n}.\sigma$ be a frame and $\mathcal{B} = [b_1, \dots, b_m]$ be a base of names in which ϕ is decomposable. Let $\sigma = \{^{M_1}/_{x_1} \dots ^{M_\ell}/_{x_\ell}\}$. We denote by $\psi_{\mathcal{B}}(\phi)$ the matrix of size $\ell \times m$ (ℓ rows and m columns) defined by $(\psi_{\mathcal{B}}(M_1); \dots; \psi_{\mathcal{B}}(M_\ell))$. This matrix is the decomposition of ϕ in $\mathcal{B}$.

Example 6. Consider the frame ϕ given in Example 3 and consider the base $\mathcal{B} = [n_1, n_2, n_3]$. We have that

$$\psi_{\mathcal{B}}(\phi) = \begin{pmatrix} 3 & 2 & 3 \\ 0 & 1 & 3 \\ 0 & 3 & 1 \\ 3 & 1 & 4 \end{pmatrix} \quad \text{since} \quad \begin{array}{l} -\ \psi_{\mathcal{B}}(3n_1 + 2n_2 + 3n_3) = (3, 2, 3), \\ -\ \psi_{\mathcal{B}}(n_2 + 3n_3) = (0, 1, 3), \\ -\ \psi_{\mathcal{B}}(3n_2 + n_3) = (0, 3, 1), \text{ and} \\ -\ \psi_{\mathcal{B}}(3n_1 + n_2 + 4n_3) = (3, 1, 4). \end{array}$$

Applying a recipe to a frame is equivalent to multiplying the corresponding matrices.

Lemma 2. *Let $\phi = \nu\tilde{n}.\sigma$ be a frame and ζ be a term in $\mathcal{T}(\Sigma, dom(\phi))$. Let $\mathcal{B}$ be a base of names in which we can decompose ϕ. We have that*

$$\psi_{\mathcal{B}}(\zeta\sigma) = \psi_{dom(\phi)}(\zeta) \cdot \psi_{\mathcal{B}}(\phi).$$

Proof. Let $\sigma = \{{}^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}\}$ and ζ be a term in $\mathcal{T}(\Sigma, dom(\phi))$. We have that $\zeta = \zeta_1 + \ldots + \zeta_\ell$ for some $\zeta_i \in \mathcal{T}(\Sigma, \{x_i\})$.

$$\begin{aligned}
\psi_{\mathcal{B}}(\zeta\sigma) &= \psi_{\mathcal{B}}(\zeta_1\sigma + \ldots + \zeta_\ell\sigma) \\
&= \psi_{\mathcal{B}}(\zeta_1[x_1 \mapsto M_1] + \ldots + \zeta_\ell[x_\ell \mapsto M_\ell]) \\
&= \psi_{\mathcal{B}}(\zeta_1[x_1 \mapsto M_1]) + \ldots + \psi_{\mathcal{B}}(\zeta_\ell[x_\ell \mapsto M_\ell]) \\
&= \psi_{x_1}(\zeta_1) \cdot \psi_{\mathcal{B}}(M_1) + \ldots + \psi_{x_\ell}(\zeta_\ell) \cdot \psi_{\mathcal{B}}(M_\ell) \\
&= \psi_{dom(\phi)}(\zeta) \cdot \psi_{\mathcal{B}}(\phi) \qquad \square
\end{aligned}$$

Note that to apply the equation stated in Lemma 2, the recipe ζ has to be built without names. To ensure that such kind of recipes always exist, we will work with frames saturated w.r.t. $\mathcal{B}$ (base of names in which the frames are decomposable).

Definition 5 (frame saturated w.r.t. $\mathcal{B}$). *Let $\phi = \nu\tilde{n}.\sigma$ be a frame and $\mathcal{B}$ be a base of names $[b_1, \ldots, b_m]$ in which ϕ is decomposable. We say that ϕ is* saturated *w.r.t. $\mathcal{B}$ if for each $b_i \in \mathcal{B}$ such that $b_i \notin \tilde{n}$ we have that $b_i = x\sigma$ for some $x \in dom(\phi)$.*

Given a frame $\phi = \nu\tilde{n}.\{{}^{M_1}/_{x_1}, .., {}^{M_\ell}/_{x_\ell}\}$ and a base of names $\mathcal{B} = [b_1, .., b_k]$ in which ϕ is decomposable, we denote by $\overline{\phi}^{\mathcal{B}}$ the frame defined as follows:

$$\overline{\phi}^{\mathcal{B}} = \nu\tilde{n}.\{{}^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}, {}^{b_{i_1}}/_{y_1}, \ldots, {}^{b_{i_p}}/_{y_p}\}$$

where $b_{i_1}, \ldots, b_{i_p}$ is a subsequence of $\mathcal{B}$ such that $b_{i_j} \notin \tilde{n}$ and $b_{i_j} \neq x\sigma$ for every $x \in dom(\phi)$. The variables $y_1, \ldots y_p$ are fresh, which means that they do not appear in $dom(\phi)$. Note that the resulting frame $\overline{\phi}^{\mathcal{B}}$ is saturated w.r.t. $\mathcal{B}$.

Example 7. Let ϕ be the frame given in Example 3. Let $\mathcal{B} = [n_1, n_2, n_3]$. We have that ϕ is decomposable on $\mathcal{B}$ and also that ϕ is saturated w.r.t. $\mathcal{B}$. However, note that ϕ is not saturated w.r.t. $\mathcal{B}' = [n_1, n_2, n_3, n_4]$. We have $\overline{\phi}^{\mathcal{B}'} = \nu\tilde{n}.\{{}^{3n_1+2n_2+3n_3}/_{x_1}, {}^{n_2+3n_3}/_{x_2}, {}^{3n_2+n_3}/_{x_3}, {}^{3n_1+n_2+4n_3}/_{x_4}, {}^{n_4}/_{y_1}\}$ where the sequence $\tilde{n}$ denotes n_1, n_2, n_3.

5 Deduction

We show that solving a deduction problem can be reduced to solving a linear system of equations in the corresponding semiring.

Theorem 1. *Let E be a monoidal theory and $\mathcal{S}_{\mathsf{E}}$ be its associated semiring. Deduction in E is reducible in polynomial time to the following problem:*

Entries*: A matrix A over $\mathcal{S}_{\mathsf{E}}$ of size $\ell \times m$ and a vector b over $\mathcal{S}_{\mathsf{E}}$ of size ℓ*
Question*: Does there exists X (element of $\mathcal{S}_{\mathsf{E}}^{\ell}$) such that $X \cdot A = b$?*

Note that when $\mathcal{S}_{\mathsf{E}}$ is commutative, this problem is equivalent to the problem of deciding whether $A^{\mathsf{T}} \cdot Y = b^{\mathsf{T}}$, *i.e* whether b^{T} is in the image of A^{T} where M^{T} is the transpose of M. Before proving the reduction we need to establish that we can restrict our attention to saturated frames. Moreover, for such frames, it is sufficient to consider recipes without names, *i.e.* such that $fn(\zeta) = \emptyset$.

Lemma 3. *Let $\phi = \nu\tilde{n}.\sigma$ be a frame and M be a ground term. Let $\mathcal{B}$ be a base of names in which ϕ and M are decomposable. We have that $\phi \vdash_{\mathsf{E}} M$ if and only if $\overline{\phi}^{\mathcal{B}} \vdash_{\mathsf{E}} M$. Moreover when $\overline{\phi}^{\mathcal{B}} \vdash_{\mathsf{E}} M$ there exists a recipe ζ of M such that $fn(\zeta) = \emptyset$.*

Proof. Intuitively, the first point is due to the fact that we extend ϕ with some names which are deducible from ϕ. Hence, in term of deducible power ϕ and $\overline{\phi}^{\mathcal{B}}$ are equivalent. More formally, if $\phi \vdash_{\mathsf{E}} M$, we can assume that there exists ζ such that $fn(\zeta) \subseteq \mathcal{B} \smallsetminus \tilde{n}$ and $fv(\zeta) \subseteq dom(\phi)$. From such a ζ it is easy to compute ζ' by replacing any occurrence of a name in $\mathcal{B} \smallsetminus \tilde{n}$ by the corresponding variable in $dom(\overline{\phi}^{\mathcal{B}})$ which refers to this name. Note that since, we can always assume that $fn(\zeta) \subseteq \mathcal{B} \smallsetminus \tilde{n}$, we have that $fn(\zeta') = \emptyset$. The reverse transformation, *i.e.* the replacement of variables in $dom(\overline{\phi}^{\mathcal{B}}) \smallsetminus dom(\phi)$ by names refereed by these variables in $\overline{\phi}^{\mathcal{B}}$, allows us to conclude for the converse. □

Reduction. Let $\phi = \nu\tilde{n}.\sigma$ be a frame and M be a ground term. Let $\mathcal{B}$ be a base of names in which ϕ and M are decomposable. We will also assume w.l.o.g. that ϕ is saturated w.r.t. $\mathcal{B}$. Let $A = \psi_{\mathcal{B}}(\phi)$, matrix of size $\ell \times m$ over $\mathcal{S}_{\mathsf{E}}$, and $b = \psi_{\mathcal{B}}(M)$, vector of size m over $\mathcal{S}_{\mathsf{E}}$.

Proof. (of Theorem 1) The construction described above is such that $X \cdot A = b$ has a solution over $\mathcal{S}_{\mathsf{E}}$ if and only if $\phi \vdash_{\mathsf{E}} M$.
($\Rightarrow$) We know that there exists $X \in \mathcal{S}_{\mathsf{E}}^{\ell}$ such that $X \cdot A = b$. Consider the recipe $\zeta = \psi^{-1}_{dom(\phi)}(X)$. By construction, we have that $fn(\zeta) \cap \tilde{n} = \emptyset$. It remains to show that $\zeta\sigma =_{\mathsf{E}} M$. For this, we establish that $\psi_{\mathcal{B}}(\zeta\sigma) = \psi_{\mathcal{B}}(M)$. Thanks to Lemma 2, we have that $\psi_{\mathcal{B}}(\zeta\sigma) = \psi_{dom(\phi)}(\zeta) \cdot \psi_{\mathcal{B}}(\phi)$. Hence we deduce that $\psi_{\mathcal{B}}(\zeta\sigma) = X \cdot A = b = \psi_{\mathcal{B}}(M)$. Hence the result.
($\Leftarrow$) Assume that $\phi \vdash_{\mathsf{E}} M$. Thanks to Lemma 3 and by the fact that ϕ is saturated w.r.t. $\mathcal{B}$, we know that there exists $\zeta \in \mathcal{T}(\Sigma, dom(\phi))$ such that $\zeta\sigma =_{\mathsf{E}} M$. Let $Y = \psi_{dom(\phi)}(\zeta)$. It remains to establish that $Y \cdot A = b$. Since $\zeta\sigma =_{\mathsf{E}} M$, we have $\psi_{\mathcal{B}}(\zeta\sigma) = \psi_{\mathcal{B}}(M)$. By Lemma 2, we have $\psi_{dom(\phi)}(\zeta) \cdot \psi_{\mathcal{B}}(\phi) = \psi_{\mathcal{B}}(M)$, *i.e.* $Y \cdot A = b$ witnessing the fact that $X \cdot A = b$ has a solution over $\mathcal{S}_{\mathsf{E}}$. □

Example 8. Consider the theory ACUNh and the term $M = n_1 + \mathsf{h}(\mathsf{h}(n_1))$. Let $\phi = \nu n_1, n_2.\{^{n_1+\mathsf{h}(n_1)+\mathsf{h}(\mathsf{h}(n_1))}/_{x_1}, {}^{n_2+\mathsf{h}(\mathsf{h}(n_1))}/_{x_2}, {}^{\mathsf{h}(n_2)+\mathsf{h}(\mathsf{h}(n_1))}/_{x_3}\}$. We have that:

$$A = \begin{pmatrix} 1+\mathsf{h}+\mathsf{h}^2 & \mathsf{h}^2 & \mathsf{h}^2 \\ 0 & 1 & \mathsf{h} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 1+\mathsf{h}^2 \\ 0 \end{pmatrix}$$

The equation $X \cdot A = b$ has a solution over $\mathbb{Z}/2\mathbb{Z}[\mathsf{h}] : (1+\mathsf{h}, \mathsf{h}, 1)$. The term M is deducible from ϕ by using the recipe $x_1 + \mathsf{h}(x_1) + \mathsf{h}(x_2) + x_3$.

As a consequence, decidability/complexity results for deduction can be deduced from decidability/complexity results for solving linear system of equations (see Section 7).

6 Static equivalence

We show that deciding whether two frames are equivalent can be reduced to deciding whether two matrices satisfy the same set of equalities.

Theorem 2. *Let E be a monoidal theory and $\mathcal{S}_\mathsf{E}$ be its associated semiring. Static equivalence in E is reducible in polynomial time to the following problem:*

Entries*: Two matrices A_1 and A_2 over $\mathcal{S}_\mathsf{E}$ of size $\ell \times m$*
Question*: Does the following equality holds?*
$\{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_1 = Y \cdot A_1\} = \{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_2 = Y \cdot A_2\}$

Similarly to deduction, we first show that we can restrict our attention to saturated frames. Moreover, we show that it is sufficient to consider recipes, *i.e.* tests (M, N), without names.

Lemma 4. *Let $\phi_1 = \nu\tilde{n}.\sigma_1, \phi_2 = \nu\tilde{n}.\sigma_2$ and $\mathcal{B}$ be a base of names in which ϕ_1 and ϕ_2 are decomposable. We have that $\phi_1 \approx_\mathsf{E} \phi_2$ if and only if $\overline{\phi_1}^{\mathcal{B}} \approx_\mathsf{E} \overline{\phi_2}^{\mathcal{B}}$. Moreover, if $\overline{\phi_1}^{\mathcal{B}} \not\approx_\mathsf{E} \overline{\phi_2}^{\mathcal{B}}$ then there exist $M, N \in \mathcal{T}(\Sigma, dom(\overline{\phi_1}^{\mathcal{B}}))$ such that $(M =_\mathsf{E} N)\overline{\phi_1}^{\mathcal{B}} \not\Leftrightarrow (M =_\mathsf{E} N)\overline{\phi_2}^{\mathcal{B}}$.*

Proof. ($\Rightarrow$) Assume that $\phi_1 \approx_\mathsf{E} \phi_2$. We have that $\overline{\phi_1}^{\mathcal{B}} = \nu\tilde{n}.(\sigma_1 \cup \sigma_1^0)$ and $\overline{\phi_2}^{\mathcal{B}} = \nu\tilde{n}.(\sigma_2 \cup \sigma_2^0)$ for some substitutions σ_1^0 and σ_2^0 such that $\sigma_1^0 = \sigma_2^0$. Indeed, otherwise, we will obtain a test of the form $x = n_i$ with $x \in dom(\phi_1)$ and $n_i \in \mathcal{B} \smallsetminus \tilde{n}$ such that $(x =_\mathsf{E} n_i)\phi_1 \not\Leftrightarrow (x =_\mathsf{E} n_i)\phi_2$. Hence, we have that $dom(\overline{\phi_1}^{\mathcal{B}}) = dom(\overline{\phi_2}^{\mathcal{B}})$. Now, assume that $\overline{\phi_1}^{\mathcal{B}} \not\approx_\mathsf{E} \overline{\phi_2}^{\mathcal{B}}$, then

there exists a test (M, N) such that $(M =_{\mathsf{E}} N)\overline{\phi_1}^{\mathcal{B}}$ whereas $(M \neq_{\mathsf{E}} N)\overline{\phi_2}^{\mathcal{B}}$ (or the converse). Let $M' = M\sigma_1^0$ and $N' = N\sigma_1^0$. We have $(M' =_{\mathsf{E}} N')\phi_1$ whereas $(M' \neq_{\mathsf{E}} N')\phi_2$. Hence contradiction.
$(\Leftarrow)$ Assume that $\phi_1 \not\approx_{\mathsf{E}} \phi_2$. Let M, N be such that $(M =_{\mathsf{E}} N)\phi_1$ whereas $(M \neq_{\mathsf{E}} N)\phi_2$ (or the converse). It is clear that $\overline{\phi_1}^{\mathcal{B}} \not\approx_{\mathsf{E}} \overline{\phi_2}^{\mathcal{B}}$. Indeed, a witness of this fact is the test (M, N).

Lastly, if we have that $\phi_1 \not\approx_{\mathsf{E}} \phi_2$, then there exists a witness (M, N) such that $fn(M) \cup fn(N) \subseteq \mathcal{B} \smallsetminus \tilde{n}$. Hence, if we consider M' and N' the terms obtained from M and N by replacing any occurrence of a name in $\mathcal{B} \smallsetminus \tilde{n}$ by the corresponding variables in $dom(\overline{\phi_1}^{\mathcal{B}})$ which refers to this name, we easily conclude. □

Reduction. Let $\phi_1 = \nu\tilde{n}.\sigma_1$ and $\phi_2 = \nu\tilde{n}.\sigma_2$ be two frames having the same domain. Let $\mathcal{B}$ be a base of names in which the two frames are decomposable. We assume w.l.o.g. that ϕ_1 and ϕ_2 are saturated w.r.t. $\mathcal{B}$. Let $m = |\mathcal{B}|$. Let $A_1 = \psi_{\mathcal{B}}(\phi_1)$ and $A_2 = \psi_{\mathcal{B}}(\phi_2)$, two matrices of size $\ell \times m$, over $\mathcal{S}_{\mathsf{E}}$.

Proof. (of Theorem 2) The construction above is such that $\phi_1 \approx_{\mathsf{E}} \phi_2$ iff $\{(X, Y) \in \mathcal{S}_{\mathsf{E}}^{\ell} \times \mathcal{S}_{\mathsf{E}}^{\ell} \mid X \cdot A_1 = Y \cdot A_1\} = \{(X, Y) \in \mathcal{S}_{\mathsf{E}}^{\ell} \times \mathcal{S}_{\mathsf{E}}^{\ell} \mid X \cdot A_2 = Y \cdot A_2\}$.
$(\Rightarrow)$ Assume by contradiction that there exists a pair (X_M, X_N) such that $X_M \cdot A_1 = X_N \cdot A_1$ and $X_M \cdot A_2 \neq X_N \cdot A_2$ (or the converse). Let $M = \psi^{-1}_{dom(\phi_1)}(X_M)$ and $N = \psi^{-1}_{dom(\phi_1)}(X_N)$. We have that

- $(M =_{\mathsf{E}} N)\phi_1$. It is sufficient to show that $\psi_{\mathcal{B}}(M\sigma_1) = \psi_{\mathcal{B}}(N\sigma_1)$, *i.e.* $\psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1) = \psi_{dom(\phi_1)}(N) \cdot \psi_{\mathcal{B}}(\phi_1)$ thanks to Lemma 2. Now to conclude, it is sufficient to notice that we have $X_M = \psi_{dom(\phi_1)}(M)$, $X_N = \psi_{dom(\phi_1)}(N)$ and $A_1 = \psi_{\mathcal{B}}(\phi_1)$ and to rely on the hypothesis.
- $(M \neq_{\mathsf{E}} N)\phi_2$ can be shown similarly.

$(\Leftarrow)$ Assume that $\phi_1 \not\approx_{\mathsf{E}} \phi_2$. We have that there exists a test (M, N) such that $(M =_{\mathsf{E}} N)\phi_1$ and $(M \neq_{\mathsf{E}} N)\phi_2$ (or the converse). Thanks to Lemma 4 and the fact that the frames are saturated, we can assume that $M, N \in \mathcal{T}(\Sigma, dom(\phi_1))$. Let $X_M = \psi_{dom(\phi_1)}(M)$ and $X_N = \psi_{dom(\phi_1)}(N)$. We have that

- $X_M \cdot A_1 = X_N \cdot A_1$. Indeed, we have that $M\sigma_1 =_{\mathsf{E}} N\sigma_1$. Hence $\psi_{\mathcal{B}}(M\sigma_1) = \psi_{\mathcal{B}}(N\sigma_1)$. By Lemma 2, we have $\psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1) = \psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1)$, *i.e.* $X_M \cdot A_1 = X_N \cdot A_1$.
- $X \cdot A_2 \neq Y \cdot A_2$ can be established in a similar way. □

Going further. Thanks to Theorem 2, we give a way to decide static equivalence in monoidal equational theories provided we can decide whether two sets of linear equations over $\mathcal{S}_{\mathsf{E}}$ have the same set of solutions. Actually, when $\mathcal{S}_{\mathsf{E}}$ is a ring or when we can extend the semiring $\mathcal{S}_{\mathsf{E}}$ into a ring $\mathcal{R}_{\mathsf{E}}$, the static equivalence problem is equivalent to the problem of deciding whether the following equality holds.

$$\{Z \in \mathcal{R}_{\mathsf{E}}^{\ell} \mid Z \cdot A_1 = 0\} = \{Z \in \mathcal{R}_{\mathsf{E}}^{\ell} \mid Z \cdot A_2 = 0\}$$

When $\mathcal{R}_{\mathsf{E}}$ is a commutative ring, it is equivalent to deciding whether $\mathsf{Ker}(A_1) = \mathsf{Ker}(A_2)$, where $\mathsf{Ker}(M)$ denotes the kernel of the matrices M, *i.e.* the set $\{X \mid M \cdot X = 0\}$.

The ring associated to a given monoidal theory E, denoted by $\mathcal{R}_{\mathsf{E}}$, is equal to $\mathcal{S}_{\mathsf{E}}$ when E is a group theory. Otherwise, it might be possible to extend the equational theory E with a new unary symbol $-$ and the law $x + -(x) = 0$ in order to obtain a theory E' that is consistent with E, *i.e.* for all $u, v \in \mathcal{S}_{\mathsf{E}}$ such that $u =_{\mathsf{E}'} v$, we have also that $u =_{\mathsf{E}} v$. In such a case, the ring $\mathcal{R}_{\mathsf{E}}$ is the semiring $\mathcal{S}_{\mathsf{E}'}$ associated to E' as explained in Section 4.1.

Example 9. We have seen that the semiring associated to AG is isomorphic to $\mathbb{Z}$ which is a commutative ring. Hence, we have that $\mathcal{R}_{\mathsf{E}}$ is isomorphic to $\mathbb{Z}$. The associated semiring to the monoidal equational theory ACU is isomorphic to $\mathbb{N}$ whereas its associated ring is $\mathbb{Z}$.

Note that the transformation described above does not allow us to associate a ring to any semiring. For instance, if we consider the theory ACUI and the theory E' obtained by the transformation described above, we have that $0 =_{\mathsf{E}'} (\mathbf{1} + \mathbf{1}) + -(\mathbf{1}) =_{\mathsf{E}'} \mathbf{1} + (\mathbf{1} + -(\mathbf{1})) =_{\mathsf{E}'} \mathbf{1}$ whereas this equality does not hold in ACUI.

7 Applications and Discussion

In this section we show that several interesting monoidal equational induce a ring or a semiring for which solving linear systems or checking for equalities of sets of solutions of linear systems are decidable. Note that any of these decidability results for deduction and static equivalence can be combined with any existing ones provided the signatures of the equational theories are disjoint [3]. For example, let E be a monoidal equational theory for which deduction and static equivalence are decidable (*e.g.* ACU, ACUNh, ...) then deduction and static equivalence are also decidable for the theory $\mathsf{E}_{\mathsf{enc}} \cup \mathsf{E}$ where $\mathsf{E}_{\mathsf{enc}}$ is defined by the following equations:

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x, \quad \mathsf{proj}_1(\langle x, y \rangle) = x \ \text{ and } \ \mathsf{proj}_2(\langle x, y \rangle) = y.$$

Theory ACU. This equational theory is the simplest monoidal theory. The semiring corresponding to this theory is $\mathbb{N}$ whereas its associated ring is $\mathbb{Z}$. This equational theory has been particularly studied. Since the problem of solving linear equations over $\mathbb{N}$ is strongly NP-complete, we obtain that deduction is a NP-complete problem. The problem of static equivalence for this theory has been shown decidable in [1]. Actually thanks to the algebraic characterization given in this paper, this problem can be solved in polynomial time [18].

At first sight, it might seem surprising since it has been shown [1] that deduction in a given theory E can be reduced in polynomial time to static equivalence in E. However, this reduction required the presence of a free function symbol and such a function symbol is not available in the theory ACU. Hence, the polynomial reduction provided in [1] does not apply in this setting.

Theories ACUI and ACUN (Exclusive Or). The semirings corresponding to these equational theories are respectively the Boolean semiring $\mathbb{B}$, which is finite, and the finite field $\mathbb{Z}/2\mathbb{Z}$. The theory ACUN has already been studied in terms of deduction [8, 7] and static equivalence [1]. Deduction and static equivalence are both decidable in polynomial time. As far as we know the theory ACUI has only been studied in term of deduction [10]. Actually, since its associated semiring is finite, we easily deduce that deduction and static equivalence are decidable.

Theory AG (Abelian Groups). The semiring associated to this equational theory is in fact a ring, namely the ring $\mathbb{Z}$ of all integers. There exist several algorithms to compute solutions of linear equations over $\mathbb{Z}$ and to compute a base of the set of solutions (see for instance [18]). Hence, we easily deduce that both problems are decidable in PTIME. Deduction for this theory has already been studied in [8] and [6].

Theories ACUh, ACUNh and AGh. The semiring associated to ACUh is $\mathbb{N}[\mathsf{h}]$, the semiring of polynomial in one indeterminate over $\mathbb{N}$ whereas the ring associated to ACUh is $\mathbb{Z}[\mathsf{h}]$. For the theory ACUNh (resp. AGh) the associated semiring is $\mathbb{Z}/2\mathbb{Z}[\mathsf{h}]$ (resp. $Z[\mathsf{h}]$). Deduction for these three equational theories have already been studied in [11, 9]. However, results obtained on static equivalence are new.

1. ACUh and AGh: Deciding static equivalence for these both theories is reducible to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$

where A and B are matrices built over $\mathsf{N}[\mathsf{h}]$ in the case of ACUh and $\mathsf{Z}[\mathsf{h}]$ in the case of AGh. This problem has been solved by F. Baader to obtain a unification algorithm for the theory AGh (see [4]). This is done by the help of Gröbner Base methods in a more general settings. Actually, he provides an algorithm even in the case of several commutating homomorphisms.

2. ACUNh: Deciding static equivalence in ACUNh is reducible to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$ where A and B are matrices built over $\mathsf{Z}/2\mathsf{Z}[\mathsf{h}]$. This is achieved in [12] by using an automata-theoretic approach.

Theory ACUIh. The semiring associated to ACUIh is $\mathbb{B}[\mathsf{h}]$. Deduction for this theory has never been studied but is clearly decidable. Indeed, to find a solution to $A \cdot X = b$, it is easy to see that each component of a solution to $A \cdot X = b$ has a degree smaller than the degree of b. Hence, the question of deciding whether there exists X such that $A \cdot X = b$ can be reduced to solving a system of linear equations over $\mathbb{B}$. Theorem 2 does not help us to provide an algorithm to solve static equivalence. Note also that we can not reduced the problem to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$ since, as for ACUI, we are not able to associate a ring to this theory.

Theory E	$\mathcal{S}_\mathsf{E}$	$\mathcal{R}_\mathsf{E}$	Deduction	Static Equivalence
ACU	$\mathbb{N}$	$\mathbb{Z}$	NP-complete	decidable [1], PTIME (*new*)
ACUI	$\mathbb{B}$	–	decidable [10]	decidable (*new*)
ACUN	$\mathbb{Z}/2\mathbb{Z}$		PTIME [7]	decidable [1], PTIME (*new*)
AG	$\mathbb{Z}$		PTIME [6]	PTIME (*new*)
ACUh	$\mathbb{N}[\mathsf{h}]$	$\mathbb{Z}[\mathsf{h}]$	NP-complete [11]	decidable (*new*)
ACUIh	$\mathbb{B}[\mathsf{h}]$	–	decidable (*new*)	?
ACUNh	$\mathbb{Z}/2\mathbb{Z}[\mathsf{h}]$		PTIME [9]	decidable (*new*)
AGh	$\mathbb{Z}[\mathsf{h}]$		PTIME [9]	decidable (*new*)
$\mathsf{AGh}_1 \ldots \mathsf{h}_n$	$\mathbb{Z}[\mathsf{h}_1, \ldots, \mathsf{h}_n]$		decidable (*new*)	decidable (*new*)

8 Conclusion

We have proposed a general schema for deciding deduction and static equivalence algorithms. This schema has to be filled with procedures for linear equations in order to yield complete algorithms. Such algorithms strongly depend on the structure of the semiring. In this paper, we have mentioned and used several existing results of Algebra. But Algebra can still provide useful techniques that allow us to deduce some new results. Moreover, efficient existing tools for solving algebraic problems can also be used to implement our algorithms.

Acknowledgment. We wish to thank Daniel Lazard for fruitful discussions.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, November 2006.
2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, London (UK), 2001. ACM.
3. M. Arnaud, V. Cortier, and S. Delaune. Combining algorithms for deciding knowledge in security protocols. In *Proc. of the 6th International Symposium on Frontiers of Combining Systems (FroCoS'07)*, LNAI, Liverpool, UK, 2007. Springer. To appear.
4. F. Baader. Unification in commutative theories, Hilbert's basis theorem, and Gröbner bases. *Journal of the ACM*, 40(3):477–503, 1993.
5. F. Baader and W. Nutt. Combination problems for commutative/ monoidal theories or How algebra can help in equational unification. *Applicable Algebra Engineering Communication and Computing*, 7(4):309–337, 1996.
6. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and product in exponents. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'03)*, volume 2914 of *LNCS*, pages 124–135, Mumbai (India), 2003. Springer-Verlag.
7. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 261–270, Ottawa (Canada), 2003. IEEE Comp. Soc. Press.
8. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 271–280, Ottawa (Canada), 2003. IEEE Comp. Soc. Press.
9. S. Delaune. Easy intruder deduction problems with homomorphisms. *Information Processing Letters*, 97(6):213–218, Mar. 2006.
10. S. Delaune, P. Lafourcade, D. Lugiez, and R. Treinen. Symbolic protocol analysis for monoidal equational theories. Research Report LSV-06-17, Laboratoire Spécification et Vérification, ENS Cachan, France, Nov. 2006. 47 pages.
11. P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 308–322, Nara (Japan), 2005. Springer.

12. P. Lafourcade, D. Lugiez, and R. Treinen. ACUNh: Unification and disunification using automata theory. In *Proc. 20th Int. Workshop on Unification (UNIF'06)*, pages 6–20, Seattle, Washington, USA, Aug. 2006.
13. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, Berlin (Germany), 1996. Springer-Verlag.
14. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01).* ACM Press, 2001.
15. W. Nutt. Unification in monoidal theories. In *Proc. 10th Int. Conference on Automated Deduction, (CADE'90)*, volume 449 of *LNCS*, pages 618–632, Kaiserslautern (Germany), 1990. Springer.
16. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
17. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 1-3(299):451–475, 2003.
18. A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

The Meaning of a Cryptographic Message via Hypothetical Knowledge and Provability

Simon Kramer

Ecole Polytechnique Fédérale de Lausanne (EPFL)
simon.kramer@a3.epfl.ch

Abstract. We propose a denotational definition for the meaning of a cryptographic message and, based on it, an equational definition for the context-sensitivity of that meaning, both via hypothetical knowledge and provability. As a result, we obtain a formalisation of the first of Abadi and Needham's principles for prudent engineering practice for — and a tentative denotational semantics of — cryptographic protocols. Further, we define the information content of a cryptographic message and, based on it, the information content of a cryptographic protocol. Furthermore, we show that protocol agents can be naturally conceived as evolving Scott information systems. Our method is applicable to arbitrary interactive programs (multi-agent systems), but is made concrete here on the example of cryptographic protocols.

Keywords cryptographic protocols, denotational semantics, modal logics, multi-agent systems

1 Introduction

Our motivation for defining the meaning of a cryptographic message comes from Abadi and Needham's Principle 1 for designing cryptographic protocols [AN96]. The principle says:

> Every message should say what it means: the interpretation of the message should depend only on its contents. [. . .]

With a *clin d'œil* and thus for the nonce, one-eyed, we observe that the principle is both self-denying and not self-denying and thus paradoxical, and that this fact can be proven by applying the principle to itself. Here is an informal proof, by contradiction:

> Assume that the principle is not self-denying. Apply it to itself by particularising it with itself. (The principle is itself a message[1] and employs universal quantification over messages.) The

[1] Observe that the principle speaks about messages *tout court* rather than *cryptographic* messages. If the principle is to speak about cryptographic messages, then it must depend on a context (which [AN96] of course constitutes) that properly frames it (which [AN96] of course does). Yet, that very dependence the principle denies.

> message of the principle does not say what it means: the interpretation of the message does not depend only on its contents. (The principle does not say what message meaning means.) Hence, the principle is self-denying. Contradiction.
>
> Deduce that the principle is self-denying. Yet, by this very fact the principle is *not* self-denying. (Every message, as the example of the principle demonstrates, should really say what it means.) Conclude that the principle is paradoxical.

This recreational proof is paradoxical in the (double) sense that it demonstrates the paradoxical nature of Principle 1 and paradoxically the importance of the subject matter of the principle. That is, *explicitness* and *context-sensitivity* of meaning. Our task shall be to define, in a unique sense, the meaning[2] of a cryptographic message relative to an execution state and communicating agent.

We stress that our definition of message meaning is sensitive to a context that possibly contains *several* concurrently-running protocols: execution states are ordered pairs of an execution history and a process term, and process terms possibly consist of parallel compositions of *several* protocols.

Related work The most relevant work (though not addressing the problem of logical omniscience) related to ours is [PR03], where, according to the authors, the meaning of a message is given in terms of how it affects the knowledge of the agents involved in the communication. This idea is spiritually close to ours (and is given a very nice philosophical treatment by the authors), but has incarnated in a very different body (of knowledge, so to say) as will become clear in the sequel. The authors define the denotation of a message at a state and w.r.t. an agent as a so-called *view transformer*. And a view transformer is a set of ordered pairs (α, β) of propositions α and β such that "if knowledge of α is part of the view that [agent] i has of the global system state before the communication event, then knowledge of β is part of its view after the communication."

Another relevant work related to ours is [Gro92]. There, the so-called (1) *objective semantics* of a message is defined to be the set of all points, states in our terminology, where the message was sent; and (2) *KS-semantics* of a message is defined to be the set of pairs of a point and an agent such that that agent sends the message at that point. The problem with this approach is, according to the authors, that it is not yet suitable for cryptography.

The relevant commonality between [PR03], [Gro92], and our approach is *that* all approaches employ epistemic logic as a means to defining message

[2] in the sense of Frege's *sense* (a message makes to an agent) as opposed to *reference* (to a bit-string)

meaning. The difference is *how* each approach does so, as we shall see now with the presentation of our approach.

2 Prerequisites

We presuppose a set $\mathcal{M}$ of cryptographic messages (denoted M, M') and a set $\mathcal{A}_{\mathtt{Eve}} \subseteq \mathcal{M}$ of names (denoted a, b) for legitimate agents and for the archetypical illegitimate agent — the adversary (Eve). The exact form of cryptographic messages is irrelevant to the present exposition (this is an advantage of our approach). However, it is helpful to think of $\mathcal{M}$ as containing, among others, encrypted messages of the form $\{\!|M|\!\}_k$, i.e., plaintexts M encrypted under (symmetric) keys $k \in \mathcal{K} \subseteq \mathcal{M}$.

The adversary can be thought of as being in full control of the network, and may, of course, also be an insider, i.e., an agent with all the rights of a legitimate agent.

We further presuppose a set $\mathcal{F}$ of closed logical formulae (denoted ϕ, ϕ'), i.e., propositions, whose truth is established on the states $\mathfrak{s} = (\mathfrak{h}, P)$ of history $\mathfrak{h} \in \mathcal{H}$ (of past protocol events, e.g., message sending/receiving) and code $P \in \mathcal{P}$ of the considered protocol. The exact form of histories and protocol code is irrelevant to the present exposition (again, this is an advantage of our approach). The logical language we have in mind is the one of CPL [Kra06,Kraar], but there is no need of going into full detail for the present exposition. The relevant syntactic constructions are the following:

1. a binary relational symbol k expressing an agent's individual knowledge (or knowledge *de re*), i.e., an agent's capability to synthesise messages from previously analysed messages that the agent has sent and received during a protocol history. For encrypted messages, we have (and this is about all we need for the present exposition) $\models (a \;\mathsf{k}\; \{\!|M|\!\}_k \wedge a \;\mathsf{k}\; k) \rightarrow a \;\mathsf{k}\; M$.
2. a unary epistemic modality $\mathsf{K}_a(\phi)$ expressing an agent's a propositional knowledge (or knowledge *de dicto*), i.e., the agent's capability to establish the truth of a proposition ϕ. Its semantics is roughly captured by the modal system **S5** [FHMV95], and is a refinement of [AT91] in the sense that epistemic necessitation is adapted to the cryptographic setting so that if $\models \phi$ then $\models a \;\mathsf{k}\; K \rightarrow \mathsf{K}_a(\phi)$, where K is a tuple of the keys occurring in ϕ. This adaptation is necessary and sufficient to solve the problem of logical omniscience in a cryptographic setting (cf. [CD05] and [Kra06,Kraar] for further discussion).
3. a binary spatial modality $\phi' \rhd \phi$ expressing the truth of ϕ on the provision of the (hypothetical) truth of ϕ'. Its semantics for a considered state $\mathfrak{s}$ is that for all $\mathfrak{s}'$, if $\mathfrak{s}' \models \phi'$ then $\mathfrak{s}' \circ \mathfrak{s} \models \phi$, where $\mathfrak{s}'$ designates the state of

a hypothetical protocol environment, and $\circ$ parallel composition (the exact form of the composition is irrelevant to the present exposition).

Notice that this adjunction (or spatial conditional) is *relevant*, i.e., it is not truth-functional [Dam89]. It is also monotonic w.r.t. positive antecedents, e.g., $\models a \mathsf{k} M \rhd a \mathsf{k} M$, but $\not\models \neg a \mathsf{k} M \rhd \neg a \mathsf{k} M$ due to the possibility for a of learning new information from the adjunction.

4. a unary Gödel-style provability modality $\mathsf{P}_a(\phi)$ expressing an agent's a capability to *prove* that ϕ is true. Remarkably, this modality is *syntactically* definable within CPL (cf. [Kra06,Kraar]). The modality complies with the modal system **S4**, with provability necessitation adapted to the cryptographic setting so that if $\models \phi$ then $\models a \mathsf{k} K \rightarrow \mathsf{P}_a(\phi)$.

   (We refer the interested reader to [AN05] for a related approach to provability, and to [Kra06,Kraar] for further discussion.)

Finally, we recall logical consequence in CPL: $\phi \Rightarrow \phi'$:iff for all $\mathfrak{s}$, if $\mathfrak{s} \models \phi$ then $\mathfrak{s} \models \phi'$.

3 Message meaning

Definition 1 (The meaning of a cryptographic message). *The (communicable) meaning of a cryptographic message $M \in \mathcal{M}$ w.r.t. an agent $a \in \mathcal{A}_{\mathsf{Eve}}$ and a protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$ shall be the set of equivalence classes $[\phi]$ of those propositions $\phi \in \mathcal{F}$ whose truth a* would *know (resp., be able to prove) if a* knew *M in $\mathfrak{s}$. (Notice the conditional mode!) Formally,*

$$[\![M]\!]^{\mathfrak{s}}_{a} := \{\, [\phi] \mid \phi \in \mathcal{F} \text{ and } \mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{K}_a(\phi) \,\}, \text{ and}$$
$$[\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} := \{\, [\phi] \mid \phi \in \mathcal{F} \text{ and } \mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{P}_a(\phi) \,\} \text{ respectively.}$$

Message meaning is defined as a set of *equivalence classes* of propositions rather than as a set of propositions because we are concerned with just *what* a message means rather than with the manifold *how* it may mean it.

Message meaning is *communicable* when defined in terms of provability because provability requires the capability to produce an actual proof (i.e., a message of a certain cryptographic form), which, by definition, *is* communicable [Kra06,Kraar].

Theorem 1. *Message meaning is a distributive proper filter (and thus a proper sub-lattice and a topped $\bigcap$-structure) w.r.t. the Boolean lattice $\langle \mathcal{F}/_{\equiv}, \leq \rangle$, i.e., it is (1) a non-empty proper sub-set of $\mathcal{F}/_{\equiv}$, (2) closed under meet and partial* *ordering (and thus is directed), and (3) distributive. Formally, let*[3]

$$\overline{[\phi]} := [\neg\phi] \quad \textit{(complement)}$$

$$[\phi] \wedge [\phi'] := [\phi \wedge \phi'] \quad \textit{(meet)} \qquad [\phi] \vee [\phi'] := [\phi \vee \phi'] \quad \textit{(join)}$$

$$[\phi] \leq [\phi'] \quad \textit{:iff} \quad \phi \Rightarrow \phi' \textit{ and } \mathrm{keys}(\phi') \subseteq \mathrm{keys}(\phi) \quad \textit{(partial ordering)}$$

$$\phi \equiv \phi' \quad \textit{:iff} \quad [\phi] \leq [\phi'] \textit{ and } [\phi'] \leq [\phi] \quad \textit{(congruence w.r.t. meet and join)}$$

where keys(ϕ) *designates the set of all key constants occurring in* ϕ*. Then,*

1. $\emptyset \neq [\![M]\!]^{\mathfrak{s}}_{a} \subset \mathcal{F}/_{\equiv}$
2. *(a) if* $[\phi], [\phi'] \in [\![M]\!]^{\mathfrak{s}}_{a}$ *then* $[\phi] \wedge [\phi'] \in [\![M]\!]^{\mathfrak{s}}_{a}$
   (b) if $[\phi] \in [\![M]\!]^{\mathfrak{s}}_{a}$ *and* $[\phi'] \in \mathcal{F}/_{\equiv}$ *and* $\phi \leq \phi'$ *then* $[\phi'] \in [\![M]\!]^{\mathfrak{s}}_{a}$
3. *if* $[\phi] \vee [\phi'] \in [\![M]\!]^{\mathfrak{s}}_{a}$ *and* $[\phi] \vee [\phi''] \in [\![M]\!]^{\mathfrak{s}}_{a}$ *then* $[\phi] \vee ([\phi'] \wedge [\phi'']) \in [\![M]\!]^{\mathfrak{s}}_{a}$.

Proof. see Appendix A

Filters represent deductively closed (cf. Condition 2.(b)), consistent (i.e., $[\bot] \notin [\![M]\!]^{\mathfrak{s}}_{a}$, otherwise $[\![M]\!]^{\mathfrak{s}}_{a} = \mathcal{F}/_{\equiv}$ by Condition 2.(b), which would violate Condition 1.), but possibly incomplete (unless they are maximal) theories.

Theorem 2. *Communicable message meaning is a distributive proper filter w.r.t. the Boolean lattice* $\langle \mathcal{F}/_{\equiv}, \leq \rangle$.

Proof. see Appendix A

Proposition 1. *Communicable message meaning is order-embedded strictly within message meaning. Formally,* $[\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} \subset [\![M]\!]^{\mathfrak{s}}_{a}$ *and* $[\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} \hookrightarrow [\![M]\!]^{\mathfrak{s}}_{a}$.

Proof. see Appendix A

Definition 2 (Context-sensitivity of message meaning). *A cryptographic message* $M \in \mathcal{M}$ *is* context-sensitive *:iff there are* $a, b \in \mathcal{A}_{\mathsf{Eve}}$ *and* $\mathfrak{s}, \mathfrak{s}' \in \mathcal{H} \times \mathcal{P}$ *s.t.* $[\![M]\!]^{\mathfrak{s}}_{a} \neq [\![M]\!]^{\mathfrak{s}'}_{b}$*. A cryptographic message* $M \in \mathcal{M}$ *that is not context-sensitive is* context-free.

Note that our notion of context-sensitivity is *semantic* as opposed to the classical notion of formal language theory, which is syntactic.

Formalisation 1 (Abadi and Needham's Principle 1)

[3] this is a refinement of the classical Lindenbaum-Tarski-algebra construction: the partial ordering is *not* mere logical consequence ($\Rightarrow$); thus the resulting equivalence ($\equiv$) is finer than logical equivalence ($\Leftrightarrow$)

"Every message should say what it means: the interpretation of the message should depend only on its contents. [...]" [AN96]

Every cryptographic message should be context-free *(cf. Definition 2).* ⌟

Examples of context-sensitive cryptographic messages abound. Perhaps the starkest example is the one of sending a bare nonce as the opening of a cryptographic protocol. See [BM03] for several *different*, and reportedly flawed protocols with such *identical* openings. Another way of creating context-sensitive cryptographic messages is the use of *indexicals* (i.e., phrases in natural language that refer to past or future messages, or to extra-protocol out-of-band communication such as personal contact and trusted couriers) in plaintexts. An advantage of our definition of the context-sensitivity of a cryptographic message is that the definition is, being equational, *indifferent to* the many ways of *how* context-sensitivity is created. It[4] only cares about *that* context-sensitivity is created.

Definition 3 (The information content of a cryptographic message). *The information content (in the sense of Kolmogorov-complexity [LV97]) of a cryptographic message $M \in \mathcal{M}$ to agent $a \in \mathcal{A}_{\mathsf{Eve}}$, written* $\mathrm{K}_a(M)$*, is defined to be the smallest*[5] *state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$ s.t. $\mathfrak{s} \models a \mathsf{k} M$.*

Note that [PR03] also define the information content of a message, but their definition is, as opposed to ours, in the sense of Shannon.

4 Protocol meaning

Slogan 1 *The purpose of a cryptographic protocol is to interactively compute, via message passing*[6]*, knowledge of the truth of desired — and, dually, knowledge of the falsehood of undesired — cryptographic states of affairs. [Kra06,Kraar]*

In other words, cryptographic protocols aim at inducing propositional knowledge, i.e., knowledge of cryptographic states of affairs expressed as propositions, by means of individual knowledge, i.e., knowledge of messages (values). Values are only the means — not the ends — of cryptographic[7] computation.

Slogan 2 *In cryptography, individual knowledge is the key to propositional knowledge.*

[4] "It", the footnote marker in this line, and "this" are three examples of indexicals.

[5] bear in mind that our protocol states are just finite strings of symbols, each string containing a process term (the program) as a substring

[6] rather than shared memory

[7] and possibly of interactive computation [GSW06] in general

Definition 4 (The meaning of a cryptographic protocol). *The meaning (or* denotational semantics*) $[\![\mathfrak{s}]\!]^*_a$ of a cryptographic protocol $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$ w.r.t. to an agent $a \in \mathcal{A}_{\mathsf{Eve}}$ shall be the (directed) union of the meanings w.r.t. a of all those messages that a comes to know during protocol execution. Formally, let $T(\cdot) := \cdot \cup \{\bigwedge \mathrm{Min}(\cdot)\}$ designate a template for the meet-completion of $\cdot$. Then,*

$$[\![\mathfrak{s}]\!]_a := T\left(\bigcup_{\substack{M \in \mathcal{M} \\ \mathfrak{s} \models a \,\mathsf{k}\, M}} [\![M]\!]^{\mathfrak{s}}_a\right) \quad [\![\mathfrak{s}]\!]^n_a := T\left(\bigcup_{\substack{\mathfrak{s}' \in \mathcal{H}\times\mathcal{P} \\ \mathfrak{s} \longrightarrow^n \mathfrak{s}'}} [\![\mathfrak{s}']\!]_a\right) \quad [\![\mathfrak{s}]\!]^*_a := T\left(\bigcup_{\substack{\mathfrak{s}' \in \mathcal{H}\times\mathcal{P} \\ \mathfrak{s} \longrightarrow^* \mathfrak{s}'}} [\![\mathfrak{s}']\!]_a\right)$$

$$[\![\mathfrak{s}]\!] := \biguplus_{a \in \mathcal{A}_{\mathsf{Eve}}} [\![\mathfrak{s}]\!]_a \qquad [\![\mathfrak{s}]\!]^n := \biguplus_{a \in \mathcal{A}_{\mathsf{Eve}}} [\![\mathfrak{s}]\!]^n_a \qquad [\![\mathfrak{s}]\!]^* := \biguplus_{a \in \mathcal{A}_{\mathsf{Eve}}} [\![\mathfrak{s}]\!]^*_a$$

where (non-deterministic) protocol execution is supposed to be modelled by an appropriate relation $\longrightarrow \subseteq (\mathcal{H} \times \mathcal{P})^2$ (e.g., [BKN06,BGK06]).

The meet-completion ensures that each collective meaning has again a least element, by taking the meet of the set of minimal elements in the union of individual meanings. (The *least* element in each individual meaning becomes a *minimal* element in the union of individual meanings.)

Note that our definition of the meaning (or denotational semantics) of a cryptographic protocol

1. is defined in terms of
   (a) the meaning of cryptographic messages
   (b) the *operational* semantics $\longrightarrow$ of that protocol, which is a very natural, but nevertheless, a novel idea. It is natural to define the *what* (the denotation) in terms of the *how* (the operations), rather than the other way round or defining them independently of each other.
2. has the advantage of being *syntax-independent*. The denotational semantics does not require inductive definition on the structure of cryptographic protocol terms $P \in \mathcal{P}$ (the operational typically does [BKN06,BGK06]).

Theorem 3. $[\![\mathfrak{s}]\!]_a = [\![\mathfrak{s}']\!]_a$ *iff [for all $\phi \in \mathcal{F}$, $\mathfrak{s} \models \mathsf{K}_a(\phi)$ iff $\mathfrak{s}' \models \mathsf{K}_a(\phi)$]*

Proof. see Appendix A

In other words, two protocol states (worlds) that make equal sense to a protocol agent are *epistemically indistinguishable* (w.r.t. to language $\mathcal{F}$) to that agent. In Wittgenstein's words: "The limits of my language mean the limits of my world." [Wit75, Paragraph 5.6].

Theorem 4.

1. $[\![\mathfrak{s}]\!]^n_a \hookrightarrow [\![\mathfrak{s}]\!]^{n+1}_a$ *is order-continuous*[8]

[8] order-continuity guarantees the existence of fixpoints

2. *$[\![\mathfrak{s}]\!]_a$, $[\![\mathfrak{s}]\!]_a^n$, and $[\![\mathfrak{s}]\!]_a^*$ are topped algebraic $\bigcap$-structures (thus algebraic lattices, thus complete lattices, and thus complete partial orders [DP02])*
3. *$[\![\mathfrak{s}]\!]$, $[\![\mathfrak{s}]\!]^n$, and $[\![\mathfrak{s}]\!]^*$ are pre-CPOs*

Proof. see Appendix A

An algebraic $\bigcap$-structure $\mathfrak{S}$ can be presented as a *Scott information system* $\mathbf{IS}(\mathfrak{S})$ with "the idea of identifying an object with a set of propositions true of it and adequate to define it. These propositions are to be thought of as tokens, each bearing a finite amount of information." [DP02]:

$$\mathbf{IS}(\mathfrak{S}) := \langle \bigcup \mathfrak{S}, \{ \Gamma \mid \text{there is } S \in \mathfrak{S} \text{ s.t. } \Gamma \Subset S \}, \vdash \rangle$$

where $\Gamma \vdash \phi$:iff $\phi \in \bigcap\{ S \mid S \in \mathfrak{S} \text{ and } \Gamma \Subset S \}$ and $\Subset$ designates finitary set-inclusion. In other words, protocol meaning induces a Scott information system for each protocol agent at each protocol state. And protocol execution induces the continuous (cf. Theorem 4.1) evolution of those information systems in time.

The definition of a denotational semantics for cryptographic protocols is not only a useful exercise of conceptual clarification, but is also useful for the actual engineering (verification, refinement) of safe cryptographic protocols:

Protocol invariant A formula $\phi \in \mathcal{F}$ is a *subjective* protocol invariant w.r.t. agent $a \in \mathcal{A}_{\text{Eve}}$ and initial protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$:iff for all $n \in \mathbb{N}$, $\phi \in [\![\mathfrak{s}]\!]_a^n$. A formula $\phi \in \mathcal{F}$ is a *universal* protocol invariant w.r.t. initial protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$:iff for all $a \in \mathcal{A}_{\text{Eve}}$, ϕ is a subjective protocol invariant w.r.t. a and $\mathfrak{s}$.

Protocol safety A protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$ is *subjectively safe* to agent $a \in \mathcal{A}$:iff the negation of every cryptographic state of affairs undesirable to a is a subjective protocol invariant w.r.t. a. A protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$ is *universally safe* :iff the negation of every cryptographic state of affairs undesirable to some $a \in \mathcal{A}$ is a subjective protocol invariant w.r.t. a.

Protocol refinement A protocol state $\mathfrak{s}' \in \mathcal{H} \times \mathcal{P}$ *refines* protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$, written $\mathfrak{s} \leq \mathfrak{s}'$, :iff the meaning of $\mathfrak{s}$ is included in the meaning of $\mathfrak{s}'$. Formally,

$$\mathfrak{s} \leq \mathfrak{s}' \quad \text{:iff} \quad [\![\mathfrak{s}]\!]^* \subseteq [\![\mathfrak{s}']\!]^*.$$

It is well-known that refinement orderings on a set of specification processes on the one side and on a set of implementation processes on the other side induce a *Galois-connection* between the two sides of sets (cf. [DP02]).

Definition 5 (The information content of a cryptographic protocol). *The information content (in the sense of Kolmogorov-complexity [LV97]) of a protocol state $\mathfrak{s} \in \mathcal{H} \times \mathcal{P}$, containing the protocol(s), to agent $a \in \mathcal{A}_{\text{Eve}}$, written $\mathrm{K}_a(\mathfrak{s})$, is defined to be the smallest message $M \in \mathcal{M}$ s.t. $[\![M]\!]_a^{\mathfrak{s}} = [\![\mathfrak{s}]\!]_a^*$.*

5 Conclusion

We believe having made an original and intuitive proposal for the meaning and information content of a cryptographic message and protocol, and coined and applied a useful notion of context-sensitivity for message meaning. The resulting conception of protocol agents as evolving Scott information systems seems natural, and connects two previously, not obviously related fields of knowledge. The obvious temptation is now to attempt a result in the style of *full abstraction* for our denotational semantics w.r.t. to the operational semantics. “Denotational semantics is to programming languages what model theory is to predicate logic.” [DP02]. In this analogy, sound and complete proof systems correspond to fully abstract denotational semantics.

Acknowledgements I would like to thank Johannes Borgström for valuable comments on a draft of this paper.

A Proofs

We recall

- Definition: $\models \phi$:iff for all $\mathfrak{s}$, $\mathfrak{s} \models \phi$
- Lemma [Kra06,Kraar]: $\phi \Rightarrow \phi'$ iff $\models \phi \to \phi'$

Proof of Theorem 1

1. $\emptyset \neq \llbracket M \rrbracket^{\mathfrak{s}}_a \subset \mathcal{F}/_{\equiv}$: $\emptyset \neq \llbracket M \rrbracket^{\mathfrak{s}}_a$ because $[\top] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ due to $\models \mathsf{K}_a(\top)$ (and also $[a \mathsf{k} M] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ due to $\models a \mathsf{k} M \rhd \mathsf{K}_a(a \mathsf{k} M)$); and $\llbracket M \rrbracket^{\mathfrak{s}}_a \subset \mathcal{F}/_{\equiv}$ because $[\bot] \in \mathcal{F}/_{\equiv}$, but $[\bot] \notin \llbracket M \rrbracket^{\mathfrak{s}}_a$ due to $\not\models \mathsf{K}_a(\bot)$.
2. (a) if $[\phi], [\phi'] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ then $[\phi] \wedge [\phi'] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$:

1	$[\phi] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ and $[\phi'] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$	hyp.
2	$\mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{K}_a(\phi)$ and $\mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{K}_a(\phi')$	1
3	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi)$ and for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi')$	2
4	for all $\mathfrak{s}'$, (if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi)$) and (if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi')$)	3
5	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\begin{pmatrix} \mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi) \text{ and} \\ \mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi') \end{pmatrix}$	4
6	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi) \wedge \mathsf{K}_a(\phi')$	5
7	$\models (\mathsf{K}_a(\phi) \wedge \mathsf{K}_a(\phi')) \to \mathsf{K}_a(\phi \wedge \phi')$	property of K_a
8	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \wedge \phi')$	6, 7
9	$\mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{K}_a(\phi \wedge \phi')$	8
10	$[\phi] \wedge [\phi'] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$	9

   (b) if $[\phi] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ and $[\phi'] \in \mathcal{F}/_{\equiv}$ and $\phi \leq \phi'$ then $[\phi'] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$:

1	$[\phi] \in \llbracket M \rrbracket^{\mathfrak{s}}_a$ and $[\phi'] \in \mathcal{F}/_{\equiv}$ and $\phi \leq \phi'$	hyp.
2	$\mathfrak{s} \models a \mathsf{k} M \rhd \mathsf{K}_a(\phi)$ and $\phi \Rightarrow \phi'$ and $\text{keys}(\phi') \subseteq \text{keys}(\phi)$	1
3	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathsf{k} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi)$	2
4	$\phi \Rightarrow \phi'$ iff $\models \phi \to \phi'$	Lemma
5	$\models \phi \to \phi'$	2, 4
6	if $\models \phi \to \phi'$ then $\models a \mathsf{k} K \to \mathsf{K}_a(\phi \to \phi')$ where K designates a tuple built from $\text{keys}(\phi \to \phi')$	prop. of K_a
7	$\models a \mathsf{k} K \to \mathsf{K}_a(\phi \to \phi')$	5, 6
8	$\models \mathsf{K}_a(\phi) \to a \mathsf{k} K'$ were K' designates a tuple built from $\text{keys}(\phi)$	property of K_a

9	$\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$	hyp.
10	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi)$	3, 9
11	$\mathfrak{s}' \circ \mathfrak{s} \models a \mathrel{\mathsf{k}} K'$	8, 10
12	$\mathfrak{s}' \circ \mathfrak{s} \models a \mathrel{\mathsf{k}} K$	2, 11(, property of k)
13	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \to \phi')$	7, 12
14	$\models \mathsf{K}_a(\phi \to \phi') \to (\mathsf{K}_a(\phi) \to \mathsf{K}_a(\phi'))$	property of K_a
15	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi) \to \mathsf{K}_a(\phi')$	13, 14
16	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi')$	10, 15
17	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi')$	9, 16
18	$\mathfrak{s} \models a \mathrel{\mathsf{k}} M \rhd \mathsf{K}_a(\phi')$	17
19	$[\phi'] \in [\![M]\!]^{\mathfrak{s}}_a$	18

3. if $[\phi] \vee [\phi'] \in [\![M]\!]^{\mathfrak{s}}_a$ and $[\phi] \vee [\phi''] \in [\![M]\!]^{\mathfrak{s}}_a$ then $[\phi] \vee ([\phi'] \wedge [\phi'']) \in [\![M]\!]^{\mathfrak{s}}_a$:

1	$[\phi] \vee [\phi'] \in [\![M]\!]^{\mathfrak{s}}_a$ and $[\phi] \vee [\phi''] \in [\![M]\!]^{\mathfrak{s}}_a$	hyp.
2	$\mathfrak{s} \models a \mathrel{\mathsf{k}} M \rhd \mathsf{K}_a(\phi \vee \phi')$ and $\mathfrak{s} \models a \mathrel{\mathsf{k}} M \rhd \mathsf{K}_a(\phi \vee \phi'')$	1
3	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee \phi')$ and for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee \phi'')$	2
4	$\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$	hyp.
5	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee \phi')$ and $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee \phi'')$	3, 4
6	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee \phi') \wedge \mathsf{K}_a(\phi \vee \phi'')$	5
7	$\models (\mathsf{K}_a(\phi \vee \phi') \wedge \mathsf{K}_a(\phi \vee \phi'')) \to \mathsf{K}_a((\phi \vee \phi') \wedge (\phi \vee \phi''))$	prop. K_a
8	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a((\phi \vee \phi') \wedge (\phi \vee \phi''))$	6, 7
9	$\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee (\phi' \wedge \phi''))$	8
10	for all $\mathfrak{s}'$, if $\mathfrak{s}' \models a \mathrel{\mathsf{k}} M$ then $\mathfrak{s}' \circ \mathfrak{s} \models \mathsf{K}_a(\phi \vee (\phi' \wedge \phi''))$	4, 9
11	$\mathfrak{s} \models a \mathrel{\mathsf{k}} M \rhd \mathsf{K}_a(\phi \vee (\phi' \wedge \phi''))$	10
12	$[\phi] \vee ([\phi'] \wedge [\phi''])] \in [\![M]\!]^{\mathfrak{s}}_a$	11

Proof of Theorem 2 Analogous to the proof of Theorem 1 because that proof only relies on **S4** (adapted to the cryptographic setting), not full **S5** (adapted to the cryptographic setting), and provability is about **S4**.

Proof of Proposition 1

1. $[\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} \subset [\![M]\!]^{\mathfrak{s}}_a$: because $\models \mathsf{P}_a(\phi) \to \mathsf{K}_a(\phi)$ (cf. [Kra06,Kraar]), but $\not\models \mathsf{K}_a(\phi) \to \mathsf{P}_a(\phi)$
2. there is an order-embedding $e : [\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} \hookrightarrow [\![M]\!]^{\mathfrak{s}}_a$: take $e := \mathrm{id}_{[\![M]\!]^{\mathfrak{s}}_a}\langle [\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a} \rangle$, i.e., the restriction to $[\![M]\!]^{\mathfrak{s}}_{\mathsf{P}_a}$ of the identity on $[\![M]\!]^{\mathfrak{s}}_a$.

Proof of Theorem 3

$$[\![\mathfrak{s}]\!]_a = [\![\mathfrak{s}']\!]_a \quad \text{iff}$$

$$T\left(\bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s} \models a \,\mathsf{k}\, M}} [\![M]\!]_a^{\mathfrak{s}}\right) = T\left(\bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s}' \models a \,\mathsf{k}\, M}} [\![M]\!]_a^{\mathfrak{s}'}\right) \quad \text{iff}$$

$$\bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s} \models a \,\mathsf{k}\, M}} [\![M]\!]_a^{\mathfrak{s}} = \bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s}' \models a \,\mathsf{k}\, M}} [\![M]\!]_a^{\mathfrak{s}'} \quad \text{iff}$$

$$\bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s} \models a \,\mathsf{k}\, M}} \{\, [\phi] \mid \phi \in \mathcal{F} \,;\, \mathfrak{s} \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi) \,\} =$$
$$\bigcup_{\substack{M \in \mathcal{M}\\ \mathfrak{s}' \models a \,\mathsf{k}\, M}} \{\, [\phi] \mid \phi \in \mathcal{F} \,;\, \mathfrak{s}' \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi) \,\} \quad \text{iff}$$

$$\{\, [\phi] \mid M \in \mathcal{M} \,;\, \mathfrak{s} \models a \,\mathsf{k}\, M \,;\, \phi \in \mathcal{F} \,;\, \mathfrak{s} \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi) \,\} =$$
$$\{\, [\phi] \mid M \in \mathcal{M} \,;\, \mathfrak{s}' \models a \,\mathsf{k}\, M \,;\, \phi \in \mathcal{F} \,;\, \mathfrak{s}' \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi) \,\} \quad \text{iff}$$

$$\left(\begin{array}{l} [\text{ for all } M \in \mathcal{M},\, \mathfrak{s} \models a \,\mathsf{k}\, M;\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s} \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi)\,] \text{ iff}\\ [\text{ for all } M \in \mathcal{M},\, \mathfrak{s}' \models a \,\mathsf{k}\, M;\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s}' \models a \,\mathsf{k}\, M \rhd \mathsf{K}_a(\phi)\,] \end{array}\right) \quad \text{iff}$$

$$\left(\begin{array}{l} [\text{ for all } M \in \mathcal{M},\, \mathfrak{s} \models a \,\mathsf{k}\, M;\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s} \models \mathsf{K}_a(\phi)\,] \text{ iff}\\ [\text{ for all } M \in \mathcal{M},\, \mathfrak{s}' \models a \,\mathsf{k}\, M;\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s}' \models \mathsf{K}_a(\phi)\,] \end{array}\right) \quad \text{iff}$$

$$\left(\begin{array}{l} [\text{ for all } M \in \mathcal{M},\, \mathfrak{s} \models \mathsf{K}_a(a \,\mathsf{k}\, M);\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s} \models \mathsf{K}_a(\phi)\,] \text{ iff}\\ [\text{ for all } M \in \mathcal{M},\, \mathfrak{s}' \models \mathsf{K}_a(a \,\mathsf{k}\, M);\, \text{for all } \phi \in \mathcal{F},\, \mathfrak{s}' \models \mathsf{K}_a(\phi)\,] \end{array}\right) \quad \text{iff}$$

$$\left(\begin{array}{l} \text{for all } \phi \in \mathcal{F},\, \mathfrak{s} \models \mathsf{K}_a(\phi) \text{ iff}\\ \text{for all } \phi \in \mathcal{F},\, \mathfrak{s}' \models \mathsf{K}_a(\phi) \end{array}\right) \quad \text{iff}$$

$$[\text{ for all } \phi \in \mathcal{F},\, \mathfrak{s} \models \mathsf{K}_a(\phi) \text{ iff } \mathfrak{s}' \models \mathsf{K}_a(\phi)\,]$$

Proof of Theorem 4

1. the map $\phi \mapsto \ominus\phi$ (a) order-embeds $[\![\mathfrak{s}]\!]_a^n$ in $[\![\mathfrak{s}]\!]_a^{n+1}$, and (b) is order-continuous:
   (a) $\phi \leq \phi'$ iff $\ominus\phi \leq \ominus\phi'$ because $\models \mathsf{K}_a(\phi) \leftrightarrow \oplus\mathsf{K}_a(\ominus\phi)$
   (b) the map $\phi \mapsto \ominus\phi$ is order-continuous because it is order-embedding (thus order-preserving) and $[\![\mathfrak{s}]\!]_a^n$, being topped, satisfies the so-called ascending chain condition [DP02, Page 148]

   where $\ominus$ designates the previous-time and $\oplus$ the next-time operator of CPL from linear temporal logic [MP84].

2. $[\![\mathfrak{s}]\!]_a$, $[\![\mathfrak{s}]\!]_a^n$, and $[\![\mathfrak{s}]\!]_a^*$ are topped algebraic $\bigcap$-structures because being (directed) unions of filters they are topped, and closed under intersection and directed unions.
3. $[\![\mathfrak{s}]\!]$, $[\![\mathfrak{s}]\!]^n$, and $[\![\mathfrak{s}]\!]^*$ are pre-CPOs because they are disjoint (and thus lacking a least element) unions of CPOs.

References

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[AN05] S. Artemov and E. Nogina. Introducing justification into epistemic logic. *Journal of Logic and Computation*, 15(6), 2005.

[AT91] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the ACM Symposium of Principles of Distributed Computing*, 1991.

[BGK06] J. Borgström, O. Grinchtein, and S. Kramer. Timed Calculus of Cryptographic Communication. In *Proceedings of the Workshop on Formal Aspects in Security and Trust*, 2006.

[BKN06] J. Borgström, S. Kramer, and U. Nestmann. Calculus of Cryptographic Communication. In *Proceedings of the LICS-Affiliated Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, 2006.

[BM03] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[CD05] M. Cohen and M. Dam. A completeness result for BAN logic. In *Proceedings of the Workshop on Methods for Modalities*, 2005.

[Dam89] M. F. Dam. *Relevance Logic and Concurrent Composition*. PhD thesis, University of Edinburgh, 1989.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990 (2002).

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[Gro92] A. J. Grove. Semantics for knowledge and communication. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1992.

[GSW06] D. Goldin, S. A. Smolka, and P. Wegner, editors. *Interactive Computation: The New Paradigm*. Springer-Verlag, 2006.

[Kra06] S. Kramer. Logical concepts in cryptography. Cryptology ePrint Archive, Report 2006/262, 2006. `http://eprint.iacr.org/`.

[Kraar] S. Kramer. Cryptographic Protocol Logic: Satisfaction for (timed) Dolev-Yao cryptography. *Journal of Logic and Algebraic Programming*, to appear.

[LV97] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, second edition, 1997.

[MP84] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1984.

[PR03] R. Parikh and R. Ramanujam. A knowledge based semantics of messages. *Journal of Logic, Language and Information*, 12, 2003.

[Wit75] L. Wittgenstein. *Tractatus Logico-Philosophicus*. Routledge, English edition, 1961, 1975.

Secrecy Checking of Protocols: Solution of an Open Problem ⋆

Zhiyao Liang Rakesh M Verma

Computer Science Department, University of Houston,
Houston TX 77204-3010, USA
Email: zliang@cs.uh.edu, rmverma@cs.uh.edu
Telephone: (713) 743–3338 Fax: (713) 743–3335

Abstract. This paper proves the undecidability of an open problem on the complexity of checking secrecy of cryptographic protocols due to Durgin, Lincoln and Mitchell. The proof is by a reduction from 2-counter machines to protocols, and we prove both directions of the reduction in detail. The modeling and proof method are generally applicable and can be conveniently adapted to solve other problems about the complexity analysis of checking properties of protocols.

Key words: Cryptographic protocols, secrecy, undecidability.

1 Introduction

Analyzing the security properties of cryptographic protocols has been one of the most important challenges nowadays when networks are ubiquitous. A significant research direction in this area is to check secrecy and authentication of the protocols against a Dolev-Yao attacker [1] assuming that the cryptographic algorithms cannot be broken. Since Lowe discovered an attack on the public key Needham-Schroeder protocol [2, 3] 17 years after it was published [4], many papers have been published on this topic. To check secrecy failure or correctness is a very hard problem. One bound on the complexity of secrecy problem is that when the number of role instances in the protocol run is bounded, the secrecy problem is NP-complete [5], even when composite keys are allowed [6].

Secrecy checking is undecidable assuming unbounded number of role instances in a protocol run (together with other specific assumptions). Undecidability of secrecy checking is mentioned by several papers [7] [8] [9] [10] [11] [12] [13] [14], and [9] [10] [11] [13] provide proofs with details. The survey paper [14] is partly motivated by the work of [13] and partly to clarify the sketched proof in [10] on showing undecidability of secrecy by reduction from PCS (Post Correspondence Problem).

In [11] and [10] the authors use MSR (multi-set rewriting) to analyze protocols. In these papers the focus is on bounding the symbolic size of each message instance that can appear in a run of the protocol, and the number of messages in

⋆ Research supported in part by NSF grants CCF 0306475 and DUE 0313880.

every role of the protocol is bounded. When the total number of distinct nonce instances that can be generated by regular principal instances is unbounded, and the symbolic message size of all messages instances are bounded by a number, secrecy verification is undecidable. The proof is by a two-stage reduction from the halting problem of a Turing machine with the style of Turing machine tableau to Horn clause theories without function symbols and then from Horn clause theory to protocols specified as a set of roles. When the attacker can generate unbounded number of nonces and the regular agents can record nonces and check the uniqueness of each received nonce in a run, and a run can have unbounded number of role instances, the complexity of checking secrecy is an *open problem* [10] [11]. The open problem is stated precisely in theorem 1.

In [13] the authors show that the undecidability result of [10] can be proved more directly by a reduction from the reachability problem of a 2-counter machine to the secrecy checking problem of a protocol as a set of roles (we call the protocol role-oriented). In addition, by replacing unique nonces with unique composite terms, [13] proved the undecidability of secrecy checking when the symbolic size of message instances are unbounded, while the nonce instances generated in the run are bounded (in fact, no nonce generation is required).

In a recently published paper [15], Froschle showed that the secrecy problem is NEXPTIME-complete of for a setting (problem 4 of [15]) where disequality tests are allowed and only bounded number of nonce can be used (can appear) in role instances (called sessions in [15]) executed by regular agents. The motivation to consider disequality tests is that the unbounded set of nonces generated by an attacker is not necessarily reducible to a bounded set of terms trivially because of disequality tests. Although in [15] the author mentioned that problem 4 is pointed out by [10], it does not match the description of the open problem of [10] (quoted in the appendix of this paper) in two ways. First, the disequality tests in problem 4 can only apply to terms occurring in the same role instance, while the 'disequality test' in the open problem are used by an agent trying to enforce freshness of a term and should be applied to terms recorded across different role instances. Second and more importantly, the open problem does not assume that "the number of fresh data used in honest sessions is bounded" as in problem 4. The notation of "bounded $\exists$" in [10] means bounded number of nonces are *generated* by role instances, and not that the total number of nonces in role instances is bounded. This assumption will make the open problem (also problem 4) obviously decidable. Since the messages sizes are bounded in a run, and assuming agent names and terms other than nonces are bounded, although unbounded number of role instances are allowed, only bounded many distinct role instances (with different messages) will appear in the run. So problem 4 is reduced to a setting with bounded number of role instances (note that we only need to consider role instances executed by regular agents), independent of whether disequality tests are allowed or not, a decidable situation. Interestingly, the authors of [10] conjectured that the open problem is undecidable (page 71 of [11]).

Freshness check of [14] means that in any situation where a nonce is received by a regular agent that should be freshly generated according to the protocol it must indeed be fresh, that is, different from all terms appeared so far in a run. One difference between the freshness check and the open problem disequality test of [10] is that the authors do not explain the implementation aspects of the freshness check, whereas the open problem considers the internal operations of agents trying to ensure freshness. The second difference is that freshness check is global to all terms that appeared in a run so far, whereas our uniqueness check that implements the disequality test of [10] only ensures freshness local to individual agents. Recording terms in the memory of individual agents seems to be the only way to implement freshness check. The third difference is that freshness check of [14] ensures a nonce to be different from all terms appearing in a run so far, whereas the open problem only requires the uniqueness of a nonce among all nonces. Our proof, which considers uniqueness check described later, can easily be adapted to show the undecidability in the corresponding setting where 'freshness check' of [14] is assumed. To prove this, the adjustment to our proof is to design a protocol such that every message must go through a special server s, who records every subterm appeared so far in the run. It is possible that this setting has been considered in [16] but we are unable to verify this since this paper is still not available to us so far upon request to the authors.

Two factors are crucial for us to solve the open problem. First, we model the problem carefully and second, we utilize an improved and more direct reduction scheme, from 2-counter machines to security protocols. We give a rigorous and complete proof of correctness of the reduction. Our scheme is also applicable beyond the open problem. Our reduction scheme using 2-counter machine is inspired from [13]. However there are key differences. The paper [13] dealt with different problems, not the open problem. Because of the constraints of the open problem, we cannot use their scheme directly and need new ideas such as stamping nonces with agent id's. Moreover, we have found and fixed two errors in the reduction of [13]: a counter can be negative, and zero can be used as a positive number. Details of the errors and our fixes are included in the appendix of [17] The proof of correctness of the reduction in [13] is sketchy and consequently misses the two errors.

2 Notations and Modeling

We introduce our notations and modeling here. A more detailed description of them can be found at [17]. Notations are chosen in a style that is commonly used in the literature, e.g., [2]. The notations for asymmetric keys are new.

A *term* is either an atomic term or a composite term. An *atomic term* is a *variable* (represented by a symbol with at least one upper case letter), and a constant (a symbol without any upper case letter). A special constant is I, the name of the attacker. Asymmetric keys are atomic terms. A pair of asymmetric keys is represented as k_X^1 and k_X^0. X is the unique ID (UID) of the asymmetric key pair. When X is the name of an agent, k_X^0 and k_X^1 represent the established private key and public key of the agent X, respectively. This notation can also be adapted to describe the asymmetric keys generated during a run. A *composite term* is a list, or an asymmetric encryption, or a symmetric encryption. A *list* has the form of $[X, Y, \cdots]$, where X and Y are terms and the list contains finite number of member terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. When a message is a list, the top level enclosing [ ] is omitted. An *asymmetric encryption*, has the form of $\{T\}_{k_A^i}^{\rightarrow}$, $i \in \{0, 1\}$, where T is the encrypted term, and k_A^i is the atomic encryption key, and it can be decrypted using the key k_A^{1-i}. A *symmetric encryption* has the form of $\{T\}_Y^{\leftrightarrow}$, where T is the encrypted term and Y is term working as the encryption key (Y could be a composite term). For both asymmetric or symmetric encryption, when a list, say $[X, Y, Z, \cdots]$ is encrypted, the enclosing square brackets are removed from within "{ }". The word *ground* means variable free. A *message* is a term. Every message appearing in a run of a protocol is a ground term.

The attacker model is, as usual, the Dolev-Yao model [1]. There are different equivalent formalizations for the Dolev-Yao model, such as (not a comprehensive list) Paulson's [18], Multiset Rewriting (MSR) [10], Constraint Solving [19], and Strand Space [20]. Our model is somewhat similar to Paulson's [18] where a run is represented as a trace which is convenient for proofs based on induction. We will clarify the unique features of our model, which are needed for the open problem.

A clear consensus of modeling can be described as follows. A protocol can be described as a set of roles, each role is a sequence of actions steps of message sending or receiving executed by an agent. A run E is a sequence of actions steps formed by interleaving (prefixes of) role instances (called strands in [20]) executed by regular agents, where before every message Msg can be received by a regular agent at a certain point of the run, say after E' which is a prefix of E, the attacker I must be able to construct Msg, or $Msg \in know_I(E')$. Here $know_I(E')$ represents the knowledge of the attacker built from a set well-known analysis and synthesis rules [18] on the messages appeared in E' and the attacker's initial knowledge $init_I$. The secrecy checking protocol is to check if a secret term Sec can be leaked, or $Sec \in know_I(E)$ after a run E of the protocol. The formal proof of our reduction is based on the formal definitions of a protocol run and $know_I(E)$, and should be independent to different but equivalent choices to define them. A reader familiar with the formal concepts of protocol run and attacker's knowledge can directly verify the correctness of the proof assuming their own definitions of *run* and $know_I(E)$.

In the reduction proofs of undecidability of published papers such as [9] [10] [13] [14] and NP-hardness [5] [6], a constructed protocol is presented directly as a set of roles, we call them *role-oriented* (RO) protocols. However we call these protocols non-matching, since they do not correspond to protocols in the form of a sequence of message exchanges, such as those in [21]. For compatibility with other papers and especially with [11] and [10], which describe the open problem, in this paper a protocol presented is RO and non-matching.

An ***agent*** is a tuple $[name, init, mem]$, where *name* is its unique name, *init* is its initial knowledge (a set of terms), and *mem* is the set of terms that it has remembered so far in the current protocol run. The *mem* field is essential to explain the open problem. The different patterns of the initial knowledge of agents will be defined in the protocol. *Regular agents* means honest agents.

An ***action*** can be an ***internal action*** or an ***external action***. Let P, A, and B be agent names. The internal action of fresh term generation is denoted as $\#_P(t1, t2, \cdots)$, where $t1, t2, \cdots$ represent the fresh terms like nonces (they should be different from all other terms appeared in the run so far) generated by agent P before P sends a message that contains these fresh terms. An external action can be a message sending or a message receiving. The action of agent A to send a message Msg, when the intended receiver is B, $A \neq B$ is denoted as $+(A \Rightarrow B) : Msg$. The action of agent A to receive a message Msg from a supposed sender B, $A \neq B$, is denoted as $-(B \Rightarrow A) : Msg$. Some internal action can be implicitly described by the protocol code, such as equivalence checking for the values of the same term. However, some other internal actions cannot be expressed implicitly. In this paper, the only kind of internal actions explicitly expressed in the action code are the fresh term generations. Some other internal actions, such as uniqueness check of terms, when they are required to be expressed, such as those required by the open problem, are described in the conditions of a specific role of the protocol.

An ***action step*** is a sequence of actions. It has four forms. 1) $\#_I(term1, term2, \cdots)$; 2) $+(A \Rightarrow B) : Msg$; 3) $-(B \Rightarrow A) : Msg$; 4) $\#_A(t1, t2, \cdots)$ $+(A \Rightarrow B) : Msg$; 1) is executed by I, while other three can be executed by both a regualar agent or I. 4) is the only kind of action step that is a sequence of more than one action.

A ***role*** or *role template* or *role type*, is a tuple $[RID, agent, vars, acts, conds]$, where RID is the UID of the role, which is a constant, *agent* is the the agent who will execute the role template, $vars$ is the set of variables that appear in *acts* or *conds* (the other atomic terms appearing in the role are constants), *acts* is the sequence of action steps numbered sequentially starting from 1, and *conds* is the internal actions that are not implicitly expressible by the *acts*. The notation $n.pre : (cond1, cond2, \cdots)$ represents the conditions that should be checked and satisfied before the n^{th} action step (before accepting a received message, or before sending a message) is executed, where n is an action step number. The statement $n.post : (cond1, cond2, \cdots)$ describes the conditions that are enforced to be satisfied after the n^{th} action step is executed to update the properties of agents. Especially, when a condition $X \in Q$ is included in *n.pre*, it means to check term X is in set Q before the n^{th} action step. Q should be defined in the context of the protocol or the role. If $X \in Q$ appears in *n.post* it means to insert term X into set Q after the n^{th} action step.

A ***role instance*** is a tuple $[agent, role, vmap, acts]$, where *agent* is the agent who executes the role instance, *role* is the role template for this role instance, *vmap* is a ground substitution, (note that$vmap(role.conds)$ should be satisfied), and *acts* is the sequence of (ground) action steps, and $acts = vmap(role.acts)$.

A ***protocol*** Pro is a tuple $[PID,\ roles,\ AN, rsts]$, where PID is the UID of the protocol (a constant), $roles$ is a set of role templates, AN is the set of agent names (insiders) which are to be instantiated in the setting of a run, $rsts$ is the restrictions describing the initial knowledge and initial memory (indicated as $mem^{initial}$) of the agents, and definitions of some related sets if needed, such as the a set of terms shared by a certain group of agents.

A ***Dolev-Yao attacker*** [1], or an ***attacker*** for short, is a tuple $[name,$ $init_I,\ know_I]$, where by convention $name = I$, $init_I$ is the initial knowledge of the attacker, and $know_I$ is a function. After a sequence E of action steps has been executed, $know_I(E)$ is the set of terms that the attacker can obtain. $know_I(E)$ is calculated as the closure of a applying a set of well-known rules of term synthesis and analysis as showed in [18]. Details of these rules are presented in [17].

A ***run*** is a tuple $[Pro,\ D,\ R,\ AN,\ E,\ conds]$, where Pro is the protocol, D is the initial knowledge pattern of the specific Dolev-Yao attacker, R is a set of role instances that are executed honestly by regular agents, AN is the names of the agents who can legally participate in a run (AN instantiates $Pro.AN$), E is a sequence of actions steps, which is called a trace in Paulson's model [18], and $conds$ is the set conditions required for a run of the protocol. $conds$ includes the conditions described below together with the definition of insider and outsider.
1) $Pro.rsts$ should be satisfied. That is, the initial knowledge of every agent in AN are assigned with a set of ground terms according to $Pro.rsts$
2) For each role instance prefix r in R, $r.agent.name \in AN$, and $r.agent.name \neq I$, The action steps of $r.acts$ are included in $run.E$ preserving the relative order.
3) For each $X \in E$ executed by some regular agent, $X \in r.acts$, for some $r \in R$.
4) Each agent with its name included in AN is called an ***insider***. Especially, if $I \in AN$ then I is an ***insider attacker***, then I's initial knowledge patter should be the same as (some) other regular agents as specified by $Pro.rsts$. Otherwise if $I \notin AN$, I is an ***outsider***, and then we assume the attacker's initial knowledge pattern D will instantiate $I.init$ by a set of ground terms that is a subset of the initial knowledge of every regular agent (insiders), usually only the agent names and public keys of the regular agents and some constants that is known to every agent.
5) Let $W \diamond X$ represent a sequence formed by appending an element X to a sequence W. For a prefix of E, call it E' and it is a sequence of action steps, suppose $E' = W \diamond X$, $X \in r.acts$ for some $r \in R$. If $X = -(A \Rightarrow B)msg$, where B is the name of a regular agent , then $msg \in know_I(W)$.
6) For a fresh term X that is generated by I, the action $\#_I(M)$, where M is a list of terms including X, is explicitly included in $run.E$ before X can appear in any message receiving action step by a regular agent.

We present some explanations of the above conditions to define a run. For 2), only the behavior of regular agents are organized into role instances. Although the attacker can execute a role instance normally as a regular agents, its behavior are covered by the Dolev-Yao model, and we only need to care about condition 5), that is, I can obtain every message before it can be received by a regular agent. For 4) we assume an outsider's initial knowledge should be less than any insider. Condition 6) is included for the convenience of describing $known_I(E)$, since the terms generated by I should be used to build the knowledge of I. Note that we do not need to explicitly record the message sending actions of the attacker.

We assume every run has an implicit stage to distribute keys and establish the initial knowledge of agents.

The set of all possible runs of a protocol Pro, with a specific initial knowledge pattern D of the attacker, is indicated as $runs^{D:Pro}$. Given a protocol Pro, and a specific D, and a set of secret terms SEC, A ***secrecy problem*** is to check the validity of the following statement.

$$\exists run, \exists X, run \in runs^{D:Pro}, X \in SEC : X \in know_I(run.E)$$

3 Solution of the Open Problem

In this section, we present the solution to the open problem. We are considering the lower bound of complexity. In other words, we show that given a set of conditions, in the worst case the problem is undecidable, and the theme is not relevant to the special cases that secrecy problems are decidable.

The open problem is described in [10] and [11], and is precisely stated in Theorem 1. Appendix A discusses it in more detail.

The protocols considered by [10] are bounded, which means two bounds. First, the number of messages in a role template (and also in a role instance), called the role length, is bounded. Second, the size of a message instance (the number of ground atomic terms appearing in a message, which is a term) that can appear in a run of the protocol is bounded. In other words, only the runs with bounded size of message instances are considered.

Note that, in the scenario of the open problem, nonce generations depend on the attacker, and the attacker can always use a composite term as a nonce. So type flaw is not avoidable. Note that in the proof we allow C_h and C_h^{-1} to be instantiated by a pair, where $h \in \{1, 2\}$. However, if we make a stronger requirement so that the open problem only considers runs of a protocol where no type flaw can occur, which means every variable can only be instantiated by an atomic term in a run considered, it is still undecidable. To prove this, we only need to adjust the protocol code in the proof and replace C_h and C_h^{-1} with pairs like $[A, C_h]$ and $[B, C_h^{-1}]$, and then encode 0 with a pair $[A, z]$ instead z, and then adjust the messages of the protocol accordingly. The rest of the proof is the same.

We need to ensure that a 2-counter machine can reach its final state if and only if there is a run of the corresponding protocol (constructed from the 2-counter machine) in which the secret term is leaked.

Definition 1. *A* **deterministic 2-counter machine** *[22] with empty input is a pair (Q, δ), where Q is a set of states including the starting state q_0 and the accepting state q_{final} and δ is a set of transition rules. A configuration of a*

2-counter machine is a tuple (q, V_1, V_2), where q is the current state and V_1 and V_2 are two non-negative integers representing the two counters. The 2-counter machine can detect whether a counter is 0 or not. A transition rule, (call the rule $T \in \delta$) is of the form $[q, i_1, i_2] \rightarrow [q', j_1, j_2]$, where $q, q' \in Q$; $i_1, i_2 \in \{0, 1\}$; $j_1, j_2 \in \{-1, 0, +1\}$. An application of T can be described as $(q, V_1, V_2) \longrightarrow^T (q', V_1', V_2')$, where LHS and RHS are the configuration before and after the transition respectively. For $h \in 1, 2$, when $i_h = 0$, it means that $V_h = 0$. When $i_h = 1$, it means that $V_h > 0$. When $j_h = +1$ ($j_h = 0$, $j_h = -1$), it means that after the transition, $V_h' = V_h + 1$ ($V_h' = V_h$, $V_h' = V_h - 1$). Especially, when $j_h = -1$, i_h must be 1, since decrementing 0 is not allowed. The reachability problem of such a 2-counter machine is to decide that, starting from the initial configuration $(q_0, 0, 0)$, after applying some applicable transition rules, whether some final configuration $(q_{final}, \_, \_)$ can be reached, where $\_$ represents an arbitrary possible value. We assume (for convenience) that $q_0 \neq q_{final}$ and, for nontriviality, that δ is not empty.

It is obvious that a 2-counter machine allowing $q_0 = q_{final}$ can be equivalently simulated by a 2-counter machine defined above, and the reachability problem of 2-counter machines defined above is undecidable.

Theorem 1. *The open problem of [10] is undecidable. Specifically, checking secrecy is undecidable, assuming: (i) the protocol has bounded number of messages in a role (role length), (ii) considering only the runs where the sizes of messages are bounded, (iii) the number of role instances in a run of the protocol is unbounded, (iv) regular agents can generate only bounded number of nonces, (v) the attacker can generate unbounded many nonces, (vi) the internal action of disequality test on two terms is allowed, (vii) considering only the runs where the number of agents is bounded, and (viii) when a term is supposed to be freshly generated nonce and is received by some regular agent, who records every nonce encountered, it must be different from all other terms the agent has recorded so far in the run, and then it is recorded by the agent.*

Proof. We translate an arbitrary 2-counter machine into a protocol which fits in the scenario of the open problem. Every role has a different scope of variables. So a variable in role is independent of the variable with the same name in another role.

Given a 2-counter machine $M = (Q, \delta)$, let $Q = \{q_0,\ q_{final},\ q_1,\ q_2,\ \cdots,\ q_m\}$ and $\delta = \{T_1,\ T_2,\ \cdots,\ T_n\}$. The following is the description of the protocol Pro constructed according to M.

The messages received in a role is in the format of: $sender$, $receiver$, $receiver's$ $role$, $\cdots$, so the receiver of the message has clear hints to understand the message and know what he should do. The variables B, A_{final}, A_0, A_f, for $1 \leq f \leq n$ are agent names. We differentiate the namess of the variables representing the executors of different roles, including A_{final}, A_0, A_f, $1 \leq f \leq n$, for the clarity of the presentation, although a single variable can be used in different roles. B represents some agent talking with the executor of every role.

The set of secret term SEC is defined in $Pro.rsts$, which is initially known by every regular agent but the attacker. We specify the secret term Sec as a variable, instead of constant in the messages of a role, for a practical concern, so even though the attacker knows the protocol code, the attacker does not know the instance of Sec unless a role instance of R_{final} can be executed in a run, where Sec will be instantiated by a member of SEC.

In a role executed by an agent A, for the variables whose values are not determined by the agents other than A, we can categorize these variables in three kinds depending on A's different treatment to them. 1) The set of terms which A must check that they belong to the $A.init$, such as the agent names. 2) The set of terms which A does not care about whether A has seen them already or not, such as C_h and C_h^{-1}, no matter they should be nonces or not. 3) The set of terms which A must check its uniqueness (where the disequality $\neq$ applies), i.e., A has never seen it before, such as C_h^{+1}. We will adapt the proof to show in theorem 2 that when the variables of kind 2) are not allowed, the open problem is still undecidable.

$Pro = [PID,\ roles,\ AN,\ rsts]$. PID is arbitrary. $roles = \{R_0,\ R_{final},\ R_1,\ R_2,\ \cdots,\ R_n\}$.

- $R_0 = [RID,\ agent,\ vars,\ acts,\ conds]$
  - $RID = r_0$; $agent = [name,\ init,\ mem]$; $vars = \{A_0,\ B\}$
  - $acts = 1.\ +(A_0 \Rightarrow B) : \quad A_0, B, \{q_0, z, z\}_{\vec{k}^0_{g1}}$
  - $conds = \{\ 1.pre : (q_0,\ A_0,\ r_0,\ B,\ k^0_{g1}\} \subseteq init,\ agent.name = A_0,\ \{A_0, B\} \subset AN, A_0 \neq B)\ \}$
- $R_{final} = [RID,\ agent,\ vars,\ acts,\ conds]$
  - $RID = r_{final}$; $agent = [name,\ init,\ mem]$; $vars = \{A_{final},\ X,\ Y,\ B,\ Sec\}$.
  - acts$=$1. $-(B \Rightarrow A_{final}) : \quad B, A_{final}, r_{final}, \{q_{final}, X, Y\}_{\vec{k}^0_{g1}}$
    2. $+(A_{final} \Rightarrow B) : \quad A_{final}, B, Sec$
  - $conds =$ 1.pre : $(\{q_{final},\ A_{final},\ r_{final},\ B,\ k^1_{g1}\} \subseteq init,$
    $agent.name = A_{final}, \{A_{final}, B\} \subset AN,\ A_{final} \neq B)\ )$;
    2.pre : $(\ Sec \in init,\ \ Sec \in SEC\ )\ \}$
- For each $T_f \in \delta$, for some f, $1 \leq f \leq n$, suppose $T_f = [q, i_1, i_2] \rightarrow [q', j_1, j_2]$. $R_f \in roles$. R_f can be constructed according to T_f by the following description. $R_f = [RID, agent, vars, acts, conds]$.
  - $RID = r_f$. $agent = [name, init, mem]$. $vars = \{A_f, B, C_1,\ C_2, C_1^{-1}, C_1^{+1}, C_2^{-1}, C_2^{+1}\}$.
  - *acts*: The following is the template of acts. The exact action sequence of each R_f will be adjusted by the specific T_f and a set of rewrite rules. Note that $q, q', i_1, i_2, j_1, j_2$ may represent different constants for different f, according to the 2-counter machine specification. The variables C_1' and C_2', which represent the new counter values, will only be used in the template and they will not appear in the actual code of the R_f, since

they will be replaced by other terms after applying the rewrite rules.

1. $-(B \Rightarrow A_f) : B, A_f, r_f, \{q, C_1, C_2\}_{\vec{k}^0_{g1}}, \{C_1^{-1}, C_1\}_{\vec{k}^0_{g2}},$
   $\{C_2^{-1}, C_2\}_{\vec{k}^0_{g2}}, C_1^{+1}, C_2^{+1}$
2. $+(A_f \Rightarrow B) : A_f, B, \{q', C_1', C_2'\}_{\vec{k}^0_{g1}}, \{C_1, [A_f, C_1^{+1}]\}_{\vec{k}^0_{g2}},$
   $\{C_2, [A_f, C_2^{+1}]\}_{\vec{k}^0_{g2}}$

For $h \in \{1, 2\}$, the following rewrite rules are applied to adjust the above role template to make each individual transition role R_f according the corresponding transition rule T_f of the 2-counter machine. Each rewrite rule is described as "condition $\Mapsto$ effects".

1. $i_h = 0 \quad \Mapsto \; C_h \rightarrowtail z;\; \{C_h^{-1}, C_h\}_{\vec{k}^0_{g2}} \rightarrowtail \varepsilon$
2. $i_h = 1 \quad \Mapsto \; \{C_h^{-1}, C_h\}_{\vec{k}^0_{g2}} \in Msg_1$
3. $j_h = +1 \; \Mapsto \; C_h' \rightarrowtail [A_f, C_h^{+1}];\; C_h^{+1} \in Msg_1;\; \{C_h, [A_f, C_h^{+1}]\}_{k^0_{g2}} \in Msg_2$
4. $j_h = 0 \quad \Mapsto \; C_h' \rightarrowtail C_h;\; \{C_h, [A_f, C_h^{+1}]\}_{\vec{k}^0_{g2}} \rightarrowtail \varepsilon$; In Msg_1 $C_h^{+1} \rightarrowtail \varepsilon$
5. $j_h = -1 \; \Mapsto \; C_h' \rightarrowtail C_h^{-1};\; \{C_h, [A_f, C_h^{+1}]\}_{\vec{k}^0_{g2}} \rightarrowtail \varepsilon$; In Msg_1 $C_h^{+1} \rightarrowtail \varepsilon$

$W \rightarrowtail V$ means to replace W with V in the above action code template of R_f. $W \rightarrowtail \varepsilon$ means to remove W. $W \in Msg_1$ means the assertion that the term W will appear in message 1. An implicit rule is that any term in the template of $R_f.acts$ which is not removed or changed will still appear in the code. We emphasize that a term will appear in a message in some rule, even without explicitly saying so, the fact should still hold. If in R_f some variables will not appear in the actions since they will be removed by applying the rules, then these variables will also be removed from other fields such as $R_f.vars$ and $R_f.conds$. If a rule is only applied to Msg_1, it is labeled with "in Msg_1". $h \in \{1, 2\}$. Here is the explanation of the above rules.

1. Counter value 0 must be represented by z. There is no previous value for counter value 0 so no "number connection" term $\{C_h^{-1}, C_h\}_{\vec{k}^0_{g2}}$ is required in the role.
2. When a counter is positive, we emphasize that there must be evidence that it has a preceding nonnegative value. This rule is redundant since a default rule is that any term that is not removed from the template will still be there.
3. When a counter is incremented, the variable C_h' is replaced by a new pair $[A_f, C_h^{+1}]$, where C_h^{+1} is new nonce received by A_f. Note that in a run C_h^{+1} is provided by the intruder who impersonates B_f. The history records that the new counter value is incremented from its precedent is represented by the term $\{C_h, [A_f, C_h^{+1}]\}_{\vec{k}^0_{g2}}$.
4. When a counter is kept the same, neither the new nonce nor the record of increment is needed.
5. When a counter is decremented, the variable C_h' is replaced by the preceding counter representation. The new nonce and the record of incremented counter are not needed. When $j_h = -1$, i_h must be 1 (and rule 2 applies) if T_f is a valid transition rule of M.

The rewrite rules are applied as much as possible. For example, when $i_h = 0$ and $j_h = 0$, rule 4 is applied to change C'_h to C_h, and then rule 1 is applied to change C_h to z. In the rules 4 and 5, the label "In Msg_1" is to make sure that after $C_h^{+1} \rightarrowtail \varepsilon$ is applied, $\{C_h, [A_f, C_h^{+1}]\}_{k_{g2}^0}^{\rightarrow} \rightarrowtail \varepsilon$ is still applicable to Msg_2. So the order of rule application is not relevant. Some examples are showed in the appendix of [17] .

The conditions of R_f ($R_f.cond$), $1 \leq f \leq n$, is the follows.

$-$ $conds = \{$ 1.$pre:$ ($C_1^{+1} \neq C_2^{+1},\ C_1^{+1} \notin mem,\ C_2^{+1} \notin mem,$
$agent.name = A_f\ \{B, A_f, r_f, q, k_{g1}^1, k_{g2}^1\} \subset init,$
$\{A_f, B\} \subset AN,\ A_f \neq B$);
1.$post:$ ($\{C_1^{+1},\ C_2^{+1}\} \subseteq mem$); 2.$pre:$ ($\{k_{g1}^0, k_{g2}^0\} \subset init$); $\}$

We continue to finish describing remaining fields of Pro.
* AN are to be instantiated in a run of Pro.
* $rsts = \{$ $pk = Q \cup AN \cup \{r_0,\ r_{final},\ r_1,\ \cdots,\ r_n\} \cup \{z, k_{g1}^1, k_{g2}^1\}$; Let SEC be a set of terms. $SEC\ \cap\ pk = \{\}$; $gk = \{k_{g1}^0, k_{g2}^0\} \cup SEC$; $\forall P(P \in CA):$ $P.init\ =\ pk \cup gk,\ P.mem^{initial}\ =\ P.init$ $\}$

The initial knowledge pattern D of I's initial knowledge (as an ousider) considered in this proof is: $init.I = pk$, where pk is defined in $Pro.conds$.

We show that the constructed protocol and the proof satisfies the bounds imposed by the open problem, before we show further details of the reduction. Note that no regular agent will generate any fresh nonce, so the nonces generated from regular agents are trivially bounded. All of the nonces, which instantiate C_h^{+1}, $h \in \{1, 2\}$, in every role instance of R_f, are unbounded many, and can only be generated from the attacker I. Every role has at most two action steps, so the role length is bounded by two. The message size in a run is bounded by any number equal to or greater than 15, the size of the first message of R_f, for some f, $1 \leq f \leq n$. And every regular agent is required to do the uniqueness check of each term received that is supposed to be fresh nonce. The number of agents can be bounded by three, since in the proof (direction 1) we only assume two regular agents a and b in a run, while the intruder I is the third agent in a run.

As explained in the introduction section, the protocol is a non-matching role oriented one. The attacker is an outsider as described by the following paragraph.

A symmetric key is used as the encryption key in [13] [10] [6], which is known to all the regular agents (who are insiders) but unknown to the attacker (who is an outsider). So the attacker can neither construct an encryption, nor understand it, which could make it not practical for attacker to deploy an attack. In the proof of this paper we also consider the attacker is an outsider and we choose asymmetric keys k_{g1}^0 and k_{g2}^0 as the encryption keys, which are unknown to the attacker, but known to all of the regular agents. $g1$ and $g2$ are the UID of the key pairs, not agent names. The attacker I knows the decryption key k_{g1}^1 and k_{g2}^1. So I cannot construct the encryptions, but can decrypt them and understand them, and easily deploy the attack. Every role can only be executed by some regular agent since attacker cannot construct the encryptions in the messages.

Before we prove the correctness of the reduction, we explain the intuition. If M can reach a final configuration $(q_{final}, \_, \_)$ starting from $(q_0, 0, 0)$, then there is a finite sequence of configurations connected by applicable rules in δ. Call this computation of M, *Comp*, which can be written as

$$(q_0, 0, 0) \longrightarrow^{t_1} (Q^1, V_1^1, V_2^1) \cdots (Q^w, V_1^w, V_2^w) \longrightarrow^{t_{w+1}} (Q^{w+1}, V_1^{w+1}, V_2^{w+1}) \\ \cdots \longrightarrow^{t_u} (q_{final}, V_1^u, V_2^u)$$

where $w, u > 0$, $t_0, t_w, t_u \in \delta$, and u is the number of transitions in *Comp*.

After running a sequence of action steps E, we say a term X is the ***encoding*** of a positive integer N, if and only if there is a sequence of terms:
$\{z, X_1\}_{\overrightarrow{k^0_{g2}}}, \{X_1, X_2\}_{\overrightarrow{k^0_{g2}}}, \{X_2, X_3\}_{\overrightarrow{k^0_{g2}}}, \cdots, \{X_{N-2}, X_{N-1}\}_{\overrightarrow{k^0_{g2}}}, \{X_{N-1}, X\}_{\overrightarrow{k^0_{g2}}}$
such that for each element T of this sequence $T \in know_I(E)$. Here X and X_l, for some integer l, $1 \leq l \leq N-1$, are different variables that can represent any terms (could be composite terms). We call N the ***i_value*** of X (i stands for integer), or X is the ***encoding*** of N, denoted as $N = \underline{X}$. We say X ***encodes*** N. The above term sequence is called the ***encoding sequence*** of X. The encoding sequence of z is z.

The encoding of 0 is the special constant z. So $0 = \underline{z}$. A positive integer is encoded by a pair $[A, X]$, where A is an agent name and X is a nonce. The encodings of numbers are connected in an encryption to show the consecutive order between numbers. $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ means that $\underline{X} = \underline{Y} - 1$.

For the E of a run, let $E = W \diamond X$. If the last action setp X of E is to receive a message Msg, we will show that $Msg \in know_I(W)$.

Direction 1: Suppose M can reach a final configuration $(q_{final}, \_, \_)$ from the initial configuration, we prove that there is a run, call it *run*, such that $Sec \in know_I(run.E)$ for some term $Sec \in SEC$. We prove this direction by constructing *run*.

run = [*Pro*, *D*, *R*, *AN*, *E*, *conds*]. *Pro*: the protocol just described. *R*: A role instance r will obviously be included in R when some actions of r will be included in E when we show the proof. $AN = \{a, b\}$. Only two agents are enough here to instantiate the sender and receiver variables in each role. E: The action sequence is described below. *conds*: *SEC* is instantiated by $\{sec\}$. So *sec* is the only secret ground term. we will justify that is every message received by a regular agent can be constructed by the attacker.

Now we focus on describing *run.E*, which can be divided into three parts: the starting, the transition, and the finishing. We build *run.E* by appending actions to *run.E*, starting from an empty sequence.

We need to prove that the constructed *run* is a run, we only have to show two things. First, given a role instance r (executed by a regular agent) of the run, the internal actions and conditions described by *r.role.conds* should be satisfied. Particularly, all the instantiation of nonce variables should pass the uniqueness checking by the agent who executes the role instance. This is obvious since in the run all nonce variables are instantiated by nonces freshly honestly generated by the attacker, who can generate unbounded many fresh nonces and has no problem to do it.

Second, we need to show that if a message msg is received by in a regular role instance r, say at the end of an action sequence E', $E' = W \diamond X$, then $msg \in know_I(W)$. We only need to explain this aspect. A regular agent will receive a message either in a role instance of R_0 or of a R_f, $1 \leq f \leq n$, or of R_{final}. We will show that this condition is satisfied when we add an action of message receiving to $run.E$.

The ***starting action steps***. At the beginning of $run.E$, we choose a role instance of R_0, call it r^0, which means that $r^0 \in run.R$, $r^0.agent.name = A_0$. A_0 is instantiated by a B is instantiated by b. The first action of $run.E$ is: $+(A_0 \Rightarrow B): \quad A_0, B, \{q_0, z, z\}_{\vec{k^0_{g1}}}$.

The ***transition action steps***. Suppose w^{th} step in $Comp$ is $(q, V_1, V_2) \longrightarrow^t (q', V_1', V_2')$, where $0 \leq w \leq u$, and $t \in \delta$. If according to t, $j_1 = +1$ or $j_2 = +1$, which means that $V_1 + 1 = V_1'$ or $V_2 + 1 = V_2'$, the following action of nonces generation by I is appended to $run.E$: $\#_I(c_1^w, c_2^w)$, where c_1^w and c_2^w are two fresh nonces. Otherwise, this action is not appended to $run.E$.

The transition rule t corresponds to a transition role in the protocol, say R_f, where $1 \leq f \leq n$, and $R_f \in Pro.roles$. A role instance of R_f is included in the run for the w^{th} transition of the 2-counter machine, call it r^w. Then $r^w \in run.R$, $r^w.role = R_f$. The 2 actions of r^w are appended to $run.E$. According to pro, the two actions have the following general form.

$-(B \Rightarrow A_f): B, A_f, r_f, \{q, C_1, C_2\}_{\vec{k^0_{g1}}}, \{C_1^{-1}, C_1\}_{\vec{k^0_{g2}}}, \{C_2^{-1}, C_2\}_{\vec{k^0_{g2}}}, C_1^{+1}, C_2^{+1}$

$+(A_f \Rightarrow B): A_f, B, \{q', C_1', C_2'\}_{\vec{k^0_{g1}}}, \{C_1, [A_f, C_1^{+1}]\}_{\vec{k^0_{g2}}}, \{C_2, [A_f, C_2^{+1}]\}_{\vec{k^0_{g2}}}$

We have to specify for each variable in R_f its ground instantiation term. A_f and B are instantiated by a and b respectively. I impersonates B to send the first message to A_f. C_h^{+1} is instantiated by c_h^w, which is just freshly generated by I, for $h \in \{1, 2\}$. Now the variables in the above message template remaining to be instantiated in r_w are C_h, and C_h^{-1}, with $h \in \{1, 2\}$. We do not need to specify C_h', since it will be replaced by one of C_h^{-1}, C_h, or C_h^{+1} depending on the specific role R_f.

Let E^w be the prefix of $run.E$ which ends immediately before the first action of r^w. We require that the instantiation of C_h must encode V_h, denoted as $V_h = \underline{C_h}$, for $h \in \{1, 2\}$, after running E^w. q and q' are the same as the state names appearing in t. Intuitively speaking, we require $\{q, C_1, C_2\}_{\vec{k^0_{g1}}}$ to encode the configuration (q, V_1, V_2). If C_h^{-1} will appear in r^w, we require that C_h^{-1} encodes $V_h - 1$.

Let Msg^w be the message received by the first action of r^w. Now we need to show that $Msg^w \in know_I(E^w)$. If $[A_f, C_h^{+1}]$ appears in Msg^w, then by the design of the protocol, it must be true that in the transition t of the 2-counter machine, $j_h = +1$. Then by the construction of the run, c_h^w is just freshly generated by I. So C_h^{+1}, which is instantiated by c_h^w is in $know_I(E^w)$. A_f is initially known by I. So $[A_f, C_h^{+1}] \in know_I(E^w)$, for $h \in \{1, 2\}$. we only need to justify that the required terms $\{q, C_1, C_2\}_{\vec{k^0_{g1}}}$, and $\{C_h^{-1}, C_h\}_{\vec{k^0_{g2}}}$ (if it appears in Msg^w), are included in $know_I(E^w)$.

We prove this by showing a stronger result below. It is obvious that if Lemma 1 is proven, then $Msg^w \in know_I(E^w)$ is justified.

Lemma 1. *For the role instance r^w and the transition steps of M just described, the following three facts are true.*

1. *There exists $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^w)$, such that $V_1 = \underline{C_1}$ and $V_2 = \underline{C_2}$.*
2. *For $h \in \{1, 2\}$, if $V_h > 0$, then $\{C_h^{-1}, C_h\}_{\overrightarrow{k^0_{g2}}} \in know_I(E^w)$, such that after running E^w, $V_h - 1 = \underline{C_h^{-1}}$.*
3. *After running the two action steps of r^w, we call the executed action sequence so far $E^{w'}$. For $E^{w'}$, $V'_h = \underline{C'_h}$, for $h \in \{1, 2\}$. And $\{q', C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^{w'})$.*

This lemma can be proven by induction on the length of the computation of M. The details are included in [17].

The ***finishing action steps***: A role instance of R_{final}, call it r^{final}, is included in *run.R*. The following two actions of r^{final} are appended to *run.E*.

$-(B \Rightarrow A_{final}): \quad B, A_{final}, r_{final}, \{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$

$+(A_{final} \Rightarrow B): \quad A_{final}, B, Sec$

A_{final} and B are instantiated by a and b respectively. X and Y can be instantiated by any terms. *Sec* is instantiated by *sec*. Let E^{final} be the prefix of *run.E* that ends immediately before the first action of r^{final}. In order to show that the first message of $r^{final} \in know_I(E^{final})$, we only need to show that $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}} \in know_I(E^{final})$, the other terms are included in $init_I$. Since we assume the 2-counter machine can reach a final configuration $(q_{final}, \_, \_)$, the last transition step must have the form $(q, V_1, V_2) \longrightarrow^t (q_{final}, V'_1, V'_2)$. It is proven by Lemma 1 that the last transition action (the u^{th}) will produce a term $\{q_{final}, C'_1, C'_2\}_{\overrightarrow{k^0_{g1}}}$, where $V'_h = \underline{C'_h}$, for $h \in \{1, 2\}$.

It is obvious that at end of *run*, $sec \in know_I(E)$. Direction 1 is proved.

***Direction 2*:** We have to show for any *run*, $run \in Runs^{D:Pro}$, if $sec \in know_I(run.E)$, then the 2-counter machine M can reach a final configuration $(q_{final}, \_, \_)$.

The following observations are easy to verify. Due to limit of space, detailed explanation are included in [17].

Observation 1: First, every encrypted term is constructed by a regular agent. Second, two encrypted terms appearing in *run* with different format cannot be unified and cannot be used interchangeably because of different encryption keys.

Observation 2: A term of the form $\{X, z\}_{\overrightarrow{k^0_{g2}}}$ will never be generated in the run. If it can be generated, z must be a freshly generated nonce, impossible.

Observation 3: Given any term X, X can appear at most once in a term of the form $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$. Assuming the contrary, we can see that X must have the form of $[A, T]$, where T has been accepted by the same agent A as a fresh nonce twice, impossible.

Observation 4: For every term X, there can be at most one encoding sequence of X, and therefore X can only encode at most one number, especially z can only encode 0. We can see this directly by Observation 3. If X is z, then by Observation 2, there can only be one encoding sequence of z, which is z itself. For an encoding sequence of X, if $X \neq z$, then there is at most one term Y can appear in $\{Y, X\}_{\overrightarrow{k^0_{g2}}}$, and then at most one Y' can appear in $\{Y', Y\}_{\overrightarrow{k^0_{g2}}}$, and the reasoning continues. So X has at most one encoding sequnce.

On the other hand, it is possible that there exist two different terms of the form $\{X, Y_1\}_{\overrightarrow{k^0_{g2}}}$, and $\{X, Y_2\}_{\overrightarrow{k^0_{g2}}}$, where $Y_1 \neq Y_2$. In other words, a number can be encoded by several different terms, while each term can only encode one number. If we connect the encoding terms together where X is the parent of Y if there is a term $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ appearing in the run, then we can form a tree, whose top node is z. Every node (a term) of the tree, can have several children nodes, but can only have one parent node. Each term can appear at most once as a node in the tree.

Observation 5: The number 0 can only be encoded by z. By Observation 2, it is impossible for z to encode any positive number. One concern is that if X appears in $\{X, Y\}_{\overrightarrow{k^0_{g2}}}$ where Y encodes 1, and $X \neq z$, then X could be used as a term encoding 0. But since 1 is the *i_value* of Y, there must be a term $\{z, Y\}_{\overrightarrow{k^1_{g2}}}$ by the definition of *i_value*. It is impossible by Observation 3.

We prove direction 2 by proving a stronger result below.

Lemma 2. *For an arbitrary run of Pro with the attacker (an outsider, as described earlier) for every configuration term of the form $\{q, C_1, C_2\}_{\overrightarrow{k^0_{g1}}}$ generated in run (q is any state), it encodes a reachable configuration, say (q, V_1, V_2), of the two counter machine $M = (Q, \delta)$, in the sense that $V_h = \underline{C_h}$, for $h \in \{1, 2\}$.*

This lemma is proven by induction on the sequence of configuration terms generated in the *run*. The detailed proof is included in [17].

Now we finish the proof of direction 2. We assume that $sec \in know_I(run.E)$. Then sec must have been sent by a regular agent, since $sec \notin init_I$. A regular agent will generate sec only in the second message of a role instance, call if r^{final}, of R_{final}. r^{final} needs to receive a term of the form $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ in its first message, where X and Y are some arbitrary terms. By Observation 1, and by free term algebra assumption, $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ must be a configuration term. By Lemma 2, $\{q_{final}, X, Y\}_{\overrightarrow{k^0_{g1}}}$ must encode a configuration (q_{final}, V_1, V_2), which is a reachable configuration to the 2-counter machine. Direction 2 is proved.

To translate a description of a 2-counter machine to the corresponding protocol *Pro* can always be done in finite amount of time, since *Pro* is always constructed by finitely many symbols. Theorem 1 is proved. □

The proof can be enhanced to cover a stronger consideration, which could be a more restricted interpretation of the open problem.

In the proof of theorem 1, when an agent, say A, receives the first message in a transition role, A does not check the uniqueness of the variables C_h and C_h^{-1}, for $h \in \{1, 2\}$, neither does A check whether C_h and C_h^{-1} belong to the initial knowledge of A. Theorem 1 deals with a general consideration such that for the variables received by A which are not created by A and may not be known by A initially, A will check the uniqueness of some of them, but will not care about the others. In other words, uniqueness check by A is allowed but not required. This situation should be consistent to the description of table 1 in Appendix A.

However we have noticed that in the protocols appearing in the proofs of [10] [11] where the open problem is mentioned, there are only two types of variables appearing a protocol run: agent names, which must belong to the initial knowledge of agents, or the nonces (created by regular agents). A stronger consideration is that an agent A will treat all of the variables, which are not names, received from other agents uniformly as fresh nonces, i.e., A will always check their uniqueness, and the variables that A does not care about such as C_h and C_h^{-1} as in the general consideration of theorem 1 are not allowed. Theorem 2 solves the open problem with the this stronger consideration.

Theorem 2. *Suppose for a variable, say X, appearing in a role executed by a regular agent A, and the value of X is not determined by A (X first appears in the role in a message received by A), A must do one of the two kinds internal actions to X upon receiving it as follows. 1) A will make sure that $X \in A.init$, e.g., X is an agent name.; Or 2) A will check that $X \notin A.mem$, e.g., X should be treated a nonce freshly generated by some agent other than A. With this consideration the open problem described in theorem 1 is still undecidable.*

The proof of Theorem 2 is based on the proof of Theorem 1. The detailed proof is included in [17]. The idea is to create fresh copies of nonces which can encode counter values already reached in the computation, and then generate fresh copies of produced configuration terms where the terms encoding counter values are replaced by fresh and equivalent (in terms of number encoding) nonces. By organizing a bounded number of agents to execute role instances alternatively, an unbounded number of fresh copies of terms can be made.

4 Summary

We solve the open problem of Durgin, Lincoln and Mitchell [10] [11] using a direct reduction scheme from the reachability problem of 2-counter machines. We give a rigorous proof of correctness and carefully consider the assumptions and scenarios of the problem. This proof method is applicable beyond the above result. For example, with extended modeling and adaptation of the above reduction, we have proved other new and important undecidability results, including undecidability of checking secrecy for matching RO protocols with an attacker who is an insider.

References

1. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–207

2. Lowe, G.: An Attack on the Needham-Schroeder Public-Key Authentication Protocol. Inf. Process. Lett. **56**(3) (1995) 131–133
3. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In: TACAS. (1996) 147–166
4. Needham, R.M., Schroeder, M.D.: Using Encryption for Authentication in Large Networks of Computers. Commun. ACM **21**(12) (1978) 993–999
5. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: CSFW. (2001) 174–
6. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. Theor. Comput. Sci. **1-3**(299) (2003) 451–475
7. Amadio, R.M., Lugiez, D., Vanackère, V.: On the symbolic reduction of processes with cryptographic functions. Theor. Comput. Sci. **290**(1) (2003) 695–740
8. Comon, H., Cortier, V.: Tree automata with one memory set constraints and cryptographic protocols. Theor. Comput. Sci. **331**(1) (2005) 143–214
9. Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: Undecidability of bounded security protocols. In Heintze, N., Clarke, E., eds.: Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy. (july 1999)
10. Durgin, N.A., Lincoln, P., Mitchell, J.C.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security **12**(2) (2004) 247–311
11. Durgin, N.A.: Logical Analysis and Complexity of Security Protocols. PhD thesis, Computer Science Department, Stanford University (March 2003)
12. Even, S., Goldreich, O.: On the security of multi-party ping-pong protocols. In: IEEE Symposium on Foundations of Computer Science. (1983) 34–39
13. Ramanujam, R., Suresh, S.P.: Undecidability of secrecy for security protocols. Manuscript (July 2003)
14. Ferucio L. Tiplea and C. Enea and C. V. Birjoveanu: Decidability and complexity results for security protocols. In: Verification of Infinite-State Systems with Applications to Security, IOS Press (2006) 185–211
15. Froschle, S.: The insecurity problem: Tackling unbounded data, `http://homepages.inf.ed.ac.uk/sib/publ.html` To be published in 20^{th} IEEE Computer Security Foundations Symposium.
16. Ferucio L. Tiplea and C. Enea and C. V. Birjoveanu: Secrecy for bounded security protocols with freshness check is nexptime-complete. (2007) To be published in Journal of Computer Security.
17. Liang, Z., Verma, R.M.: Secrecy Checking of Protocols: Solution of an Open Problem. Technical report, `http://www.cs.uh.edu/preprint` (April 2007) Technical Report UH-CS-07-04.
18. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6**(1-2) (1998) 85–128
19. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: ACM Conference on Computer and Communications Security. (2001) 166–175
20. Thayer, F.J., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. Journal of Computer Security **7**(1) (1999)
21. Clark, J., Jacob, J.: A survey of authentication protocol literature: Version 1.0. Technical report (1997)
22. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computability. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)

Table 1. The complexity table provided by [10] of checking secrecy, with added explanation by us. In this table, role length and size of message instances in a run are assumed bounded. $\exists$ means nonce generation, $\neq$ means regular agents require uniqueness check on nonce instances, and disequality test is allowed, while $=$ means that uniqueness check and disequality test are not allowed.

		Bounded role instance num.	Unbounded role instance num.	
			Bounded total $\exists$ from regular agents	Unbounded total $\exists$ from regular agents
I with unbounded $\exists$	$\neq$	NPC	???	Undec.
	$=$	NPC	DEXPC	Undec.
I with no $\exists$	$\neq$	NPC	DEXPC	Undec.
	$=$	NPC	DEXPC	Undec.

A Understanding the open problem

We provide some descriptions of the open problem from [10] and [11]. Further discussion can be found in [17]

Table 1 shows the complexity results provided by [10] and [11] (Page 282 of [10] and Page 47 of [11]), with more explanations added by us. In [10] the authors focus on bounded security protocols, which means message numbers (role length) in a role is bounded, and only consider runs where message sizes are bounded.

We think the bound on the role length is not essential, since without it all the complexity results of [10] [11] are still true.

We think assuming "I with bounded $\exists$" instead of "I with no $\exists$" may not make the above table different, since it seems the proofs of [11] and [10] do not distinguish the two cases. Some descriptions are quoted here.

On page 48 of [11]: **"The series of ??? in the box at the top of column two indicates an unresolved question for the upper bound in the case of unbounded roles, bounded protocol existentials, and unbounded intruder existentials, when disequality tests are allowed."**

On page 259 of [10]: "We do not need to add a condition to test for equality, because it is expressible by matching the names of the variables in the terms."

On page 282 of [10]: "These rows are further subdivided into the cases where the roles can perform disequality tests which would allow them to determine whether two fresh values are different from each other. The $\neq$ row allows both equality and disequality tests, while the $=$ row allows only equality tests. In a protocol, a test for disequality on a nonce would mean the protocol compares a supposedly fresh nonce it receives against all the other nonces it has received, to make sure it is actually fresh. If disequality is not allowed, then this test is not performed." We think the word "protocol" should be replaced by "principal", since a protocol does not receive nonces, while a principal does.

On page 290 of [10]: "Computationally, the meaning of $\exists$ in $MSR_{\neq}$ is clear - each value generated by an $\exists$ is unequal to all others. We have not investigated the correspondence between logic and $MSR_{\neq}$."

Authority Analysis for Least Privilege Environments

Toby Murray and Gavin Lowe

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom
{toby.murray, gavin.lowe}@comlab.ox.ac.uk

Abstract. The rise of limited-privilege environments has been accompanied by the emergence of vulnerabilities in which a subject is able to maliciously wield their limited privileges to indirectly cause unwanted effects. Unfortunately, conventional safety analyses for access control systems are ill-equipped to deal with this problem because they do not detect the indirect effects that a subject can cause, but merely the permissions a subject can acquire.
We present a technique that characterises a subject's authority as all of the effects they can cause to occur. Our technique is based on an analysis of causation, applied to a CSP model of a system. These analyses can be expressed as CSP refinements and, hence, automatically performed by a refinement-checker such as FDR. We demonstrate the ability of our technique to successfully identify excess authority by examining the "Confused Deputy" scenario, whose vulnerability goes undetected with conventional safety analyses.

Key words: Access control, authority, causation, least privilege, formal definition, automatic verification, CSP, model checking, security.

1 Introduction

1.1 The Rise of Least Privilege

There appears to be a growing consensus that a failure to adhere to the principle of *least privilege* [25] has led current mainstream computing systems to be far less secure than they might have been. Executable email attachments are dangerous because, when opened by a user on current conventional operating systems, they are allowed to cause any effect that the user herself is allowed to cause. The severity of a remote code-execution vulnerability in a network-facing application is proportional to the privileges granted to that application.

Recently, many different solutions have been proposed and implemented in an attempt to rectify this situation. Some [19, 32, 12, 27, 11, 35] have provided startlingly clear examples of what the future might hold for secure computing if current applications and systems were reimplemented on top of architectures that naturally support the principle of least privilege. Other attempts [33, 3, 15, 20, 31, 26] have shown the challenges involved in trying to retrofit least privilege to the current mainstream computing base.

A common feature of all of these systems is the means to confine the set of *permissions* available to a running instance of an application. By permissions, we mean the set of objects in the system that the instance can access, or interact with, directly. In order to adhere to the principle of least privilege, instances are initially given minimal sets of permissions. While an instance is running it may acquire new permissions as its function alters. For example, when a user opens a new document in a word processor, the word processor might be granted the permission to read the file that contains the document. Indeed, in order to adhere to the principle of least privilege, an instance *must* be able to acquire new privileges as it is running. Otherwise, instances must be given the union of all permissions they might ever need, completely violating the principle of least privilege.

However, this leads to a potential problem. If instances can acquire new permissions, how can one be sure that a running instance cannot acquire a permission that it should not be allowed to have? For example, how can one be sure that a word processor won't be able to obtain the permission to write to the kernel binary?

This is an instance of the *safety problem* [7] for access control systems, which seeks to determine whether a particular subject can ever acquire a particular permission. Many formal models and decision procedures have been proposed [7, 1, 30, 9] in order to reason about this problem. When designing and analysing a limited-privilege system, it is imperative to be able to apply safety analyses in order to show that a running instance cannot obtain extra privileges in excess of the minimum required to perform its function at the current time.

1.2 Re-Enter An Old Attack

Unfortunately, as limited-privilege solutions are becoming more widely available, we are beginning to see classes of attack emerging that had previously been confined to the academic community [6]. These are attacks in which one subject s is able to use their permission to access another subject t to cause t to perform some action on s's behalf that violates the principle of least privilege.

The prototypical example [6] of this attack carries the name "Confused Deputy". In this attack, one subject, Alice, has access to another subject, Carol, a compiler. Alice has permission to invoke Carol with an output filename. Carol has permission to write to a special purpose billing file, Bill, in which Carol maintains a log of her own usage. By invoking Carol with the name of Bill, Alice can indirectly cause Bill to be overwritten, despite the fact that Alice does not have permission to write to Bill. Here Carol's permission is being used incorrectly on behalf of Alice.

A real-world example of this scenario is described by Spiessens [28]. Here, a user of a dynamic-firewall application grants network access to all instances of the ssh program, in order to allow her to connect securely to remote machines without having to click through a firewall dialogue each time. Unfortunately, in doing so, she has turned ssh into the ultimate confused deputy. Any other application that can execute ssh can now gain indirect access to an encrypted, authenticated channel to any remote host on the Internet. Among other things, this provides a useful path of egress for any piece of spyware on the system. Here ssh's permission is being used incorrectly on behalf of a malicious piece of spyware.

Another example [34] of this attack involves the User Account Control [33] (UAC) feature of Windows Vista. Under UAC, when an ordinary application tries to perform a sensitive function that requires administrator privileges, UAC displays a dialogue that may allow the user to grant the privileges to the application. Applications that are part of the operating system produce a dialogue labelled "Windows needs your permission to continue". The executable RunLegacyCPLElevated allows legacy dynamic libraries to be identified by UAC as part of the operating system. RunLegacyCPLElevated is invoked with a dynamic library filename as its argument, which it then executes on the invoker's behalf, thereby allowing the executing code to be identified as part of Windows. We argue that RunLegacyCPLElevated is a confused deputy. A subject, Dave, with the permission to execute RunLegacyCPLElevated and to write dynamic libraries to disk now has the indirect ability to cause arbitrary code to be executed that will be identified by UAC as part of Windows. Here RunLegacyCPLElevated's permission is being used incorrectly on behalf of Dave.

Current models for the analysis of the safety problem are ill-equipped to reason about the indirect effects that a subject can cause by use of its permissions. We refer to all such indirect effects as a subject's *authority*. Traditional safety analyses are limited to reasoning about authority in terms of the direct permissions a subject can acquire. As shown by the examples above, this can grossly underestimate a subject's total authority. As we shall demonstrate later, in the first example a simple safety analysis fails to reveal that Alice has authority to overwrite Bill, since Alice never has permission to overwrite Bill. Similarly, a simple safety analysis would fail to reveal the excess authority in the second and third examples as well.

These vulnerabilities highlight the importance of the principle of *least authority* [18, 17]. It is not enough to simply limit a subject's permissions in order to enforce meaningful least privilege. As we shall see later on, in the Confused Deputy scenario, limiting Alice's permissions to the smallest reasonable set still provides Alice with excess authority. In order to provide meaningful security, we require methods that can correctly analyse and detect excess authority once a subject's permissions have been minimised.

1.3 Contribution

In this paper, we present a technique specifically designed to reason about the indirect effects that a subject may be able to cause, in order to give an upper bound on the subject's authority. We use the process algebra CSP [21], and its stable-failures denotational semantics, to model and reason about causation and thereby define those actions that a subject may indirectly cause to occur.

CSP has been successfully applied to reasoning about many security-relevant systems, problems and properties including the safety problem in access control systems [10, 2], information flow [16], cryptographic protocols [23] and real-world security policies [22]. We begin with a brief overview of the syntax and semantics of CSP in Section 2.

In Section 3 we define authority in terms of all the events an object can cause to occur. Excess authority, then, is authority not allowed by the security policy. We give a straightforward definition of causation, which we call *Traces-Causation*. Unfortunately, this definition suffers from the refinement paradox: there are processes in which a particular event e cannot be caused by some object o, but that have refinements in which o can cause e to occur. We argue that this is undesirable, and that we should consider a definition of causation that holds whenever a process has a refinement for which Traces-Causation holds. We then give an alternative, equivalent, characterisation of causation, in terms of the traces and failures of the process.

In Section 4 we show how this property can be tested for using a model checker such as FDR [14]. In Section 5 we model the Confused Deputy scenario; we show how a simple safety analysis fails to detect the attack, but that our technique accurately detects Alice's authority to cause Bill to be overwritten. In Section 6 we sum up, compare and contrast our technique to related work, and consider some areas for future work.

2 A brief overview of CSP

In this section we give a brief overview of CSP. More details can be obtained elsewhere [21].

CSP is a process algebra, with various semantics, for describing and reasoning about concurrent systems. A system modelled in CSP comprises a set of concurrently executing *processes*. Each process usually models some particular component of the system in question. Processes execute by performing *events*. An event represents an atomic communication; this might either be between two processes or between a process and the environment. Processes communicate with each other by synchronising on common events. We write Σ for the set of all visible events.

2.1 Syntax

The process $STOP$ can perform no events. The process $a \rightarrow P$ can perform the event a, and then act like P. The process $?a : A \rightarrow P_a$ offers the set of events A; if a particular event a is performed, the process then acts like P_a. (The prefixing operator "$\rightarrow$" binds tighter than all other operators.)

CSP allows multi-part events where each part is separated by an infix dot ".", such as the event $\mathsf{up}.3$. The process $\mathsf{up}?a : A \rightarrow P_a$ initially offers the set of events $\{\mathsf{up}.a \mid a \in A\}$. The output operator "!" is used to offer specific events to the environment. The process $\mathsf{move}?x{:}X!3 \rightarrow P$ initially offers the set of events $\{\mathsf{move}.x.3 \mid x \in X\}$. The notation $\{|\mathsf{move}|\}$ denotes the set of events whose first part is move. The first part of a multi-part event is sometimes called a *channel*.

The process $P \;\Box\; Q$ represents an external choice between P and Q; the initial events of both processes are offered to the environment; when an event is performed, that resolves the choice. $P \sqcap Q$ represents an internal or nondeterministic choice between P and Q; the process can act like either P or Q, with the choice being made according to some criteria that we do not model. The process $P \lessdot b \gtrdot Q$ acts like P if the boolean condition b is true; otherwise it acts like Q.

The process $CHAOS_A$ is the most nondeterministic, nondivergent process with alphabet A; it can perform any sequence of events from A, and refuse any events.

If c is a channel, then $c.P$ represents the process that acts like P except every event x is renamed to $c.x$.

$P \underset{A}{\parallel} Q$ represents the parallel composition of P and Q, synchronising on events from A. For a set of processes, $\{P_1, \ldots, P_n\}$, and a set of alphabets $\{A_1, \ldots, A_n\}$ (each a subset of Σ), $\|_{1 \le i \le n}(P_i, A_i)$ represents the parallel composition of the P_i, where each P_i is allowed to perform events only from the set A_i, and all processes that share a common event must synchronise on it. $P \;|||\; Q$ represents the interleaving of P and Q, i.e., parallel composition with no synchronisation.

2.2 Semantics

A *trace* is a sequence of visible events that a process can perform. We write $traces(P)$ for the set of all traces of P.

We write traces within angle brackets ($\langle \ldots \rangle$). $s \,\hat{}\, t$ denotes the concatenation of s and t. $s \upharpoonright A$ denotes the trace obtained by taking s and removing all events not in A. $s \setminus A$ denotes the opposite: the trace obtained by taking s and removing all events in A: $s \setminus A = s \upharpoonright (\Sigma - A)$. Trace s is said to be a *prefix* of t, written $s \le t$, if there exists a sequence u, where $t = s \,\hat{}\, u$. Within traces, the special event $\checkmark$ represents termination, and can occur only at the end of a trace.

A *stable failure* is a pair (s, X), where s is some trace that P can perform, and X is a set of events that P can stably refuse after performing s: i.e., after s, P can reach a state where no internal activity is possible and none of the events from X is possible. We write $failures(P)$ for the set of all stable failures of P.

A *divergence* is a trace after which a process can diverge, i.e., perform an infinite amount of internal activity. We write $divergences(P)$ for the set of divergences of P. All of the processes we consider in this paper are *divergence free.* In this case the traces and stable failures are related by:

$$traces(P) = \{s \mid (s, X) \in failures(P)\}.$$

Within the failures-divergences semantics of CSP, a process is represented by its failures and its divergences. The following axioms hold for the failures F and divergences D of a process P.

F1. $traces(P) = \{t \mid (t, X) \in F\}$ is non-empty and prefix closed.
F2. $(v, X) \in F \wedge Y \subseteq X \Rightarrow (v, Y) \in F$.
F3. $(v, X) \in F \wedge v\hat{}\langle a\rangle \notin traces(P) \Rightarrow (v, X \cup \{a\}) \in F$.
F4. $v\hat{}(\surd) \in traces(P) \Rightarrow (v, \Sigma) \in F$.
D1. $s \in D \cap \Sigma^* \wedge t \in \Sigma^{*\surd} \Rightarrow s\hat{}t \in D$.
D2. $s \in D \Rightarrow (s, X) \in F$.
D3. $s\hat{}\langle\surd\rangle \in D \Rightarrow s \in D$.

We say that process Q refines process P, written $P \sqsubseteq Q$, when

$$failures(Q) \subseteq failures(P) \wedge divergences(Q) \subseteq divergences(P).$$

3 Characterising Authority

In this section, we produce a semantic characterisation of causation.

The systems we model comprise a set of *objects.* (By object, we mean both subjects and objects as defined in the standard access control literature: we make no distinction between the two.) Each object has a set of events that constitute its *alphabet.* These are the events that the object is able to partake in. Intuitively, if some object can perform some events from its alphabet that can, perhaps indirectly, cause some other event e to occur, then that object has authority to cause e.

We assume that the system is modelled by a CSP process. For simplicity, we restrict ourselves to divergence-free processes in this paper (i.e. those that can never perform an infinite amount of internal activity without communicating with their environment), and assume a finite alphabet.

3.1 Defining Causation

We base our understanding of causation on Lewis' counterfactual definition [13] that states that x causes y if y would be possible if x had occurred, but y would not be possible if x had not occurred. This leads to a simple definition for causation that can be applied to the traces of a process P. An object o with alphabet A can cause some event e to occur if there is some trace s after which e can follow, but when the events of A are removed from s, e cannot follow. We define the predicate $TC_P(A, e)$ (Traces-Causation) to capture this.

$$TC_P(A, e) \mathrel{\widehat{=}} \exists\, s \bullet s\hat{}\langle e\rangle \in traces(P) \wedge (s \setminus A)\hat{}\langle e\rangle \notin traces(P).$$

Often, we will be interested in the negation of this predicate, i.e. when the event e cannot be caused by any object with alphabet A. Thus, we define the predicate

$$NTC_P(A, e) \mathrel{\widehat{=}} \neg TC_P(A, e).$$

Unfortunately, this definition is too strong when applied to nondeterministic processes. Consider the process

$$P = a \rightarrow b \rightarrow STOP \sqcap b \rightarrow STOP,$$

and suppose the nondeterminism is resolved to the left. In this case, a can certainly cause b to occur; however, P doesn't satisfy the above definition of causation since both $\langle a, b\rangle$ and $\langle b\rangle$ are in $traces(P)$.

The problem here is that P is refined by processes, such as $Q = a \to b \to STOP$, in which a can cause b to occur. The refinements of P represent all of the possible ways in which nondeterminism in P can be resolved. As argued in [16], it is important to be able to detect when some property does not hold for a refinement of P, despite the fact that it does hold for P itself. P will often represent the design of a system, rather than its implementation; in this case, nondeterminism in P might represent underspecification which is to be resolved when the system is implemented. At other times, P might represent a model of a system in which nondeterminism is used to abstract away from low-level details of the real system in question. In both cases, it's important to be sure that, however the nondeterminism is resolved, the original property will still hold.

This realisation leads to the predicate $C_P(A, e)$ that holds precisely when P has a refinement for which $TC_P(A, e)$ holds:

$$C_P(A, e) \mathrel{\widehat{=}} \exists\, Q \bullet P \sqsubseteq Q \land TC_Q(A, e).$$

The predicate

$$NC_P(A, e) \mathrel{\widehat{=}} \neg C_P(A, e)$$

captures the negation of $C_P(A, e)$. Thus, NC is the *refinement-closure* of NTC:

$$NC_P(A, e) \equiv \forall\, Q \bullet P \sqsubseteq Q \Rightarrow NTC_Q(A, e).$$

The above property appears difficult to test, because of the quantification over all refinements of P. In order to derive a method for testing if a process satisfies NC, we give an alternative characterisation of causation. Here, an event e can be caused by an object o with alphabet A if there exists some trace of the system, in which o participates, after which e can follow; but when the events of A are removed, it's possible that e cannot follow, in the sense that e or an earlier event can be refused. We define the predicate $FC_P(A, e)$ (Failures-Causation) as follows:

$$\begin{aligned} FC_P(A, e) \mathrel{\widehat{=}} \exists\, s, t \bullet s \mathbin{\hat{}} t \mathbin{\hat{}} \langle e\rangle \in traces(P) \land s \restriction A \neq \langle\rangle \land \\ (s \setminus A, \{first(t \setminus A \mathbin{\hat{}} \langle e\rangle)\}) \in failures(P). \end{aligned}$$

Notice that the process P defined above does satisfy this definition of causation. We can see that $FC_P(\{a\}, b)$ holds by taking $s = \langle a\rangle$ and $t = \langle\rangle$, since $\langle a, b\rangle \in traces(P)$ and $(\langle\rangle, \{b\}) \in failures(P)$.

We will show, below, that $C_P(A, e) \equiv FC_P(A, e)$. We begin by showing that, under FC, non-causation is refinement-closed.

Lemma 1. *If $\neg FC_P(A, e)$ and $Q \sqsupseteq P$ then $\neg FC_Q(A, e)$.*

Proof. Suppose, for a contradiction, that $\neg FC_P(A, e)$, $Q \sqsupseteq P$ and $FC_Q(A, e)$. Then for some s, t:

$$s \mathbin{\hat{}} t \mathbin{\hat{}} \langle e\rangle \in traces(Q) \land s \restriction A \neq \langle\rangle \land (s \setminus A, \{first(t \setminus A \mathbin{\hat{}} \langle e\rangle)\}) \in failures(Q).$$

But $traces(Q) \subseteq traces(P) \wedge failures(Q) \subseteq failures(P)$ so

$$s \,\hat{}\, t \,\hat{}\, \langle e \rangle \in traces(P) \wedge s \restriction A \neq \langle\rangle \wedge (s \setminus A, \{first(t \setminus A \,\hat{}\, \langle e \rangle)\}) \in failures(P),$$

contradicting $\neg FC_P(A, e)$. $\square$

We now show that for divergence-free processes, Traces-Causation implies Failures-Causation.

Lemma 2. *If Q is non-divergent and $TC_Q(A, e)$ then $FC_Q(A, e)$.*

Proof. Consider a divergence-free process Q for which $TC_Q(A, e)$ holds. Then there is some trace s such that

$$s \,\hat{}\, \langle e \rangle \in traces(Q) \wedge (s \setminus A) \,\hat{}\, \langle e \rangle \notin traces(Q).$$

Thus $s \neq s \setminus A$, so $s \restriction A \neq \langle\rangle$. Partition s about its first event a from A, into the sequence $t \,\hat{}\, \langle a \rangle \,\hat{}\, u$ so

$$s = t \,\hat{}\, \langle a \rangle \,\hat{}\, u \wedge a \in A \wedge t \restriction A = \langle\rangle.$$

Then

$$s \,\hat{}\, \langle e \rangle = t \,\hat{}\, \langle a \rangle \,\hat{}\, u \,\hat{}\, \langle e \rangle \in traces(Q) \wedge (s \setminus A) \,\hat{}\, \langle e \rangle = t \,\hat{}\, (u \setminus A) \,\hat{}\, \langle e \rangle \notin traces(Q).$$

Now, $t \in traces(Q)$, so let v be the longest prefix of u such that $t \,\hat{}\, (v \setminus A) \in traces(Q)$, and let w be the remainder of u:

$$v \,\hat{}\, w = u \wedge t \,\hat{}\, (v \setminus A) \in traces(Q) \wedge t \,\hat{}\, (v \setminus A) \,\hat{}\, \langle first(w \setminus A \,\hat{}\, \langle e \rangle) \rangle \notin traces(Q).$$

Now Q is divergence-free, so $(t \,\hat{}\, (v \setminus A), \{\}) \in failures(Q)$, and hence by Axiom **F3** (see Section 2),

$$(t \,\hat{}\, (v \setminus A), \{first(w \setminus A \,\hat{}\, \langle e \rangle)\}) \in failures(Q).$$

Combining the above results we have

$$\begin{array}{l} t \,\hat{}\, \langle a \rangle \,\hat{}\, v \,\hat{}\, w \,\hat{}\, \langle e \rangle \in traces(Q) \wedge \\ ((t \,\hat{}\, \langle a \rangle \,\hat{}\, v) \setminus A, \{first(w \setminus A \,\hat{}\, \langle e \rangle)\}) \in failures(Q), \end{array}$$

and hence $FC_Q(A, e)$ holds. $\square$

We now use these lemmas to prove that Causation and Failures-Causation are equivalent:

Theorem 1. *For any divergence-free process, P:*

$$C_P(A, e) \equiv FC_P(A, e).$$

Proof. We begin by showing that $C_P(A,e) \Rightarrow FC_P(A,e)$. So suppose that $C_P(A,e)$, i.e., that there exists $Q \sqsupseteq P$ such that $TC_Q(A,e)$. Then Q must also be divergence-free, so by Lemma 2, $FC_Q(A,e)$ holds. Hence, by Lemma 1, $FC_P(A,e)$ holds.

Conversely, suppose $FC_P(A,e)$ holds. Then for some s and t:

$$\begin{array}{l} s\,\hat{}\,t\,\hat{}\,\langle e\rangle \in \mathit{traces}(P) \wedge s \upharpoonright A \neq \langle\rangle \wedge (s \setminus A, \{c\}) \in \mathit{failures}(P), \\ \text{where } c = \mathit{first}(t \setminus A\,\hat{}\,\langle e\rangle). \end{array}$$

We construct a process Q that refines P and such that $TC_Q(A,e)$. Define Q to be the divergence-free process that has the same failures as P, except with all those corresponding to the trace $s \setminus A\,\hat{}\,\langle c\rangle$ removed:

$$\begin{aligned} \mathit{failures}(Q) = \mathit{failures}(P) - {} & \\ & \{(s \setminus A\,\hat{}\,\langle c\rangle\,\hat{}\,t, X) \mid t \in \Sigma^*, X \subseteq \Sigma\} - \\ & \{(s \setminus A, X) \mid (s \setminus A, X \cup \{c\}) \notin \mathit{failures}(P)\}. \end{aligned}$$

Lemma 3, which follows, shows that such a process exists.

Observe that $P \sqsubseteq Q$ since $\mathit{failures}(Q) \subseteq \mathit{failures}(P)$.

Now, Q is divergence-free, so $\mathit{traces}(Q) = \{v \mid (v, X) \in \mathit{failures}(Q)\}$. Hence

$$s\,\hat{}\,t\,\hat{}\,\langle e\rangle \in \mathit{traces}(Q),$$

since this is not one of the traces removed. Also, $s \setminus A\hat{}\langle c\rangle \leq (s \setminus A)\hat{}(t \setminus A)\hat{}\langle e\rangle = (s\,\hat{}\,t) \setminus A\,\hat{}\,\langle e\rangle$ so

$$(s\,\hat{}\,t) \setminus A\,\hat{}\,\langle e\rangle \notin \mathit{traces}(Q).$$

Hence, $TC_Q(A,e)$ holds. □

Finally, we must show that the process Q constructed in Theorem 1 exists. We will need the following result from [21, Section 9.3]:

Theorem 2. *Assuming the alphabet Σ is finite, for any choice of (F, D) that satisfies the axioms (see Section 2) of the failures-divergences model of CSP, there is a CSP process Q whose failures and divergences are F and D respectively.*

Hence it will be enough to show that $\mathit{failures}(Q)$ and $\mathit{divergences}(Q)$ satisfy the axioms.

Lemma 3. *$\mathit{failures}(Q)$, as defined in Theorem 1, and $\mathit{divergences}(Q) = \{\}$ satisfy the axioms* **F1**–**F4** *and* **D1**–**D3** *of the failures-divergences model.*

Proof. Observe that since $\mathit{divergences}(Q) = \{\}$, Axioms **D1**–**D3** hold trivially. We consider each of the remaining axioms in turn. Note that

$$\mathit{traces}(Q) = \mathit{traces}(P) - \{s \setminus A\,\hat{}\,\langle c\rangle\,\hat{}\,t \mid t \in \Sigma^*\}.$$

Axiom F1. Clearly $\mathit{traces}(Q)$ is non-empty: it contains, at least, the empty trace. It is prefix-closed since $\mathit{traces}(P)$ is, and we remove an extensions-closed set of traces.

Axiom F2. Q satisfies **F2** since P does, and whenever we remove a failure, we remove all failures with larger refusal sets.

Axiom F3. Suppose $(v, X) \in failures(Q)$ and $v \,\hat{}\, \langle a \rangle \notin traces(Q)$. Then $(v, X) \in failures(P)$. We perform a case analysis.

- Case $v \,\hat{}\, \langle a \rangle \neq s \setminus A \,\hat{}\, \langle c \rangle$. Then $v \,\hat{}\, \langle a \rangle \notin traces(P)$, and so $(v, X \cup \{a\}) \in failures(P)$, since P satisfies **F3**. And hence $(v, X \cup \{a\}) \in failures(Q)$, by construction.
- Case $v = s \setminus A \wedge a = c$. Recall that $(s \setminus A, \{c\}) \in failures(P)$. Hence $(s \setminus A, X \cup \{c\}) \in failures(P)$, by Axiom **F2**. And hence $(v, X \cup \{a\}) = (s \setminus A, X \cup \{c\}) \in failures(Q)$, by construction.

Axiom F4. Suppose $v \,\hat{}\, \langle \surd \rangle \in traces(Q)$. Then $v \,\hat{}\, \langle \surd \rangle \in traces(P)$ and $s \setminus A \,\hat{}\, \langle c \rangle \not\leq v$. Then $(v, \Sigma) \in failures(P)$ since P satisfies **F4**. Hence $(v, \Sigma) \in failures(Q)$, by construction, whether or not v equals $s \setminus A$. □

4 Testing for Authority

We now construct a refinement test, which can be automatically carried out by a model checker such as FDR [14], that checks whether $NC_P(A, e)$ holds, i.e., that checks that an object with alphabet A cannot cause e to occur. Note that we restrict ourselves to finite-state processes, where this question is decidable. We generalise the test from a single event e to a set of events B, where $A \cap B = \{\}$: we define $NC_P(A, B) \mathrel{\widehat{=}} \forall\, e \in B \bullet NC_P(A, e)$.

The test works as follows. We run two copies of P in parallel, in a harness, with a controller (or scheduler). Initially, we allow only the left-hand copy of P to perform A events, and force both copies to do the same non-A events. At some point, after the left-hand copy has done at least one event from A, and has just performed some event $c \notin A$, we pause the right-hand copy of P. At this point, the left-hand copy will have performed some trace $s \,\hat{}\, \langle c \rangle$ with $s \upharpoonright A \neq \langle\rangle$ and the right-hand copy will have performed $s \setminus A$. Following the definition of FC, we continue to run the left-hand copy until it has performed a trace $s \,\hat{}\, t \,\hat{}\, \langle e \rangle$, for some $e \in B$ and trace t; since $c \notin A$ we will have $c = first(t \setminus A \,\hat{}\, \langle e \rangle)$. At this point we restart the right-hand copy of P and test whether it can refuse the event c; if so, $FC_P(A, B)$ holds.

The events of the left- and right-hand copies of P are distinguished by using a renaming transformation that has each copy perform its events on separate fresh channels, left and right. The harness in which the two copies of P are run with the controller is defined as follows.

$$Harness(P) = (\mathsf{left}.P \;|||\; \mathsf{right}.P) \underset{\{|\mathsf{left},\mathsf{right}|\}}{\|} Ctrl1.$$

Here, $\mathsf{left}.P$ ($\mathsf{right}.P$) denotes the process that performs the event $\mathsf{left}.x$ ($\mathsf{right}.x$) whenever P performs x. The controller process, $Ctrl1$, is defined as follows.

$$\begin{aligned}
Ctrl1 &= \mathsf{left}?c \to (Ctrl2 \lhd c \in A \rhd \mathsf{right}.c \to Ctrl1),\\
Ctrl2 &= \mathsf{left}?c \to (Ctrl2 \lhd c \in A \rhd (\mathsf{right}.c \to Ctrl2 \sqcap \mathsf{ping} \to Ctrl3(c)))\,,\\
Ctrl3(c) &= Ctrl5(c) \lhd c \in B \rhd Ctrl4(c),\\
Ctrl4(c) &= \mathsf{left}?d \to (Ctrl5(c) \lhd d \in B \rhd Ctrl4(c)),\\
Ctrl5(c) &= \mathsf{ping} \to \mathsf{right}.c \to STOP.
\end{aligned}$$

The controller initially forces the right-hand copy of P to perform the same events as the left-hand copy, until the latter performs an event from A. The controller then (in state $Ctrl2$) continues to force the right-hand copy to perform the same non-A events as the left-hand copy, except after a non-A event c it can (nondeterministically) choose to pause the right-hand copy, signalled by the event ping. It then continues to run the left-hand copy until it performs an event from B; if c itself is in B, then this is immediate (states $Ctrl3$ and $Ctrl4$). The right-hand copy is then re-awoken (in state $Ctrl5$), also signalled by the event ping, in order to test whether it can refuse c.

$Spec1$ is the most general process that mirrors the behaviour of the harness, except that it never refuses the final event $\mathsf{right}.c$.

$$\begin{aligned}
Spec1 &= \mathsf{left}?c \to (Spec2 \lhd c \in A \rhd (\mathsf{right}.c \to Spec1 \sqcap STOP)) \sqcap STOP,\\
Spec2 &= \mathsf{left}?c \to\\
&\qquad\quad (Spec2 \lhd c \in A \rhd (\mathsf{right}.c \to Spec2 \sqcap \mathsf{ping} \to Spec3(c) \sqcap STOP))\\
&\qquad \sqcap STOP,\\
Spec3(c) &= Spec5(c) \lhd c \in B \rhd Spec4(c),\\
Spec4(c) &= \mathsf{left}?d \to (Spec5(c) \lhd d \in B \rhd Spec4(c)) \sqcap STOP,\\
Spec5(c) &= \mathsf{ping} \to \mathsf{right}.c \to STOP.
\end{aligned}$$

The states of the specification correspond to the states of the controller. Notice that $Spec5$ cannot refuse to perform $\mathsf{right}.c$. Thus, $Harness(P)$ will refine $Spec1$ if and only if the right-hand copy of P can never refuse the final c event, i.e., if and only if $NC_P(A, B)$ holds. Thus

$$Spec1 \sqsubseteq Harness(P) \;\equiv\; NC_P(A, B).$$

The refinement can be tested using a model checker like FDR. If the refinement fails, FDR will produce a counter-example; the ping events in the counter-example mark the points at which the right-hand copy of P was paused and restarted, and hence aid in its interpretation.

If P has N states then the size of $Harness(P)$ is $O(N^2)$, since it runs two copies of P. In most cases, however, the size of $Harness(P)$ should be significantly less than $O(N^2)$. This is because for each state of the first copy of P, there is likely to be a fairly small number of states that the second copy of P can be in at the same time. If this number is bounded by some constant k, then the total number of states is $O(k.N)$. Hence, in most cases, the time to perform the test should grow linearly with the size of P.

5 Analysing The Confused Deputy

Having described and explained our technique, we now demonstrate its utility for reasoning about Alice's excess authority in the Confused Deputy scenario described in Section 1. First, we model the system in CSP. We then show how a simple safety analysis fails to detect Alice's authority to overwrite Bill, before demonstrating how to accurately detect Alice's excess authority using a refinement check of the sort described in Section 4.

5.1 Modelling the Scenario in CSP

We define a set of operations $Op = \{\mathsf{Read}, \mathsf{Write}, \mathsf{Append}, \mathsf{Exec}\}$ and a set of objects $Object = \{\mathsf{Alice}, \mathsf{Bill}, \mathsf{Carol}\}$[1]. We then define events of the form $o_1.o_2.op$ to represent object o_1 performing operation op on object o_2. The events associated with an Exec operation also carry an object name, and are of the form $o_1.o_2.\mathsf{Exec}.arg$, representing object o_1 executing object o_2, passing the argument arg. Thus we use alphabet

$$\{o_1.o_2.op \mid o_1, o_2 \in Object \wedge op \in \{\mathsf{Read}, \mathsf{Write}, \mathsf{Append}\}\} \cup \\ \{o_1.o_2.\mathsf{Exec}.arg \mid o_1, o_2 \in Object \wedge arg \in Object\}.$$

An object o is involved in events that represent it operating on some other object p, and events that represent p operating on it. Hence, the alphabet of each $o \in Object$ is defined as:

$$\alpha(o) = \{\!|o.p \mid p \in Object - \{o\}|\!\} \cup \{\!|p.o \mid p \in Object - \{o\}|\!\}.$$

Notice that the definition of α is such that an operation is only defined between two distinct objects.

The configuration of permissions is defined by the acl function, which takes an object and an operation and returns the set of objects who have permission to perform that operation on that object: Carol has permission to write and append to Bill; Alice has permission to execute Carol.

$$\begin{array}{ll} acl(\mathsf{Bill}, \mathsf{Write}) = \{\mathsf{Carol}\}, & acl(\mathsf{Bill}, \mathsf{Append}) = \{\mathsf{Carol}\}, \\ acl(\mathsf{Carol}, \mathsf{Exec}) = \{\mathsf{Alice}\}, & acl(other, other) = \{\}. \end{array}$$

We define a set of parameterised CSP processes that represent the behaviour of different types of entities within the system.

The $Compiler$ process defines the behaviour of a compiler, with identity me, that is able to be executed by any object with Exec permission as defined by its access control list. Once invoked, it writes to the specified file, before appending to its billing file, $logFile$.

$$\begin{aligned} Compiler(me, logFile) = {} & ?s : acl(me, \mathsf{Exec})!me!\mathsf{Exec}?file \rightarrow \\ & me.file.\mathsf{Write} \rightarrow \\ & me.logFile.\mathsf{Append} \rightarrow Compiler(me, logFile). \end{aligned}$$

[1] Recall that we use the term "object" to include what are termed "subjects" in some of the access control literature.

The *File* process defines the behaviour of a file, with identity *me*, that can be written, appended or read by anyone with the appropriate permission.

$$\begin{aligned} File(me) = {} & ?s : acl(me, \mathsf{Write})!me!\mathsf{Write} \rightarrow File(me) \,\Box \\ & ?s : acl(me, \mathsf{Append})!me!\mathsf{Append} \rightarrow File(me) \,\Box \\ & ?s : acl(me, \mathsf{Read})!me!\mathsf{Read} \rightarrow File(me). \end{aligned}$$

The *User* process defines the behaviour of a user, with identity *me*, who tries to execute any program they can, and to read, write and append to any file they can. In this manner, we capture the most general behaviour of any user within our model.

$$\begin{aligned} User(me) = {} & me?prog!\mathsf{Exec}?arg \rightarrow User(me) \,\Box \\ & me?file!\mathsf{Read} \rightarrow User(me) \,\Box \\ & me?file!\mathsf{Write} \rightarrow User(me) \,\Box \\ & me?file!\mathsf{Append} \rightarrow User(me). \end{aligned}$$

The total system, *System*, is then the parallel composition of $User(\mathsf{Alice})$, $File(\mathsf{Bill})$ and $Compiler(\mathsf{Carol}, \mathsf{Bill})$, with the above alphabets.

5.2 A Simple Safety Analysis

We can perform a simple safety analysis [7] to determine whether Alice can ever obtain permission to overwrite Bill. This example uses static access control lists, so clearly it is impossible for Alice to obtain permission to overwrite Bill. However, in more complex examples, the configuration of permissions can change over time, and so a safety analysis of the sort presented here is necessary to determine whether a particular subject can ever acquire a particular permission.

In our example, were Alice able to obtain any permission to Bill, then *System* would be able to perform some event in $\{|\mathsf{Alice.Bill}|\}$. We can test whether *System* is ever able to perform such an event by testing if it refines the most general process that performs no such event:

$$Spec = CHAOS_{\Sigma - \{|\mathsf{Alice.Bill}|\}}.$$

FDR indicates that $Spec \sqsubseteq_T System$. As we expect, this simple safety analysis reveals that Alice can never gain permission to overwrite Bill.

5.3 An Authority Analysis

We now analyse whether Alice has authority to cause Bill to be overwritten, i.e., if she can cause some event from $B = \{o.\mathsf{Bill.Write} \mid o \in Object\}$. We can check that Alice has no such authority by testing the following refinement (with $A = \alpha(\mathsf{Alice})$):

$$Spec1 \sqsubseteq Harness(System)$$

FDR completes the test in under a second and indicates that this refinement does not hold. It provides the failure

$$(\langle \mathsf{left.Alice.Carol.Exec.Bill}, \mathsf{left.Carol.Bill.Write}, \mathsf{ping}, \mathsf{ping}\rangle, \{\mathsf{right.Carol.Bill.Write}\})$$

as a counter-example. As expected, the refinement-check reveals that by invoking Carol with the name Bill, Alice can cause Bill to be overwritten (in the left-hand copy), since Carol can refuse to write to Bill if not so invoked (in the right-hand copy).

6 Discussion

The rise of least privilege environments necessitates techniques for formal analysis that can accurately reason about a subject's authority, beyond the set of permissions they can acquire. In particular, the emergence of instances of the Confused Deputy vulnerability demonstrates the need to be able to detect a subject's excess authority in spite of their minimal privileges. Safety analyses are ill-equipped for this task because they are limited to characterising authority in terms of acquirable permissions. As shown, this can grossly underestimate a subject's total authority.

We have presented a technique based on an analysis of causation for reasoning about authority in the presence of least-privilege. We have demonstrated its utility for detecting a subject's excess authority in the Confused Deputy scenario. We hope that as least privilege environments become pervasive, that such analyses will become as important as safety and information-flow analyses are today, in order to ensure that the principle of least authority is upheld.

In the remainder of this section we discuss some related work and prospects for extending the work of this paper.

6.1 Analysing Authority in Capability Systems

In the Confused Deputy example, it is interesting to consider how Alice might be prevented from having authority to overwrite Bill. One solution might be for Carol to check the filename she is passed when invoked and to not write to this file if it is Bill. However this raises the question: what if the subject that executes Carol has permission to write to Bill? Should Carol then write to Bill on that subject's behalf? Unfortunately, as noted by Spiessens [28], Carol does not have the information available to make this determination. Even if she can examine the access control list for Bill, it's possible that Alice is executing her on behalf of some other subject who does have the relevant permission.

When the Confused Deputy was first described [6], it was noted that this problem largely disappears when considered within the context of an access control system that unifies designation and permission, such as those based on capabilities [4]. If Alice can designate Bill to Carol if and only if she herself has permission to Bill, then Carol will write to Bill if and only if Alice has permission to write to Bill, since Carol uses Alice's designation when attempting to write the output. Hence, another possible remedy would be to abandon the use of identity-based access controls, such as the access control lists modelled in the example, in favour of a capability-based approach.

Capability systems are interesting not only because they elegantly solve the Confused Deputy problem. They also naturally support the construction of systems that adhere to the principle of least authority. Further, many of the common abstractions used in capability systems are designed to provide one object with authority to access another, but not direct permission. Hence, being able to reason about authority is crucial for an understanding of many of these abstractions. For these reasons, capability systems present an attractive target for the application of our techniques.

6.2 Pseudo-Permissions in Safety Analyses

Previous attempts to model and reason about authority include efforts based on models for safety-analysis in which *pseudo-permissions* are used to model authority. One such case is the use of *de-facto rights* in Take-Grant systems [1]. For example, a subject s with read permission to subject t, where t is writable by subject r, has *de-facto* read permission to subject r. Unfortunately, the use of de-facto rights fails to take into account the influence that t's behaviour has on s's authority [28]. For example, t might behave in such a way as to prevent any information flowing from r to s, in which case s has no authority to read r. Thus, the use of de-facto rights necessarily over-estimates a subject's total authority. More generally, using pseudo-permissions to model aspects of authority requires specific knowledge about how various permissions may interact with one another in order to give rise to authority.

In contrast, our technique does not rely on any specific knowledge of the interaction between permissions, but rather extracts the causal relationships between events from the semantics of a process. Our approach also allows the restrictive behaviour of a subject to be explicitly modelled via appropriate CSP process definitions. For example, by altering the definition of the *Compiler* process, we could redefine Carol's behaviour in the Confused Deputy example to not write to the file designated by Alice if that file is Bill. An authority analysis would reveal that Carol's restricted behaviour reduces Alice's authority, preventing her from being able to cause Carol to overwrite Bill.

6.3 Non-Interference as the Absence of Authority

The property of *non-interference* [5] and related notions of information-flow can be viewed as characterising the absence of any causal flow from High subjects to Low subjects [24]. We believe that there is a connection between the absence of causal flow and the absence of authority, as defined in this paper. Intuitively, when applying our technique, one subject, High, has no authority over another, Low, if High can never cause some event $l \in \alpha(\mathsf{Low})$ to occur. However, a lack of information flow requires not only that High is unable to cause any event in $\alpha(\mathsf{Low})$ from occurring, but also that High is unable to cause any event in $\alpha(\mathsf{Low})$ from *not* occurring.

We intend to investigate further the relationship between our work and that on information flow; in particular, there seem to be strong similarities with the work of [16].

6.4 Knowledge-Behaviour Models

The work of Spiessens *et al.* [28] on Knowledge-Behaviour Models (KBMs) and the SCOLL language seeks to reason about authority by explicitly taking into account the possible restrictive effects that a subject's behaviour can have on another's authority. This work directly inspired our work on using causal analyses to characterise authority.

Here a subject's permissions and behaviours are represented by a set of predicates, with rules that define how new predicates can be derived from the current set. SCOLL has been applied to reason about indirect authority; however, many examples [28, 8, 29] measure authority in terms of acquirable permissions. As noted in Section 1 and demonstrated by the analysis of the Confused Deputy example in Section 5, this can greatly underestimate a subject's total authority.

Despite this limitation, KBMs have been used to accurately model the Confused Deputy scenario [28]. In order to model the scenario, a behaviour predicate *useForClient* is introduced to describe Carol's intent to use some designation on Alice's behalf. Testing whether *useForClient(*Bob*)* is derivable accurately detects whether Carol is able to use the designation Bob on Alice's behalf.

In this sense, the $useForClient$ predicate can be viewed as a means to characterise part of Alice's authority, independently of the permissions Alice can acquire. Unfortunately, this approach requires the incorporation of predicates, like *useForClient*, that explicitly capture the notion of on whose behalf a subject might be acting, in order to reason about authority. In contrast, our approach requires no extra work to be undertaken, since it automatically determines on whose behalf a subject might be acting.

References

1. Matt Bishop and Lawrence Snyder. The transfer of information and authority in a protection system. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, pages 45–54. ACM Press, 1979.
2. Jeremy Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35. ACM Press, 2005.
3. Bruno Castro da Silva and Raul Fernando Weber. TuxGuardian: Um firrewall de host voltado para o usuário final. In *5 Fórum Internacional de Software Livre*, 2004. Available at: `http://tuxguardian.sourceforge.net/tg-sbrc.pdf`.
4. Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–154, March 1966.
5. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy 1982*, pages 11–20, 1982.
6. Norm Hardy. The confused deputy (or why capabilities might have been invented). *Operating Systems Review*, 22(4):36–38, October 1988.

7. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 1976.
8. Yves Jaradin, Fred Spiessens, and Peter Van Roy. SCOLL: A language for safe capability based collaboration. Technical Report Research Report INFO-2005-10, Université catholique de Louvain, 2005.
9. Eldar Kleiner and Tom Newcomb. On the decidability of the safety problem for access control policies. In *Sixth International Workshop on Automatic Verification of Critical Systems (AVoCS 2006)*, pages 91–103, 2006.
10. Eldar Kleiner and Tom Newcomb. Using CSP to decide safety problems for access control policies. Technical Report Research Report RR-06-04, Oxford University Computing Laboratory, University of Oxford, January 2006.
11. Ivan Kristić. System security on the One Laptop per Child's XO laptop: The Bitfrost security platform, 2007. Available at: `http://wiki.laptop.org/go/Bitfrost`.
12. Maxwell Krohn, Petros Efstathopoulos, Cliff Frey, Frans Kaashoek, Eddie Kohler, David Mazières, Robert Morris, Michelle Osborne, Steve VanDeBogart, and David Ziegler. Make least privilege a right (not a privilege). In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, June 2005.
13. David Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, 1973.
14. Formal Systems (Europe) Limited. Failures divergences refinement: FDR2 user manual, 2005. Available at: `http://www.fsel.com/documentation/fdr2/fdr2manual.ps`.
15. Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)*, 2001.
16. Gavin Lowe. On information flow and refinement-closure. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '07)*, 2007.
17. Mark S. Miller and Jonathan S. Shapiro. Paradigm regained: Abstraction mechanisms for access control. In *Proceedings of the 8th Asian Computing Science Conference (ASIAN03)*, pages 224–242, December 2003.
18. Mark S. Miller, Bill Tulloh, and Jonathan S. Shapiro. The structure of authority: Why security is not a separable concern. In *Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Revised Selected and Invited Papers, LNCS 3389*, pages 2–20. Springer, 2005.
19. Mark Samuel Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.* PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.
20. David S. Peterson, Matt Bishop, and Raju Pandey. A flexible containment mechanism for executing untrusted code. In *Proceedings of the 11th USENIX Security Symposium*, 2002.
21. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, Upper Saddle River, NJ, USA, 1997.
22. Peter Ryan and Ragni Ryvold Arnesen. A process algebraic approach to security policies. In *DBSec*, pages 301–312, 2002.
23. Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols: the CSP Approach.* Addison Wesley, 2000.
24. Peter Y. A. Ryan. Mathematical models of computer security. In R. Gorrieri, editor, *Proceedings of the 2000 FOSAD Summer School, LNCS 2171.* Springer, 2000.

25. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1208–1308, September 1975.
26. Mark Seaborn. Plash: tools for practical least privilege, 2007. Available at: `http://plash.beasts.org`.
27. Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: A fast capability system. In *SOSP '99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 170–185, 1999.
28. Alfred Spiessens. *Patterns of Safe Collaboration.* PhD thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, February 2007.
29. Fred Spiessens, Yves Jaradin, and Peter Van Roy. SCOLL and SCOLLAR: Safe collaboration based on partial trust. Technical Report Research Report INFO-2005-12, Université catholique de Louvain, 2005.
30. Fred Spiessens and Peter Van Roy. A practical formal model for safety analysis in capability-based systems. In *Trustworthy Global Computing, International Symposium, TGC 2005, Revised Selected Papers, LNCS 3705*, pages 248–278. Springer, 2005.
31. Marc Stiegler, Alan H. Karp, Ka-Ping Yee, Tyler Close, and Mark S. Miller. Polaris: Virus safe computing for Windows XP. *Communications of the ACM*, 49(9):83–88, September 2006. Available at: `http://www.hpl.hp.com/techreports/2004/HPL-2004-221.html`.
32. Marc Stiegler and Mark S. Miller. A capability based client: The DarpaBrowser. Technical Report Focused Research Topic 5 / BAA-00-06- SNK, Combex, Inc., June 2002. Available at: `http://www.combex.com/papers/darpa-report/index.html`.
33. Microsoft TechNet. User Account Control. Microsoft Corporation, 2007. Available at: `http://technet.microsoft.com/en-us/windowsvista/aa905113.aspx`.
34. Ollie Whitehouse. An example of why UAC prompts in Vista can't always be trusted. posted to Symantec Security Risks Weblog, February 20, 2007 05:00 AM, 2007. Available at: `http://www.symantec.com/enterprise/security_response/weblog/2007/02/an_%example_of_why_uac_prompts.html`.
35. Ka-Ping Yee. Aligning security and usability. *IEEE Security and Privacy*, 2(5):48–55, September/October 2004.

A Security Analysis on Diffie-Hellman Key Exchange against Adaptive Adversaries using Task-Structured PIOA

Kazuki Yoneyama, Yuichi Kokubun, and Kazuo Ohta

The University of Electro-Communications,
1-5-1, Chofugaoka, Chofu-shi, Tokyo, Japan.
yoneyama@ice.uec.ac.jp

Abstract. The task-structured probabilistic I/O automata (task-PIOA) framework allows formal analysis of cryptographic primitives in a computational model, and considering probabilistic and non-deterministic behaviors. However, case study of analysis on cryptographic primitives is only an oblivious transfer protocol. Moreover, the way to formulate adversaries which carry out adaptive corruption is not known since the adversary model is restricted to static model in previous analysis. In this paper, we analyze the security of Diffie-Hellman key exchange against adaptive adversaries in task-PIOA framework. Firstly, we reformulate the definition of DDH assumption in the task-PIOA framework and prove the equivalence between the original definition and our reformulated one. Then, we define the key exchange functionality in the task-PIOA framework, which is based on the definition in Universal Composability framework. Finally, we construct real and ideal Diffie-Hellman key exchange systems, and prove that real system realizes the key exchange functionality against passive and adaptive adversaries.

keywords: formal method, task-PIOA, Diffie-Hellman key exchange, DDH assumption, adaptive adversary

1 Introduction

An analysis on cryptographic protocols is typically complex, and thus in many cases it is error-prone. As a solution for this, the technique of formal method is effective as it has some advantages over the handwritten analysis, the most remarkable one is that we are able to analyze protocols automatically. The most famous approach is Dolev-Yao's symbolic model [1]. This approach is mainly interested in authentication and confidentiality properties, and has been used in the analysis of several practical protocols, e.g., SSH, TLS, Kerberos. However, the basic symbolic analysis does not a priori carry any cryptographic *soundness* guarantees.

Thus, recently, formal analysis frameworks that can grasp soundness property are increasingly paid attention to. Canetti and Herzog proposed combining the symbolic model and the universally composable (UC) security framework [2] in order to create a unified framework for proving security of protocols, they named universally composable symbolic analysis (UCSA) [3]. This framework enables a security analysis that is completely symbolic, and at the same time cryptographically sound with strong composability property. However, it is uncleared whether this framework enables to realize several cryptographic protocols without dealing with cryptographic primitives, e.g., public-key encryption, as symbolic operations. Also, though participants in the protocol are modeled by Interactive Turing Machines (ITMs), a complete analysis of protocols modeled by ITMs is impractical because it includes too many low-level machine details.

Based on this result, the task-structured Probabilistic I/O Automata (task-PIOA) framework was proposed by Canetti et al. [4–6]. It models the participants in a protocol by slight variant of PIOAs that enable to carry out a proof on protocols formally, at a high level of abstraction. Indeed, in the computational cryptography community, since protocols are typically described using an informal high-level language, and proof sketches are given in terms of the informal protocol descriptions, using PIOA is suitable. However, since the concrete example of analysis is made only on an oblivious transfer protocol [6] against passive and static adversary, then there are unknown parts about the analysis capacity of the task-PIOA framework. As a first step for estimating the analysis capacity, we will apply the task-PIOA framework to analyze a protocol on a different primitive and show the way for the formulation of the adaptive adversary model.

Our Contribution. We choose Diffie-Hellman(DH) key exchange [7] (in passive and adaptive adversary model) by following two reasons:

– DH protocol is based on the hardness of a computational problem, i.e., DDH problem. Since the basic symbolic approach cannot deal with this type of protocols, it is easy to see the deference between the task-PIOA framework and other previous approaches.
– It is desirable that the analyzed protocol is simple since we are able to concentrate on the difference with the analysis of an oblivious transfer protocol in the previous study. From the same reason, we will consider the case against only passive adversaries.

Our contributions are (1). reformulating DDH assumption in task-PIOA framework and proving the equivalence between the original DDH assumption and our reformulated one, (2). formulating adaptive adversary model by representing halfway corruption on codes for task-PIOAs in the real and ideal system, and (3). showing that task-PIOA framework has an adequate capacity by proving the security of DH protocol in the framework.

Related Works. As well as task-PIOA framework, there are some approaches that directly capture "computational security" with the formal model.

Mitchell et al. introduced a *Probabilistic Polynomial-time Calculus* [8–12] to prove authentication properties of security protocols.

Backes, Pfitzmann, and Waidner designed *Reactive Simulatability* framework [13–15] that is an abstract cryptographic library including various cryptographic primitives. They shown soundness with respect to various computational cryptographic primitives under active attacks.

Blanchet developed a computationally sound mechanized proofs [16] by *sequences of games* that are usual tools for cryptographic communities. He gave analyses of public-key encryption and signature schemes, and successfully tested on examples with the tool "CryptoVerif".

Corin and den Hartog used a *probabilistic Hoare-style logic* [17] for formalizing game-based cryptographic proofs.

Küsters applied *Inexhaustible Interactive Turing Machine* [18] for proofs of simulation-based paradigm.

2 Task-PIOA framework

In this section, we overview the PIOA and task-PIOA framework. We can't show all needed definitions or notations here. So, please refer to [6] for them.

First, we describe the definition of PIOA.

A PIOA is defined as a tuple $\mathcal{P} = (Q, \overline{q}, I, O, H, D)$, where each parameter represents a set of states, a start state, a set of input actions, a set of output actions, a set of internal (hidden) actions and a transition relation which is included $(Q \times (I \cup O \cup H) \times \mathsf{Disc}(Q))$ where $\mathsf{Disc}(Q)$ is the set of discrete probability measures on Q. Especially, $I \cup O \cup H$ is called actions of $\mathcal{P}$ and $I \cup O$ is called the external actions of $\mathcal{P}$.

The execution of some protocol in PIOA is expressed by sequences like $\alpha = q_0 a_1 q_1 a_2 \cdots$, where each $q_i \in Q$ and $a_i \in A$. Here, trace of α denoted by $trace(\alpha)$ is defined as $\{a_i \,|\, a_i \in \alpha \, and \, a_i \in E\}$, i.e. the set of external actions in α.

PIOA can deal with probabilistic protocol execution, which is described using *probability measures on execution fragments* of PIOA $\mathcal{P}$.

Applying this to *trace*, trace distribution of a probabilistic protocol execution $tdist(\epsilon)$ is defined and a set of *tdist* of probabilistic protocol executions is also defined and denoted by $tdists(\mathcal{P})$, which is used to define indistinguishability in task-PIOA framework.

Other property for PIOA is, for example, its composability. If two PIOAs $\mathcal{P}_1$ and $\mathcal{P}_2$ satisfy certain restriction (if they are "compatible"), then the composed PIOA is denoted by $\mathcal{P}_1||\mathcal{P}_2$. And when PIOAs are composed, it is usually necessary to turn some output actions of one composed PIOA into internal tasks. For this purpose, there is a hiding operation in PIOA framework. It turns some output actions of certain PIOA into internal actions.

Now, based on the PIOA framework, task-PIOA framework is defined as follows.

Definition 1. *A task-PIOA is a pair* $\mathcal{T} = (\mathcal{P}, R)$*, where*
- $\mathcal{P} = (Q, \overline{q}, I, O, H, G)$ is a PIOA.
- R is an equivalence relation on the locally-controlled actions $(O \cup H)$. The equivalence classes of R are called tasks.

In task-PIOA framework, there is a notion of a task scheduler, which chooses the next task to perform. The scheduler is simply a sequence of tasks.

As a task-PIOA is defined as a pair of PIOA and R, composition and hiding can be defined as in PIOA framework. That is, composition of task-PIOAs $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ is defined to be $\mathcal{P}_1||\mathcal{P}_2$ as in PIOA framework, and $R_1 \cup R_2$ for the relation part. Hiding is also the same as for PIOA part, and relation part receives no effect by hiding.

Now, we describe how indistinguishability of external behavior for a task-PIOA is formulated. It is formulated by the relation $\leq_0$, which is defined as follows.

Definition 2. *Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are task-PIOAs having the same I/O. Then, $\mathcal{T}_1 \leq_0 \mathcal{T}_2$ if, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, $tdists(\mathcal{T}_1||\mathcal{E}) \subseteq tdists(\mathcal{T}_2||\mathcal{E})$. $\leq_0$ is called implementation relation.*

This definition means that implementation relation holds if the trace distribution set made in $\mathcal{T}_1||\mathcal{E}$ is also made in $\mathcal{T}_2||\mathcal{E}$. So, $\mathcal{E}$ can't distinguish the two protocols. Therefore, to prove the security of a protocol, it is needed to construct a real protocol using task-PIOAs, and then construct an ideal protocol that is indistinguishable from the real protocol for any environment $\mathcal{E}$.

For proving the implementation relation, there is another relation called *simulation relation* which shows the sufficient conditions for proving the implementation relation. Simulation relation is the equivalence relation on probabilistic executions, which makes it possible to verify whether states in PIOAs are equivalent or not task by task. Proof in this way is favorable to automate a proof. In the security proof, we establish a relation and prove that it is a simulation relation, and apply the next lemma, where comparable means to have the same I/O and closed action-deterministic is some restriction and assumption on task-PIOAs.

Lemma 1. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ are two comparable closed action-deterministic task-PIOAs. If there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $tdists(\mathcal{T}_1||\mathcal{E}) \subseteq tdists(\mathcal{T}_2||\mathcal{E})$.*

Task-PIOA framework has the ability to deal with computational issues. This is made possible by considering that each task-PIOA is polynomial-time-bounded. For time-bounded PIOAs, the implementation relation is defined as the following definition.

Definition 3. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable task-PIOAs, $\epsilon, b \in \mathbb{R}^{\geq 0}$, and $b_1, b_2 \in \mathbb{N}$. Then $\mathcal{T}_1 \leq_{\epsilon,b,b_1,b_2} \mathcal{T}_2$ provided that, for every b-time-bounded environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, and for every b_1-time-bounded task scheduler ρ_1 for $\mathcal{T}_1||\mathcal{E}$, there is a b_2-time-bounded task scheduler ρ_2 for $\mathcal{T}_2||\mathcal{E}$ such that*

$$|Paccept(\mathcal{T}_1||\mathcal{E}, \rho_1) - Paccept(\mathcal{T}_2||\mathcal{E}, \rho_2)| \leq \epsilon$$

where Paccept is the probability that the environment outputs accept.

Usually, $\leq_{\epsilon,b,b_1,b_2}$ is denoted by $\leq_{neg,pt}$ for short. A useful property of this relation is that it is transitive; if $\mathcal{T}_1 \leq_{neg,pt} \mathcal{T}_2$ and $\mathcal{T}_2 \leq_{neg,pt} \mathcal{T}_3$ then $\mathcal{T}_1 \leq_{neg,pt} \mathcal{T}_3$. So, a security proof can be done like this; first divide the proof into some parts, then prove each part, and finally compose them. In this way, the task-PIOA framework enables to carry out a proof on protocols at a high level of abstraction. Other properties such as composition, hiding and simulation relation hold for time-bounded PIOA, too.

3 Reformulation of DDH assumption

To prove the security of Diffie-Hellman key exchange against passive adversary, we need to use DDH assumption. So, in this section, we reformulate the original DDH assumption definition in task-PIOA framework. First, we show the original DDH assumption.

Definition 4. *Let $G = \{G_k\}_{k \in \mathbf{N}}$ be a group family having a generator set $G_G = \{(G_G)_k\}_{k \in \mathbf{N}}$ and a exponent set $G_E = \{(G_E)_k\}_{k \in \mathbf{N}}$. For every PPT algorithm family $\mathcal{A} = \{\mathcal{A}_k\}_{k \in \mathbf{N}}$, there is a negligible function ϵ such that, for all k,*

$$\left| Pr\begin{bmatrix} g \leftarrow (G_G)_k; \\ a \leftarrow (G_E)_k; \\ b \leftarrow (G_E)_k; \\ c \leftarrow a \cdot b; \\ \mathcal{A}_k(g, g^a, g^b, g^c) = 1 \end{bmatrix} - Pr\begin{bmatrix} g \leftarrow (G_G)_k; \\ a \leftarrow (G_E)_k; \\ b \leftarrow (G_E)_k; \\ R \leftarrow G_k; \\ \mathcal{A}_k(g, g^a, g^b, R) = 1 \end{bmatrix} \right| \leq \epsilon(k).$$

Then, we describe this reformulated definition using task-PIOAs. Prior to this, we define a task-PIOA $Src(D)$ which is used to define DDH assumption.

Definition 5. *A task-PIOA $Src(D)$ chooses a random value uniformly from the designated domain D and outputs the value.*

$Src(D)$ realizes this functionality with two tasks: $\{choose\text{-}rand\}$ and $\{rand(*)\}$, where $\{choose\text{-}rand\}$ chooses a value (this is an internal task) and $\{rand(*)\}$ outputs the value (this is an output task).

Using this task-PIOA, we define two task-PIOA families $\overline{SDDH}$ and $\overline{SDDHR}$ defined as follows.

Definition 6 ($\overline{SDDH}$). *The task-PIOA family $\overline{SDDH}$ is defiled as ($\overline{Src_{gval}}$ || $\overline{Src_{aval}}$ || $\overline{Src_{bval}}$ || $\overline{Cal}$), where*

- $\overline{Src_{gval}} = \{(Src_{gval})_k\}_{k \in \mathbf{N}}$, *where each* $(Src_{gval})_k$ *is isomorphic to* $Src(G_G)$,
- $\overline{Src_{aval}} = \{(Src_{aval})_k\}_{k \in \mathbf{N}}$, $\overline{Src_{bval}} = \{(Src_{bval})_k\}_{k \in \mathbf{N}}$, *where each* $(Src_{aval})_k$ *and* $(Src_{bval})_k$ *are isomorphic to* $Src(G_E)$,
- $\overline{Cal} = \{Cal_k\}_{k \in \mathbf{N}}$, *where each* Cal_k *receives the elements* $a, b \in G_E$ *from* $(Src_{aval})_k$ *and* $(Src_{bval})_k$, *and outputs the values* $A = g^a$, $B = g^b$ *and* $C = g^{ab}$. *Each* Cal_k *is defined as* $Cal(G_k, G_{E_k}, G_{G_k})$, *where* $Cal(G, G_E, G_G)$ *is defined in figure 1.*

Definition 7 ($\overline{SDDHR}$). *The task-PIOA family $\overline{SDDHR}$ is defined as hide($\overline{Src_{gval}}||\overline{Src_{aval}}||\overline{Src_{bval}}||\overline{Src_{Cval}}||\overline{Cal'}, rand(C)_{Cval}$), where*

- $\overline{Src_{gval}} = \{(Src_{gval})_k\}_{k \in \mathbf{N}}$, *where each* $(Src_{gval})_k$ *is isomorphic to* $Src(G_G)$,
- $\overline{Src_{aval}} = \{(Src_{aval})_k\}_{k \in \mathbf{N}}$, $\overline{Src_{bval}} = \{(Src_{bval})_k\}_{k \in \mathbf{N}}$, *where each* $(Src_{aval})_k$ *and* $(Src_{bval})_k$ *are isomorphic to* $Src(G_E)$,
- $\overline{Src_{Cval}} = \{(Src_{Cval})_k\}_{k \in \mathbf{N}}$, *where each* $(Src_{Cval})_k$ *is isomorphic to* $Src(G)$,
- $\overline{Cal'} = \{Cal'_k\}_{k \in \mathbf{N}}$, *where each* Cal'_k *receives the elements* $a, b \in G_E$ *from* $(Src_{aval})_k$, $(Src_{bval})_k$ *and the elements* $C \in G$ *from* $(Src_{Cval})_k$, *then outputs the values* $A = g^a$, $B = g^b$ *and* C. *Each* Cal'_k *is defined as* $Cal'(G_k, G_{E_k}, G_{G_k})$, *where* $Cal'(G, G_E, G_G)$ *is defined by replacing the internal action* $fix - Cval$ *in the code of* $Cal(G, G_E, G_G)$ *to the input action* $rand(C)_{Cval}$.

$Cal(G, G_E, G_G)$:
Signature:

Input:
$rand(g)_{gval}, g \in G_G$
$rand(a)_{aval}, a \in G_E$
$rand(b)_{bval}, b \in G_E$
Output:
$out1(A),\ A \in G$
$out2(B),\ B \in G$
$out3(C),\ C \in G$

Internal:
$fix - Aval$
$fix - Bval$
$fix - Cval$

State:
$gval \in G_G \cup \{\perp\}$, initially $\perp$
$aval, bval \in G_E \cup \{\perp\}$, initially $\perp$
$Aval, Bval, Cval \in G \cup \{\perp\}$, initially $\perp$

Transitions:

$rand(g)_{gval}$
Effect:
if $gval = \perp$ then $gval := g$

$rand(a)_{aval}$
Effect:
if $aval = \perp$ then $aval := a$

$rand(b)_{bval}$
Effect:
if $bval = \perp$ then $bval := b$

$fix - Aval$
Precondition:
$gval,\ aval \neq \perp$
Effect:
$Aval := gval^{aval}$

$fix - Bval$
Precondition:
$gval,\ bval \neq \perp$
Effect:
$Bval := gval^{bval}$

$fix - Cval$
Precondition:
$gval, aval, bval \neq \perp$
Effect:
$Cval := gval^{aval \cdot bval}$

$out(A)_{Aval}$
Precondition:
$A = Aval \neq \perp$
Effect:
none

$out(B)_{Bval}$
Precondition:
$B = Bval \neq \perp$
Effect:
none

$out(C)_{Cval}$
Precondition:
$C = Cval \neq \perp$
Effect:
none

Tasks: $\{fix - Aval\}, \{fix - Bval\}, \{fix - Cval\}, \{out1(*)\}, \{out2(*)\}, \{out3(*)\}$

Fig. 1. DDH automaton, $Cal(G, G_E, G_G)$

Now, we define DDH assumption in task-PIOA framework.

Definition 8. *DDH assumption in task-PIOA framework is to assume the following: for a group family $G = \{G_k\}_{k \in \mathbf{N}}$, $\overline{SDDH} \leq_{neg,pt} \overline{SDDHR}$ holds, where $\overline{SDDH}$ and $\overline{SDDHR}$ are defined in Definition 6 and Definition 7 respectively.*

Using this definition, the following two theorems hold.

Theorem 1. *If DDH assumption in Definition 4 holds for a group family $G = \{G_k\}_{k \in \mathbf{N}}$, then DDH assumption in Definition 8 also holds for the same G.*

Proof. (Sketch) This proof suffices to show the existence of a negligible function ϵ for every $k \in \mathbf{N}$ and for every polynomial-time-bounded environment $\mathcal{E}$ for both $SDDH_k$ and $SDDHR_k$, and for every polynomial-time-bounded task scheduler ρ_1 for $SDDH_k \mid\mid Env$, there exist polynomial-time-bounded task scheduler ρ_2 for $SDDHR_k \mid\mid Env$ such that

$$\Big| Paccept(SDDH_k || \mathcal{E}, \rho_1) - Paccept(SDDHR_k || \mathcal{E}, \rho_2) \Big| \leq \epsilon(k)$$

where $Paccept$ is the probability that an environment outputs *accept*.

To show this, define a probabilistic polynomial-time algorithm $\mathcal{A}$ and then apply Definition 4. This proof is similar to the proof of Theorem 6.5 of [6].

Theorem 2. *If DDH assumption in Definition 8 holds for a group family $G = \{G_k\}_{k \in \mathbf{N}}$, then DDH assumption in Definition 4 also holds for the same G.*

Proof. (Sketch) This proof needs to show that for any probabilistic polynomial-time algorithm $\mathcal{A}$, the inequality in Definition 4 holds. This can be shown by defining an appropriate polynomial-time environment family $\overline{\mathcal{E}}$ and a polynomial-time-bounded task scheduler family $\overline{\rho_1}$ for $\overline{SDDH} \mid\mid \overline{\mathcal{E}}$, then there exists a polynomial-time-bounded task scheduler family $\overline{\rho_2}$ for $\overline{SDDHR} \mid\mid \overline{\mathcal{E}}$ such that the inequality appeared in the previous proof holds. Then, it can be shown that the two equalities state the same thing, which implies that the goal is shown.

4 Diffie-Hellman Key Exchange

In this section, we show the construction of a real system and an ideal system of Diffie-Hellman key exchange in the task-PIOA framework.

4.1 Ideal System *IS*

An ideal system IS is defined as the specification for the correctness and secrecy properties which have to be guaranteed by Key Exchange.

We parameterize IS by a finite domain G with sets of generators G_G and exponents G_E.

IS consists of two interacting task-PIOAs: the key exchange functionality $Funct$ and a simulator Sim. We describe the details of these task-PIOAs.

Key Exchange Functionality *Funct* We define key exchange functionality in task-PIOA framework as a task-PIOA *Funct*. It receives *in* message from the environment, then it chooses a random value $Kval \in G$ and outputs it. This definition is based on the functionality $\mathcal{F}_{\mathrm{KE}}$ [19] in UC framework. The code for $Funct(G)$ is defined in Figure 2. Note that $\psi \subseteq \{Init, Resp\}$ is the corrupted endpoints.

$Funct(G)$:

Signature:

Input:
$rand(K), K \in G$
$corrupt(Cparty), Cparty \in \{Init, Resp\}$

Output:
$out(K)_{Init},\ K \in G$
$out(K)_{Resp},\ K \in G$
$out'(K)_{Init},\ K \in G$
$out'(K)_{Resp},\ K \in G$

State:
$Kval \in G \cup \{\bot\}$, initially $\bot$
$\psi \subseteq \{Init, Resp\}$, initially $\emptyset$

Transitions:

```
  $rand(K)$
Effect:
if $Kval = \bot$ then $Kval := K$
$corrupt(Cparty)$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$
$out(K)_{Init}$
Precondition:
$K = Kval \neq \bot$, $Init \notin \psi$
Effect:
none

  $out(K)_{Resp}$
Precondition:
$K = Kval \neq \bot$, $Init \notin \psi$
Effect:
none
$out'(K)_{Init}$
Precondition:
$K = Kval \neq \bot$, $Init \in \psi$
Effect:
none
$out'(K)_{Resp}$
Precondition:
$K = Kval \neq \bot$, $Init \in \psi$
Effect:
none
```

Tasks: $\{out(*)_{Init}\}$, $\{out(*)_{Resp}\}$, $\{out'(*)_{Init}\}$, $\{out'(*)_{Resp}\}$.

Fig. 2. Code for $Funct(G)$

Simulator *Sim* The simulator *Sim* is an arbitrary task-PIOA, but there are some constraints to fix the interface. Constraints on *Sim* is given in Figure 3.

Complete ideal System *IS* Complete ideal system *IS* is the composition of $Funct(G)$ and $Sim(G, G_E, G_G)$. After occurring $Init \in \psi$, $\{out'(*)_{Init}\}$ tasks are hidden. Also, after occurring $Resp \in \psi$, $\{out'(*)_{Resp}\}$ tasks are hidden.

A ideal system family $\overline{IS}$ for group family $\overline{G}$ is a family $\{IS_k\}_{k \in \mathbf{N}}$, where for each k, IS_k is a ideal system. So, IS_k is the composition of *Funct*, *Sim* and $(Src_{Kval})_k$ with hiding $rand(K)_{Kval}$ actions.

Signature:

Input:	Output:
$in(g)_{Init}$, $g \in G_G$	$out(K)_{Init}, K \in G$
$in(g)_{Resp}$, $g \in G_G$	$out(K)_{Resp}, K \in G$
$out(K)_{Init}$, $K \in G$	Arbitrary other output actions
$out(K)_{Resp}$, $K \in G$	Internal:
$out'(K)_{Init}$, $K \in G$	Arbitrary other internal actions
$out'(K)_{Resp}$, $K \in G$	
Arbitrary other input actions	

Tasks: $\{out(*)_{Init}\}$, $\{out(*)_{Resp}\}$.

Fig. 3. Constraints on $Sim(G, G_E, G_G)$

4.2 Real System *RS*

A real system RS is also parameterized by a finite domain G with sets of generators G_G and exponents G_E. Based on these parameters, we define a derived set of the message alphabet M equal to $\{(1, A) : A \in G\} \cup \{(2, B) : B \in G\}$.

RS consists of three interacting task-PIOAs: The Initiator $Init$, the Responder $Resp$ and the Adversary Adv. We show the construction of these task-PIOAs.

The Initiator *Init* $Init$ receives a random value $gval$ and $aval$ from Src_{aval}, calculates $Aval = gval^{aval}$, and sends it. When $Bval$ is sent from $Resp$, $Init$ calculates $Kval = Bval^{aval}$ and outputs $Kval$ for the environment. We show the code for $Init$ in Figure 4.

The Responder *Resp* $Resp$ acts in a similar way to $Init$. So, the code for $Resp$ can be obtained by simply replacing each "a" or "A" with "b" or "B". For space limitations, we omit the code for $Resp$.

Adversary *Adv* The adversary can read messages which $Init$ and $Resp$ send to each other. This can be captured that Adv receives *send* inputs from and provides *receive* outputs to $Init$ and $Resp$. In addition, if $\psi \neq \emptyset$(this means at least one endpoint is corrupted), Adv receives inputs from the environment or Src automatons of the corrupted endpoints. It also acts as an intermediary for outputs of endpoints in ψ. Adv may communicate with the environment using arbitrary inputs and outputs (called "new" inputs and outputs) which are disjoint from all the other actions that appear in any of explicitly-defined components.
We abbreviate the code for Adv as the fact described here it the essence for Adv. The code for Adv is defined in Figure 5.

Complete real system A complete real system RS is the composition of the following five task-PIOAs: $Init$, $Resp$, Adv, $Src(G_G)_{gval}$,

$Init(G, G_E, G_G)$:

Signature:

Input:
$rand(g)_{Init}$, $g \in G_G$
$rand(a)_{aval}$, $a \in G_E$
$receive(2, B)_{Init}, B \in G$
$corrupt(Cparty)_{Init}, Cparty \in \{Init, Resp\}$

Output:
$send(1, A)_{Init}$, $A \in G$
$out(K)_{Init}$, $K \in G$
$out'(K)_{Init}$, $K \in G$

Internal:
$fix - Aval_{Init}$
$fix - Kval_{Init}$

State:
$received \in \{\perp, \top\}$, initially $\perp$
$aval \in G_E \cup \{\perp\}$, initially $\perp$
$Aval, Bval, Kval_{Init} \in G \cup \{\perp\}$, initially $\perp$
$\psi \subseteq \{Init, Resp\}$, initially $\emptyset$

Transitions:

```
$rand(g)_{Init}$
Effect:
if $gval = \perp$ then $gval := g$

$rand(a)_{Init}$
Effect:
if $aval = \perp$ then $aval := a$

$receive(2, B)_{Init}$
Effect:
if $Bval = \perp$ then $Bval := B$

$corrupt(Cparty)_{Init}$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup c$

$fix - Aval_{Init}$
Precondition:
$gval, aval \neq \perp$, $Aval = \perp$
Effect:
$Aval := gval^{aval}$

$send(1, A)_{Init}$
Precondition:
$A = Aval \neq \perp$
Effect:
none

$fix - Kval_{Init}$
Precondition:
$aval, Bval \neq \perp$,
$Kval_{Init} = \perp$
Effect:
$Kval_{Init} := Bval^{aval}$

$out(K)_{Init}$
Precondition:
$K = Kval \neq \perp$, $Init \notin \psi$
Effect:
none

$out'(K)_{Init}$
Precondition:
$K = Kval \neq \perp$, $Init \in \psi$
Effect:
none
```

Tasks: $\{send(1, *)_{Init}\}, \{fix - Aval_{Init}\}, \{fix - Kval_{Init}\}$, $\{out'(k)_{Init}\}$, $\{out(k)_{Init}\}$.

Fig. 4. Code for $Init(G, G_E, G_G)$

$Src(G_E)_{aval}$ and $Src(G_E)_{bval}$, with all the *rand*, *send* and *receive* actions are hidden. If $Init \in \psi$, hide it's out' outputs, and If $Resp \in \psi$, hide it's out' outputs too.

A real system family $\overline{RS}$ for group family $\overline{G}$ is a family $\{RS_k\}_{k \in \mathbf{N}}$, where for each k, RS_k is a real system. So, RS_k is the composition of $Init(G, G_E, G_G)$, $Resp(G, G_E, G_G)$, $Adv(G, G_E, G_G)$, $Src((G_G)_k)_{gval}$, $Src((G_E)_k)_{aval}$ and $Src((G_E)_k)_{bval}$, with all the *rand*, *send* and *receive* actions are hidden.

```
$Adv(G, G_E, G_G)$ :
Signature:
Input:
  $send(1, A)_{Init}, A \in G$
  $send(2, B)_{Resp}, B \in G$
  $rand(g)_{Init},\ g \in G_G$
  $rand(g)_{Resp},\ g \in G_G$
  $rand(a)_{aval}, a \in G_E$
  $out'(K)_{Init},\ K \in G$
  $rand(b)_{bval}, b \in G_E$
  $out'(K)_{Resp},\ K \in G$
  Arbitrary other input actions
  (called "new" input actions)
Output:
  $receive(1, A)_{Resp}, A \in G$
  $receive(2, B)_{Init}, B \in G$
  $out(K)_{Init},\ K \in G$
  $out(K)_{Resp},\ K \in G$
  $corrupt(Cparty)_{Init}, Cparty \in \{Init, Resp\}$
  $corrupt(Cparty)_{Resp}, Cparty \in \{Init, Resp\}$
  Arbitrary other output actions
  (called "new" output actions)
Internal:
  Arbitrary internal actions
  (called "new" internal actions)

State:
$messages$, a set of pairs in $M \times \{Init, Resp\}$, initially $\emptyset$
$Kval(Init) \in G \cup \{\bot\}$, initially $\bot$
$Kval(Resp) \in G \cup \{\bot\}$, initially $\bot$
$\psi \subseteq \{Init, Resp\}$, initially $\emptyset$

Transitions:

$send(m)_{Init}$
Effect:
$messages := messages \cup \{(m, Resp)\}$

$send(m)_{Resp}$
Effect:
$messages := messages \cup \{(m, Init)\}$

$receive(m)_{Init}$
Precondition:
$(m, Init) \in messages$
Effect:
none

$receive(m)_{Resp}$
Precondition:
$(m, Resp) \in messages$
Effect:
none

$corrupt(Cparty)_{Init}$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$

$corrupt(Cparty)_{Resp}$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$

$out'(K)_{Init}$
Precondition:
$Init \in \psi$
Effect:
If $Kval(Init) = \bot$, $Kval(Init) = K$

$out'(K)_{Resp}$
Precondition:
$Resp \in \psi$
Effect:
If $Kval(Init) = \bot$, $Kval(Resp) = K$

$out(K)_{Init}$
Precondition:
$K = Kval(Init) \neq \bot$, $Init \in \psi$
Effect:
none

$out(K)_{Resp}$
Precondition:
$K = Kval(Resp) \neq \bot$, $Resp \in \psi$
Effect:
none

$in(m)_{Init}, rand(a)_{aval}$
Precondition:
$Init \in \psi$
Effect:
Arbitrary changes to new state variables

$rand(b)_{bval}$
Precondition:
$Resp \in \psi$
Effect:
Arbitrary changes to new state variables

New input action
Effect:
Arbitrary changes to new state variables

New output or internal action
Precondition:
Arbitrary
Effect:
Arbitrary changes to new state variables

Tasks: $\{receive(1, *)_{Resp}\}$, $\{receive(2, *)_{Init}\}$, $\{corrupt(*)_{Init}\}$, $\{corrupt(*)_{Resp}\}$, $\{out(*)_{Init}\}$,
$\{out(*)_{Resp}\}$.
Arbitrary tasks for new actions.
```

Fig. 5. Code for $Adv(G, G_E, G_G)$

5 Main Theorem

Here, we show the main theorem we need to prove.

5.1 Families of sets

We use the family of sets $\overline{G}$ which is a family $\{G_k\}_{k\in\mathbf{N}}$ with $\{G_{E_k}\}_{k\in\mathbf{N}}$ and $\{G_{G_k}\}_{k\in\mathbf{N}}$. Also, we define the derived family of sets of the message alphabet $\overline{M}$, where $M_k = \{(1, A) : A \in G_k\} \cup \{(2, B) : B \in G_k\}$.

Theorem 3. *Let $\overline{RS}$ be a real-system family for $\overline{G}$, in which the adversary family $\overline{Adv}$ is polynomial-time-bounded. Then, there exists an ideal-system family $\overline{IS}$ for $\overline{G}$, in which the simulator family $\overline{Sim}$ is polynomial-time-bounded, and such that $\overline{RS} \leq_{neg,pt} \overline{IS}$.*

6 Correctness Proof

In this section, we prove $\overline{RS} \leq_{neg,pt} \overline{IS}$. For this purpose, we introduce an intermediate system family $\overline{Int}$ and divide the proof into two parts: showing $\overline{RS} \leq_{neg,pt} \overline{Int}$ and $\overline{Int} \leq_{neg,pt} \overline{SIS}$, where $\overline{SIS}$ is a family of a structured ideal system. This makes the proof clear. We give a sketch proof for each of them.

6.1 Simulator structure

For each k, we define a simulator $SSim_k$ as the composition of the following four task-PIOAs, with all *send*, *receive*, *rand* and *out'* tasks hidden.

- IR, an abstract combination of $Init$ and $Resp$.
- $Src((G_G)_k)_{gval}, Src((G_E)_k)_{aval}$, $Src((G_E)_k)_{bval}$, isomorphic to $Src(D_k)$.
- Adv'_k, identical to the adversary Adv_k in RS_k except that its $out'(K)_{Init}$ input actions are renamed to $Iout(K)$ and $out'(K)_{Resp}$ input actions are renamed to $Rout(K)$.

The code for IR is defined in Figure 6 .

We define the structured ideal system SIS to be the composition $Funct_k||SSim_k||Src((G_E)_k)_{Kval}$, where all the $rand, send, receive$ actions hidden. After occurring $Init \in \psi$, then hide $Iout'$ outputs in IR, and also after occurring $Resp \in \psi$, then hide $Rout'$ outputs in IR.

6.2 Proof for $\overline{RS} \leq_{neg,pt} \overline{Int}$

$\overline{Int}$ To begin with, we define the intermediate task-PIOA $\overline{Int}$.

Definition 9. *A task-PIOA family $\overline{Int} = \{Int_k\}_{k\in\mathbf{N}}$ is defined to be the same as $\overline{IS}$ except that IR is replaced by IR'.*

IR' is basically just combining $Init$ and $Resp$ in RS together, so most of the tasks and states are the same as them. Some exceptions are that IR' have input action $out(K)$ from $Funct$ and the effects of *receive* actions are changed.

$IR(G, G_E, G_G)$:

Signature:

Input:
$out'(K)$, $K \in G$
$rand(g)_{gval}$, $g \in G_G$
$rand(a)_{aval}$, $a \in G_E$
$rand(b)_{bval}$, $b \in G_E$
$receive(1, A)_{Resp}$, $A \in G$
$receive(2, B)_{Init}$, $B \in G$
$corrupt(Cparty)$, $Cparty \in \{Init, Resp\}$

Output:
$send(1, A)_{Init}$, $A \in G$
$send(2, B)_{Resp}$, $B \in G$
$Iout, Rout(K)$, $K \in G$

Internal:
$fix - Aval_{Init}$
$fix - Bval_{Resp}$

State:
$gval \in G_G \cup \bot$, initially $\bot$
$aval, bval \in G_E \cup \bot$, initially $\bot$
$Aval, Bval, Kval \in G \cup \{\bot\}$, initially $\bot$
$received_{Init}, received_{Resp} \in \{\bot, \top\}$, initially $\bot$
$\psi \subseteq \{Init, Resp\}$, initially $\emptyset$

Transitions:

$out'(K)$
Effect:
if $Kval = \bot$ then $Kval := K$

$rand(g)_{gval}$
Effect:
if $gval = \bot$ then $gval := g$

$rand(a)_{aval}$
Effect:
if $aval = \bot$ then $aval := a$

$rand(b)_{bval}$
Effect:
if $bval = \bot$ then $bval := b$

$receive(1, A)_{Resp}$
Effect:
if $received_{Resp} = \bot$ then
$received_{Resp} := \top$

$receive(2, B)_{Init}$
Effect:
if $received_{Init} = \bot$ then
$received_{Init} := \top$

$fix - Aval_{Init}$
Precondition:
$gval, aval \neq \bot$, $Aval = \bot$
Effect:
$Aval := gval^{aval}$

$corrupt(Cparty)$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$

$fix - Bval_{Resp}$
Precondition:
$gval, bval \neq \bot$, $Bval = \bot$
Effect:
$Bval := gval^{bval}$

$send(1, A)_{Init}$
Precondition:
$A = Aval \neq \bot$, $Kval' \neq \bot$
Effect:
none

$send(2, B)_{Resp}$
Precondition:
$B = Bval \neq \bot$
Effect:
none

$Iout(K), Rout(K)$
Precondition:
$K = Kval \neq \bot$,
$received_{Init} \neq \bot$
Effect:
none

Tasks: $\{send(1, *)_{Init}\}$, $\{send(2, *)_{Resp}\}$, $\{fix - Aval_{Init}\}$, $\{fix - Bval_{Resp}\}$, $\{Iout(*)\}$, $\{Rout(*)\}$.

Fig. 6. Code for $IR(G, G_E, G_G)$

Proof sketch What we need to show is for every k, $RS_k \leq_0 Int_k$. We fix some k and remove its description below. To prove the relation, it is sufficient to define a equivalence relation R between the two systems (i.e. establishing a simulation relation). We describe how to define the equivalence relation below.

In this case, R can be defined for most parts by simply relating the tasks or states having the same name. This is because we defined Int almost the same as RS, and this makes the proof very simple. But there are an exception like *receive* inputs. *receive* tasks have different effects between two systems. However, related tasks or states for them are easily found by comparing the code for $Init$, $Resp$ and IR'.

The established relation R can be proven to be a simulation relation. Thus, for every k, $RS_k \leq_0 Int_k$ holds and so, we have $\overline{RS} \leq_{neg,pt} \overline{Int}$ as we needed.

6.3 $\overline{Int} \leq_{neg,pt} \overline{IS}$

We use DDH assumption in Definition 8 to prove the relation. To do this, we first define the subsystems of $\overline{Int}$ and $\overline{IS}$.

$\overline{SubInt}$ and $\overline{SubIS}$ Subsystems for $\overline{Int}$ and $\overline{IS}$ are denoted by $\overline{SubInt}$ and $\overline{SubIS}$ respectively, and defined as follows.

Definition 10. *Task-PIOA families $\overline{SubInt}$ and $\overline{SubIS}$ are defined as follows:*
Let $U = \{\{rand(*)_{gval}\}, \{rand(*)_{aval}\}, \{rand(*)_{bval}\}\}$ *and* $U' = \{\{rand(*)_{gval}\}, \{rand(*)_{aval}\}, \{rand(*)_{bval}\}, \{rand(*)_{Kval}\}\}$ *then*

- $\overline{SubInt} = hide(\overline{IR'}||\overline{Src_{gval}}||\overline{Src_{aval}}||\overline{Src_{bval}}, U)$
- $\overline{SubIS} = hide(\overline{IR}||\overline{Src_{gval}}||\overline{Src_{aval}}||\overline{Src_{bval}} \mid\mid \overline{Src_{Kval}}, U')$

The only difference between these two subsystems is the way $Kval$ state variable is computed: $Kval = gval^{aval \cdot bval}$ in $\overline{SubInt}$ and it is replaced by a random value R in $\overline{SubIS}$. This is exactly the difference between $\overline{SDDH}$ and $\overline{SDDHR}$ which are defined previously, and proven that $\overline{SDDH} \leq_{neg,pt} \overline{SDDHR}$ holds.

So, in order to use this relation, we define a new polynomial-time-bounded task-PIOA family $\overline{Ifc}$ (Ifc means interface) and we compose it to $\overline{SDDH}$ and $\overline{SDDHR}$ in order to mimic $\overline{SubInt}$ and $\overline{SubIS}$ respectively. The code for Ifc is shown in Figure 7.

6.4 $\overline{DH1} \leq_{neg,pt} \overline{DH2}$

Using $\overline{Ifc}$, we define task-PIOA families $\overline{SDDHDH}$ and $\overline{SDDHRDH}$.

Definition 11. *A task-PIOA family $\overline{DH1}$ is defined as*

$$hide(\overline{SDDH} \mid\mid \overline{Ifc} \mid\mid, \{\{out1(*)\}\,\{out2(*)\}, \{out3(*)\}\})$$

where $\overline{SDDH}$ is the same as in DDH assumption, and $\overline{Ifc}$ is defined in Figure 7.

Definition 12. *A task-PIOA family $\overline{DH2}$ is defined as*

$$hide(\overline{SDDHR} \mid\mid \overline{Ifc} \mid\mid, \{\{out1(*)\}\,\{out2(*)\}, \{out3(*)\}\})$$

where $\overline{SDDHR}$ is the same as in DDH assumption, and $\overline{Ifc}$ is as in Definition 11.

We call $SDDHDH$ $DH1$ and $SDDHRDH$ $DH2$ for short, respectively. For these two task-PIOA families, the following Lemma holds:

Lemma 2. $\overline{DH1} \leq_{neg,pt} \overline{DH2}$

Proof. This is straightforward using the definitions of $DH1$ and $DH2$, and $\overline{SDDH} \leq_{neg,pt} \overline{SDDHR}$.

6.5 $\overline{\boldsymbol{SubInt}} \leq_0 \overline{\boldsymbol{DH1}}$

To prove this, we again establish an equivalence relation R between the two systems, and show it is a simulation relation. In this case too, many of the states or tasks of $SubInt$ correspond to the same states or tasks of $DH1$.

The exceptions in this case are each fix tasks. This is because in $DH1$, all the calculations of $Aval$, $Bval$ and $Cval$ are done by $SDDH$, the value is transmitted to Ifc by $out(*)$ tasks. So, fix tasks in $SubInt$ corresponds to the pair of fix and out tasks in $DH1$. The same consideration can be done on the state equivalence. These consideration concludes that the relation R is a simulation relation we needed. So, $\overline{SubInt} \leq_{neg,pt} \overline{DH1}$ holds.

6.6 $\overline{\boldsymbol{DH2}}$ implements $\overline{\boldsymbol{SubIS}}$

In this case, similar discussion is possible as in the previous subsection. That is, if $DH1$ is replaced by $DH2$ and $SubInt$ by $SubIS$, it can be seen that this case is similar to the previous case. So, in the similar way, the existence of a simulation relation can be proven.

Therefore, theorem 3 is proven. □

Ifc :

Signature:

Input:
$out(K),\ K \in G$
$out(1, A),\ A \in G$
$out(2, B),\ B \in G$
$out(3, C),\ C \in G$
$receive(1, A)_{Resp}, A \in G$
$receive(2, B)_{Init}, B \in G$
$corrupt(Cparty)_{Init}, Cparty \in \{Init, Resp\}$
$corrupt(Cparty)_{Resp}, Cparty \in \{Init, Resp\}$

Output:
$send(1, A)_{Init},\ A \in G$
$send(2, B)_{Resp},\ B \in G$
$Iout(C),\ C \in G$
$Rout(C),\ C \in G$
$Iout'(C),\ C \in G$
$Rout'(C),\ C \in G$

State:
$Kval \in G \cup \{\perp\}$, initially $\perp$
$Aval, Bval, Cval \in G \cup \{\perp\}$, initially $\perp$
$received_{Init}, received_{Resp} \in \{\perp, \top\}$, initially $\perp$
$\psi \subseteq \{Init, Resp\}$, initially $\emptyset$

Transitions:

```
$out(K)$
Effect:
if $Kval = \perp$ then $Kval := K$

$out(1, A)$
Effect:
if $Aval = \perp$ then $Aval := A$

$out(2, B)$
Effect:
if $Bval = \perp$ then $Bval := B$

$out(3, C)$
Effect:
if $Cval = \perp$ then $Cval := C$

$receive(1, A)_{Resp}$
Effect:
if $received_{Resp} = \perp$ then $received_{Resp} := \top$

$receive(2, B)_{Init}$
Effect:
if $received_{Init} = \perp$ then $received_{Init} := \top$

$corrupt(Cparty)_{Init}$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$

$corrupt(Cparty)_{Resp}$
Effect:
if $Cparty \notin \psi$ then $\psi := \psi \cup Cparty$

$send(1, A)_{Init}$
Precondition:
$A = Aval \neq \perp$
Effect:
none

$send(2, B)_{Resp}$
Precondition:
$received_{Resp} \neq \perp$, $B = Bval \neq \perp$
Effect:
none

$Iout(C)$
Precondition:
$Kval \neq \perp$, $received_{Init} \neq \perp$, $C = Cval \neq \perp$, $Init \notin \psi$
Effect:
none

$Rout(C)$
Precondition:
$Kval \neq \perp$, $received_{Init} \neq \perp$, $C = Cval \neq \perp$, $Resp \notin \psi$
Effect:
none

$Iout'(C)$
Precondition:
$Kval \neq \perp$, $received_{Init} \neq \perp$, $C = Cval \neq \perp$, $Init \in \psi$
Effect:
none

$Rout'(C)$
Precondition:
$Kval \neq \perp$, $received_{Init} \neq \perp$, $C = Cval \neq \perp$, $Resp \in \psi$
Effect:
none
```

Tasks: $\{send(1, *)_{Init}\}, \{send(2, *)_{Resp}\}, \{Iout(*)\}, \{Rout(*)\}, \{Iout'(*)\}, \{Rout'(*)\}$

Fig. 7. Code for Ifc

References

1. Dolev, D., Yao, A.C.C.: On the Security of Public Key Protocols. In: FOCS 1981. (1981) 350–357
2. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: FOCS 2001. (2001) 136–145 **http://eprint.iacr.org/2000/067/**.
3. Canetti, R., Herzog, J.: Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In: TCC 2006. (2006) 380–403
4. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-Bounded Task-PIOAs: A Framework for Analyzing Security Protocols. In: DISC 2006. (2006) 238–253
5. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Task-Structured Probabilistic I/O Automata. In: WODES 2006. (2006)
6. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Using Task-Structured Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol. Technical report, MIT CSAIL-TR-2007-011 (2007)
7. Diffie, W., Hellman, M.E.: New Directions in Cryptography. In: IEEE Trans. on Info. Theory, vol.IT-22, No.6. (1976) 644–654
8. Lincoln, P., Mitchell, J.C., Mitchell, M., Scedrov, A.: A Probabilistic Poly-Time Framework for Protocol Analysis. In: ACM Conference on Computer and Communications Security 1998. (1998) 112–121
9. Lincoln, P., Mitchell, J.C., Mitchell, M., Scedrov, A.: Probabilistic Polynomial-Time Equivalence and Security Analysis. In: World Congress on Formal Methods 1999. (1999) 776–793
10. Mateus, P., Mitchell, J.C., Scedrov, A.: Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Process Calculus. In: CONCUR 2003. (2003) 323–345
11. Ramanathan, A., Mitchell, J.C., Scedrov, A., Teague, V.: Probabilistic Bisimulation and Equivalence for Security Analysis of Network Protocols. In: FoSSaCS 2004. (2004) 468–483
12. Mitchell, J.C., Ramanathan, A., Scedrov, A., Teague, V.: A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. In: Theor. Comput. Sci. 353. (2006) 118–164
13. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: ACM Conference on Computer and Communications Security 2000. (2000) 245–254
14. Pfitzmann, B., Waidner, M.: A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In: IEEE Symposium on Security and Privacy 2001. (2001) 184–
15. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: ACM Conference on Computer and Communications Security 2003. (2003) 220–230
16. Blanchet, B., Pointcheval, D.: Automated Security Proofs with Sequences of Games. In: Advances in Cryptology-CRYPTO 2006. (2006) 537–554
17. Corin, R., den Hartog, J.: A Probabilistic Hoare-style logic for Game-based Cryptographic Proofs. In: ICALP 2006. (2006) 252–263
18. Küsters, R.: Simulation-Based Security with Inexhaustible Interactive Turing Machines. In: CSFW 2006. (2006) 309–320
19. Canetti, R., Krawczyk, H.: Universally Composable Notions of Key Exchange and Secure Channels. In: Advances in Cryptology-EUROCRYPT 2002. (2002) 337–351

A Secure Simplification of the PKMv2 Protocol in IEEE 802.16e-2005

Ender Yüksel[1], Hanne Riis Nielson[2], Christoffer Rosenkilde Nielsen[2], and Mehmet Bülent Örencik[1]

[1] Department of Computer Engineering, Istanbul Technical University, 34469 Istanbul, Turkey
[2] Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads bldg 321, DK-2800 Kongens Lyngby, Denmark
`yuksele@itu.edu.tr` `riis@imm.dtu.dk` `crn@imm.dtu.dk`
`bulent.orencik@bte.mam.gov.tr`

Abstract. Static analysis is successfully used for automatically validating security properties of classical cryptographic protocols. In this paper, we shall employ the same technique to a modern security protocol for wireless networks, namely the latest version of the Privacy and Key Management protocol for IEEE 802.16e, PKMv2. This protocol seems to have an exaggerated mixture of security features. Thus, we iteratively investigate which components are necessary for upholding the security properties and which can be omitted safely. This approach is based on the LySa process calculus and employs the corresponding automated analysis tool, the LySaTool.

Keywords: Protocol Validation, Process Calculi, Static Analysis, Authentication, IEEE 802.16e

1 Introduction

Security in wireless networks is of great concern as the wireless medium faces different threats from wired networks. Thus, in order to provide secure communication, these threats must be taken into account in the design of security protocols for wireless networks. However the standard for wireless metropolitan area networks, IEEE 802.16, incorporated a pre-existing standard called Data Over Cable Service Interface Specifications, designed for wired networks. Therefore the standard failed to protect the IEEE 802.16 link [1] and significant changes in its Privacy and Key Management protocol (PKMv1) were required.

The latest standard, IEEE 802.16e-2005 [2], includes a new version (PKMv2) of the protocol that caters for the shortcomings of the first version. Derivation of the authorization key is now derived by the contribution of both parties using well known standards such as RSA and EAP, where it initially was done only by the base station (BS). Additionally, BS is extended with a certificate, allowing for mutual authentication with the mobile station (MS), which was missing in

PKMv1. Finally, nonces are incorporated in order to avoid replay attacks. These corrections, though thought to benefit the security of the protocol, have also intensively complicated it. This motivates an investigation of which extensions that are really necessary and which that can be omitted without compromising the protocol.

Formal analysis of cryptographic protocols is normally concerned with whether a given protocol satisfies a number of security criteria such as correct establishment of a secret shared key or authentication the principals involved. This has been a very active area of research over the last decades, and the tools, that have been constructed based on the theoretical development, have successfully located many hitherto unknown flaws. One of the most well-known example is the Lowe's attack [3, 4] of the Needham Schroeder public key protocol [5] using the process algebra Communicating Sequential Processes (CSP) and the Failures-Divergences Refinement which is the model checker for CSP [6]. Similar examples are obtained by Shmatikov and Stern [7] using Murphi, and Corin et al. [8] using symbolic traces and Pure-past Security - Linear Temporal Logic successfully.

This paper builds on this development, but with a different focus of interest. Relying on a well-established verification tool, the LySaTool, we shall iteratively attempt to remove components of the PKMv2 protocol and investigate the influence it has on the security properties. Our analysis shows that not only is the PKMv2 SA-TEK 3-Way Handshake secure, but that it can even be simplified by removal of some redundant fields without compromising the overall security protocol.

2 PKMv2 in IEEE 802.16e-2005

The second version of the Privacy and Key Management (PKMv2) protocol of IEEE 802.16 is described in IEEE 802.16e-2005 and aims to fix the bugs in the former version. In the first part of the protocol, an Authorization Key (AK) is generated using RSA or EAP or both. After that, each party generates a Key Encryption Key (KEK) using their AKs. KEKs are used in encrypting and distributing Traffic Encryption Keys (TEK), TEKs can be taken as session keys, while AK/KEK are long term keys. Then comes the second part, SA-TEK 3-Way Handshake, which lets MS to gather TEKs from BS. In the handshake, TEKs are encrypted by KEKs. The process can be seen in Fig. 1.

The important part of PKMv2 is the SA-TEK 3-Way Handshake. It is based on the second part of the former protocol, but now it has more security features. The original specification has three messages with H-MACs and in total twenty-one fields. The main fields are described in Table 1.

The PKMv2 SA-TEK 3-Way handshake sequence proceeds as shown in Table 2.

We had simplified the protocol, making use of the work of John Mitchell [14] (that was used in his security review together with IETF EAP Work Group), made the necessary changes that are necessary for LySa and obtained the follow-

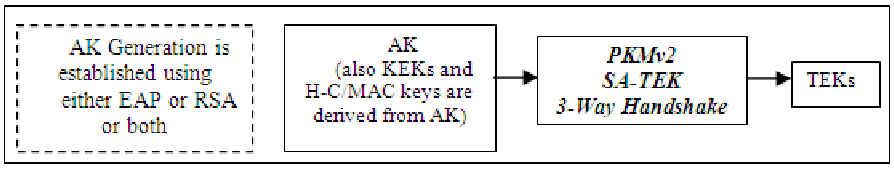
AK Generation is established using either EAP or RSA or both
AK (also KEKs and H-C/MAC keys are derived from AK)
PKMv2 SA-TEK 3-Way Handshake
TEKs

Fig. 1. The PKMv2 Process

Table 1. The PKMv2 SA-TEK 3-Way Handshake Protocol Fields

Attribute	**Content**
MS_Random	Random number received from MS
BS_Random	Random number included in SA-TEK-Challenge or SA-Challenge
KeySeqNo	AK Sequence Number
AKID	Id of the AK that was used for protecting this message
SA-TEK-Update	TEKs encrypted by KEKs, optionalonhandoveretc.
FrameNo	The frame number that old PMKs and AKs should be discarded
SA_Descriptors	Descriptors of the SA, only for initial entry
SecNegParam	Confirms messages security capabilities
HMAC/CMAC	Message Authentication Codes

Table 2. The PKMv2 SA-TEK 3-Way Handshake Protocol Narration

1. SA-TEK-Challenge
BS → MS: $BS\_Random, KeySeqNo, AKID, [KeyLifeTime], H-C/MAC$
2. SA-TEK-Request
MS → BS: $MS\_Random, BS\_Random, KeySeqNo, AKID, SecurityCapabilities, SecNegParam, PKMConfSettings, H-C/MAC$
3. SA-TEK-Response
BS → MS: $MS\_Random, BS\_Random, KeySeqNo, AKID, [SA-TEKUpdate], FrameNo, [SADescriptors], SecNegParam, H-C/MAC$

ing protocol narration in Table 3. A, B, id, na, nb, S, T, K stands for BS, MS, AKID, BS_Random, MS_Random, SecurityCapabilities + SecNegParam +PKM-ConfSettings, SA-TEKUpdate and AK, respectively.

The first message, named as PKMv2 SA-TEK-Challenge, includes a random number generated by A (na) and the id of the authorization key (id) and protected by the message authentication code (MAC).

The second message is the PKMv2 SA-TEK-Request which includes na and id received in the first message in addition to the random number generated by B, nb, security configuration details S and the message authentication code for the remaining fields.

Table 3. PKMv2 SA-TEK 3-Way Handshake Simplified Protocol Narration

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A → B: $na, nb, id, T, MAC\{na, nb, id, T\}_K$

Upon reception A checks the id, MAC and the na of and if any of these values are invalid, it discards the message. Otherwise, it checks the security capabilities provided by the B and if the properties does not match it reports this inconsistency to the higher layers.

If the second message is successfully validated by A then message 3 which is named as the PKMv2 SATEK-Response is sent to B. This message has the TEKs T.

If the last message is successfully verified by B using the MAC, the received TEKs and associated parameters will be installed by the B. The security negotiation parameters of A should also be verified by B but the failure of this verification may not cause halt of the protocol since B may continue by adopting the security negotiation parameters encoded in SA-TEK Response message.

This simplification of the protocol is verified using Murphi model checker in [14].

3 LySa Calculus

To analyze the IEEE 802.16 PKMv2 SA-TEK 3-Way Handshake protocol we need to formalize it in LySa calculus. LySa [10] is a process calculus based on the π-calculus [16] and incorporates cryptographic operations using ideas from the Spi-calculus [13]. However, there are two main differences between LySa and spi/pi calculus. First difference is that, LySa does not have channels but one global ether. That is because in usual implementations like ethernet-based or wireless, anyone can eavesdrop or act as an active attacker and that is definitely not the channel based communication. The second difference is about the usage of pattern matching in the expression of the tests associated with input and decryption.

LySa consists of terms and processes; terms consist of names (keys, nonces, messages, etc.), variables, public/private keys and the compositions of them using symmetric/asymmetric encryptions. The syntax of terms E is shown in Table 4.

The tuples of terms $E_1, \ldots, E_k$ are encrypted under a term E_0 representing a key in the cases of symmetric or asymmetric encryption. An assumption of perfect cryptography is adopted, meaning that the only inverse function of encryption is to use decryptions with the correct key.

The syntax of processes P which is mostly familiar to the polyadic Spi-calculus [13] is shown in Table 5.

Table 4. LySa Terms

$E ::= x$	variable
n	name
k^+/k^-	public and private keys
$\{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$	symmetric encryption
$\{\!\mid E_1, \ldots, E_k \mid\!\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$	asymmetric encryption

Table 5. LySa Processes

$P ::= 0$	nil
$P_1 \mid P_2$	parallel
$!P$	replication
$(\nu\, n)\, P$	restriction (name)
$(\nu_{\pm}\, m)\, P$	restriction (key pair)
$\langle E_1, \ldots, E_k \rangle P$	output
$(E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P$	input
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$	symmetric decrypt.
$\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\mid E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k \mid\!\}^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$	asymmetric decrypt.

The input operation with pattern matching will only succeed if the prefix of the message matches the terms specified before semi-colon in the input operation. The input process $(E_1, \ldots, E_j\ ;\ x_{j+1}, \ldots, x_k).P$ means that a k-tuple of values $(E'_1, \ldots, E'_k)$ is taken as the input and if the first $1 \leq i \leq j$ values E'_i are pairwise matched to the values E_i, the remaining k-j values of the input will be bound to the variables $x_{j+1}, \ldots, x_k$. In other words, the values before the semi-colon are to matched to the beginning part of the input and if the matching is successful the remaining part of the input will be assigned to variables after the semi-colon. This pattern matching is also used in decryptions as shown in Table 5. If no matching will be performed, then nothing is written before the semi-colon. Similarly, if no binding will be performed, then nothing is written after the semi-colon. For example,

$$P = \mathsf{decrypt}\ \{y\}_K\ \mathsf{as}\ \{x;\ \}_K P'$$

means that the decryption in P succeeds only if $x = y$ whereas

$$Q = \mathsf{decrypt}\ \{y\}_K\ \mathsf{as}\ \{;\ x\}_K Q'$$

means that the decryption in Q always succeeds, binding x to y. In both examples, the remainder processes P' and Q' are only executed if the decryptions succeed and of course P and Q have the implicit check that the length of the tuples are the same.

LySa syntax also have annotations for origin and destination in order to describe the intentions of the protocols. Encryptions can be annotated with fixed labels, called *crypto-points* defining its position in the process, and with *assertions* specifying the origin and destination of encrypted messages. Crypto-points ℓ are from some enumerable set $\mathcal{C}$ and added to state where the encryptions and decryptions occur. The LySa term for encryption:

$$\{E_1, \ldots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\,\mathcal{L}\,]$$

means that the encryption is created at crypto-points ℓ and specifies the intended crypto-points $\mathcal{L} \subseteq \mathcal{C}$ for decryption of the encrypted value in the assertion [dest $\mathcal{L}$]. Similarly, in the LySa term for decryption:

$$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}^{\ell}_{E_0}[\mathsf{orig}\,\mathcal{L}\,]\ \mathsf{in}\ P$$

[orig $\mathcal{L}$] specifies the crypto-points $\mathcal{L} \subseteq \mathcal{C}$ that E is allowed to have been encrypted.

The actual semantics have been omitted for lack of space, but are present in [10].

A LySa process may generate a large number of names which would cause infinite sets of names to be recorded. These sets are partitioned into finitely many equivalence classes for the efficiency of the analysis. A canonical value is a representative for each of these equivalence classes. For each name n, there is a canonical representative $\lfloor n \rfloor$ and extended homomorphically to terms, $\lfloor E \rfloor$ is the term where all names and variables are replaced by their canonical versions. Since it allows us to analyze an infinite number of principals, canonical value is an important analysis element [15].

3.1 LySa Model of IEEE 802.16 PKMv2 SA-TEK 3-Way Handshake

We are now ready to model the IEEE 802.16 PKMv2 SA-TEK 3-Way Handshake protocol in LySa. We have the protocol narration in Table 3. We extend the narration to distinguish between inputs and corresponding outputs and also make clear which checks must be performed [10]. Then we translate the narration into LySa by dividing the narration into two processes, one for each principal. Notice that the checks to be performed are represented by the pattern matchings on input and decryption. In the LySa specification we add annotations to all cryptographic operations as described before in this section. The LySa model of the PKMv2 SA-TEK 3-Way Handshake is given in Table 6.

In PKMv2, a keyed MAC is used to verify the integrity of messages. The message is hashed along with the key and then encrypted with the MAC key. Since the hash functions are one way functions, they can be modelled by using a public name for the encryption key and with no corresponding key for decryption. Therefore, the message is encrypted by asymmetric encryption first. After that symmetric encryption is applied. More details about LySa implementation can be found on [17] and [18].

Table 6. PKMv2 LySa Model

$$
\begin{array}{l}
(\nu\, K)\, (\nu\, id)\, (\\
!\ (\nu\, na)\, \langle id, na, \{\{\!|id, na|\!\}_{Hash}\}_K [\mathsf{at}\ a1\ \mathsf{dest}\,\{b1\}\,]\rangle. \\
\quad (na, id;\ xnb, xS, xmac). \\
\quad \mathsf{decrypt}\ xmac\ \mathsf{as}\ \{\{\!|na, id, xnb, xS|\!\}_{Hash};\ \}_K [\mathsf{at}\ a2\ \mathsf{orig}\,\{b2\}\,]\ \mathsf{in} \\
\quad (\nu\, T)\, \langle na, nb, id, T, \{\{\!|na, nb, id, T|\!\}_{Hash}\}_K [\mathsf{at}\ a3\ \mathsf{dest}\,\{b3\}\,]\rangle.0 \\
| \\
!\ (id;\ yna, ymac). \\
\quad \mathsf{decrypt}\ ymac\ \mathsf{as}\ \{\{\!|id, yna|\!\}_{Hash};\ \}_K [\mathsf{at}\ b1\ \mathsf{orig}\,\{a1\}\,]\ \mathsf{in} \\
\quad (\nu\, nb)\, (\nu\, S)\, \langle yna, id, nb, S, \{\{\!|yna, id, nb, S|\!\}_{Hash}\}_K [\mathsf{at}\ b2\ \mathsf{dest}\,\{a2\}\,]\rangle. \\
\quad (na, nb, id;\ yT, ymac). \\
\quad \mathsf{decrypt}\ ymac\ \mathsf{as}\ \{\{\!|na, nb, id, yT|\!\}_{Hash};\ \}_K [\mathsf{at}\ b3\ \mathsf{orig}\,\{a3\}\,]\ \mathsf{in}\ 0 \\
)
\end{array}
$$

4 Static Analysis

Static Analysis is a formal method which enables the security analysis of LySa processes. The analysis is based on tracking messages communicated on the network along with the possible values of the variables in the protocol and recording the potential violations of the destination/origin annotations.

A LySa process describes a set of possible operations, the analysis uses an over-approximation of this set, therefore the analysis could investigate a trace which is impossible at all. But this is needed to do a safe approximation because under-approximation could miss some traces.

The main components of the analysis are:

The variable environment ρ, an over-approximation of the potential values of each variable that may be bound to.

The network component κ, an over-approximation of the set of messages that can be communicated over the network

The error component ψ, the set of error messages of the form (ℓ, ℓ') indicating that something encrypted at ℓ was unexpectedly decrypted at ℓ'.

The details of the analysis and the proofs of the soundness of the analysis can be found in [11, 12].

In practice, the protocols - especially the ones in wireless networks - are executed in medium with malicious attackers. In this study, LySa processes are analyzed in parallel with Dolev-Yao attacker [9] which can perform operations like sending/receiving messages and encryption/decryption same as a legitimate principal.

We have new canonical name and variables (see section 3 on page 6) for the attacker: all the canonical names of the attacker are mapped to $n_\bullet$ and all the canonical variables of the attacker are mapped to $z_\bullet$. We also have $\ell_\bullet$ which is a crypto-point in the attacker, and we have the set $\mathcal{C}$ which is the set of crypto-points in the original process P in parallel with the attacker. Finally, there exists a public/private key-pair belonging to the attacker $m_\bullet^+$, $m_\bullet^-$.

The descriptions of the Dolev-Yao conditions are:

– The attacker initially has the knowledge of the canonical name $n_\bullet$ and all free names of the process P but he can improve his knowledge by eavesdropping all messages sent on the network.
– The attacker can improve his knowledge by decrypting messages with the keys he already knows. Unless the intended recipient of the message was attacker, an error $(\ell,\ell_\bullet)$ should be added to the error component ψ which means that something encrypted at ℓ was actually decrypted by the attacker at $\ell_\bullet$.
– The attacker can construct new encryptions using the keys he already knows. If this message is received and decrypted by a principal, then an error $(\ell_\bullet,\ell)$ should be added to the error component ψ which means that something encrypted at the attacker was decrypted by the attacker by a process P at ℓ.
– The attacker can send messages on the network using his knowledge and thus forge new communications.

This conditions enable the attacker to establish the attack scenarios including eavesdropping, modification and replay. The soundness of Dolev-Yao condition is proved in [10].

The flow of the analysis is shown in the Fig.2. First of all, we have a protocol narration as we had in section 2. Then we extend the narration and convert into LySa model, as we did in section 3. We also have our attacker model which is covered in this section. The LySa model is analyzed in parallel with the attacker model and is processed by the LySa-tool which implements the analysis. The results of the analysis are used to validate destination/origin authentication and confidentiality properties of the protocols. If no violation is detected, namely the error component ψ is empty, than it is guaranteed that the protocol satisfies the destination/origin authentication properties. Besides, the potential values that are learned by the attacker $(\rho(z_\bullet))$ helps us in validating the confidentiality properties. Since the analysis is an over-approximation there may occur false positives.

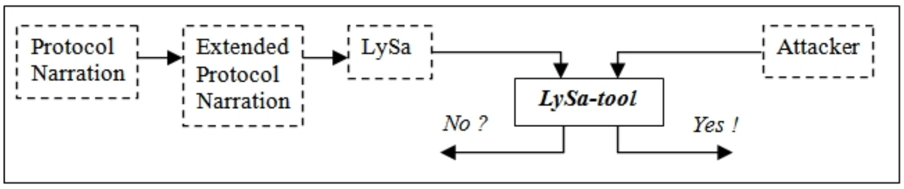

Fig. 2. The flow of the analysis

4.1 Scenarios

We shall analyze the PKMv2 SA-TEK 3-Way Handshake in scenarios with a number n of As, $A_1, \ldots, A_n$, and Bs, $B_1, \ldots, B_n$. As mentioned previously, A is the abbreviation for the base station and B for the mobile station.

We use the pair $(i, j))$ to refer to the instance of the protocol where A_i is communicating with the B_j. Thus we add the two indices i, j to all variables, constants and crypto-points of the model of the protocol in Table 6 and obtain the LySa code that allows the analysis to distinguish between the various instances. We then introduce the index 0 to refer to the attacker and the resulting analysis scenario takes the form

$$|_{i=1}^{n} ||_{j=0}^{n} P_{A_{i,j}} \quad | \quad |_{i=0}^{n} |_{j=1}^{n} P_{B_{i,j}}$$

Here we use $P_{A_{i,j}}$ and $P_{B_{i,j}}$ to denote the processes needed at the A_i and the B_j principals, respectively, in order to perform a mutual handshake. Notice that the scenario describes that not only are all principals ready to interact with all other honest principals, but the attacker is also allowed to act as a legitimate principal. The analysis is carried out for $n = 2$ which models two groups of As and Bs.

4.2 Validating the Protocol

In order to improve the PKMv2 SA-TEK 3-Way Handshake, first we have to verify the base protocol which is given in Table 3. The result of our static analysis is: *no violations possible.* This means that the protocol is secure and the attacker could not violate the authentication properties. This result is similar to the work in [14] which is established using model checking using Murphi. Now, we can make our modifications convenient with our experiment logic.

5 Simplifying the Protocol

Our approach is based on checking the limits of robustness in IEEE 802.16 PKMv2 by removing enhancements in PKMv2 one by one, and in different combinations. Thus, we can see if some improvements are unnecessary and the result may lead us to a simplified by still strong and secure protocol. Our experiments are accomplished using the LySa-tool which runs with our LySa code.

We based our model on the simplified version of the IEEE 802.16 PKMv2 SA-TEK 3-Way Handshake . After that we developed our LySa model in section 3.1. We start with our base protocol model and try to simplify the model by removing components and analyzing with attacker to find flaws.

We made the experiments systematically, and the road map of the experiment can be seen in Fig. 3. First, we start with the base protocol and show that it has no flaws. After that, we have three major paths: *Removing the Nonces, Removing the Ids* and *Removing both Nonces and Ids.* The shaded nodes in the figure shows the experiments with violations.

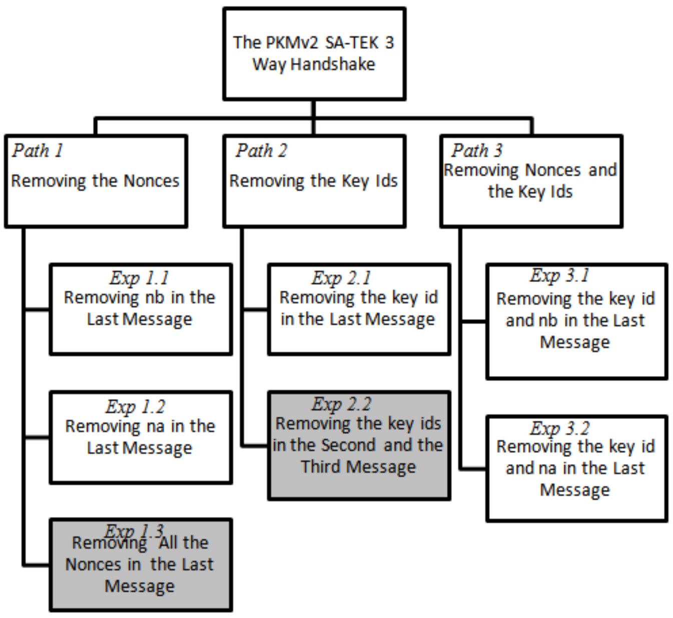
The PKMv2 SA-TEK 3 Way Handshake
Path 1
Removing the Nonces
Path 2
Removing the Key Ids
Path 3
Removing Nonces and the Key Ids
Exp 1.1
Removing nb in the Last Message
Exp 1.2
Removing na in the Last Message
Exp 1.3
Removing All the Nonces in the Last Message
Exp 2.1
Removing the key id in the Last Message
Exp 2.2
Removing the key ids in the Second and the Third Message
Exp 3.1
Removing the key id and nb in the Last Message
Exp 3.2
Removing the key id and na in the Last Message

Fig. 3. Experiment Road Map

In the first path, we start by removing the outermost nonces, namely the nonces in the last message. We remove one nonce at a time, and both nonces also. Therefore, we have three experiments about the nonces in the last message. In the second path, we remove the key ids. We start with the key id in the last message. Then we remove another key id which is actually in the second message.

In the last path, we join the successful experiments, in other words the modification of the base model where no flaws could be found. There are two successful experiments in the first path and one in the second, therefore we have two experiments in the last path.

5.1 Experiment 1.1

This experiment is the first part of the first path, removing the nonces. We remove *nb* in the last message. The protocol narration of this modification is given in Table 7 in the appendix section. The result of the analysis is: *no violations possible*. This means that the protocol is still secure and the attacker still could not violate the authentication properties even though we did not use the nonce of principal *B* in the last message. This is an interesting result because now the *na* in message two seems to be meaningless because there is no response for it. MAC's seem to save the protocol to verify the security properties. In addition, this is also an important result because it supports our assertion But we have to try the other combinations to conclude about the analysis.

5.2 Experiment 1.2

In this experiment, we remove the other nonce, *na* from the base protocol. The protocol narration of this modification is given in Table 8 in the appendix section. The result of the analysis is: *no violations possible.* This means that the protocol is still secure and the attacker still could not violate the authentication properties even though we did not use the nonce of principal A in the last message. Actually, this result supports our assertion and this is an optimized alternative to the protocol.

5.3 Experiment 1.3

In the last part of the first path, we remove both nonces from the last message of the base protocol. The protocol narration of this modification is given in Table 9 in the appendix section. This time we find violation of authentication properties. The result is given as:
$\psi = (a1_{11}, b3_{11}), (a3_{11}, b1_{11}), (a1_{21}, b3_{21}), (a3_{21}, b1_{21}), (a1_{12}, b3_{12}),$
$(a3_{12}, b1_{12}), (a1_{22}, b3_{22}), (a3_{22}, b1_{22})$
Sample trace for $(a1_{11}, b3_{11})$ can be shown as:

1. $\mathbf{A}_1 \rightarrow \mathbf{B}_1$ **:** id_{11}, na_{11}, MAC{ id_{11}, $\mathrm{na}_{11}\}_{K_{11}}$
1'. $\mathbf{A}_1 \rightarrow \mathbf{M(B}_1\mathbf{)}$ **:** id_{11}, na_{11}, MAC{ id_{11}, $\mathrm{na}_{11}\}_{K_{11}}$
2. $\mathbf{B}_1 \rightarrow \mathbf{A}_1$ **:** na_{11}, id_{11}, nb_{11}, S_{11}, MAC{ na_{11}, id_{11}, nb_{11}, $\mathrm{S}_{11}\}_{K_{11}}$
3. $\mathbf{M(A}_1\mathbf{)} \rightarrow \mathbf{B}_1$ **:** id_{11}, T_0, MAC{ id_{11}, $\mathrm{T}_0\}_{K_{11}}$

The results show that some encrypted values are decrypted in wrong places and some decrypted values were actually encrypted in the wrong places. The crypto-points are all from legitimate principals so there can be a replay attack. A possible trace of this error can be summarized as: the attacker eavesdropped the first message and he used the encrypted value in the first message, which is actually the MAC of the message, that he could not decrypt in a reply attack. In the third message, he replayed the MAC's, namely he used the MAC of message one in message-3. This is a flaw so we found a level that the protocol lost its robustness property.

This results show that in the implementation, the length of the fields are important. If somehow the lengths of the *na* value and the T value are the same, then there exists the security flaw.

5.4 Experiment 2.1

This experiment is the first part of the second path, removing the key ids. We start with removing the *id* only in the last message. The protocol narration of this modification is given in Table 10 in the appendix section. The result of the analysis is: *no violations possible.* This means that the protocol is still secure and the attacker still could not violate the authentication properties even though we did not use the key id in the last message.

5.5 Experiment 2.2

In this experiment we remove the id fields in both the last and the second message. The protocol narration of this modification is given in Table 11 in the appendix section. Now we have found violation of authentication properties. The result is given as:
$\psi = (b2_{11}, b3_{11}), (a3_{11}, a2_{11}), (b2_{21}, b3_{21}), (a3_{21}, a2_{21}), (b2_{12}, b3_{12}), (a3_{12}, a2_{12}),$
$(b2_{22}, b3_{22}), (a3_{22}, a2_{22}), (a3_{10}, a2_{10}), (a3_{20}, a2_{20}), (b2_{02}, b3_{02}), (b2_{01}, b3_{01})$
We found traces for specific types of violation. Sample trace for $(b2_{11}, b3_{11})$ can be shown as:

1. $\mathbf{A_1 \rightarrow B_1}$	**:** id$_{11}$, na$_{11}$, MAC{ id$_{11}$, na$_{11}$}$_{K_{11}}$
2. $\mathbf{B_1 \rightarrow A_1}$	**:** na$_{11}$, nb$_{11}$, S$_{11}$, MAC{ na$_{11}$, nb$_{11}$, S$_{11}$}$_{K_{11}}$
2'. $\mathbf{B_1 \rightarrow M(A_1)}$	**:** na$_{11}$, nb$_{11}$, S$_{11}$, MAC{ na$_{11}$, nb$_{11}$, S$_{11}$}$_{K_{11}}$
3. $\mathbf{M(A_1) \rightarrow B_1}$	**:** na$_{11}$, nb$_{11}$, S$_{11}$, MAC{ na$_{11}$, nb$_{11}$, S$_{11}$}$_{K_{11}}$

This result shows that we cannot remove both *id*s in the protocol.

5.6 Experiment 3.1

Now we construct the third path of the experiments by taking the successfully validated experiments of the first two path. Therefore, this path is called removing the nonces and the key ids. In this first experiment of this path, we combine the experiment 1.1 and 2.1 so we remove the key *id* and *nb* in the last message. The protocol narration of this modification is given in Table 12 in the appendix section. The result of the analysis is: *no violations possible*. This means that the protocol is still secure and the attacker still could not violate the authentication properties even though we did not use the key id and *nb* in the last message. Definitely, this is a better result and better optimization. But now *nb* in the second message is useless, therefore this result is not practical.

5.7 Experiment 3.2

In this experiment of this path, we combine the experiment 1.2 and 2.1 so we remove the key id and *nb* in the last message. The protocol narration of this modification is given in Table 13 in the appendix section. The result of the analysis is: *no violations possible*. This means that the protocol is still secure and the attacker still could not violate the authentication properties even though we did not use neither the key id nor nb in the last message.

Finally, this point is the best point of optimization since it is still secure and also practical. Namely, this version makes use of both nonces of *A* and *B* (actually BS and MS), and also key ids. Now we have seen the limits of the protocol and removed the redundant fields.

5.8 Further Experiments

The experiments above present enough evidence to support our proposal but we went further to see if we could fix the flaws that appeared when we removed fields from the base protocol. The problems occur in the MAC part, and we claim that this can be fixed by adding sequence numbers inside the MACs. Therefore, we applied sequence number revision to the experiments with erroneous results which can be seen as shaded in Fig. 3. The results have empty error components which mean that we have fixed the flaws. After that we took another way and decided to test the affect of the order of the fields in the messages. As can be seen in Fig. 4, we just swapped the appropriate fields in the last message in order to get the same order with the second message. The result is very interesting, now we have violations in the base code. Besides, we can easily find traces for those violations which means that the results are not false positive, but real flaws. We fixed this situation with the sequence numbers and had no violations afterwards.

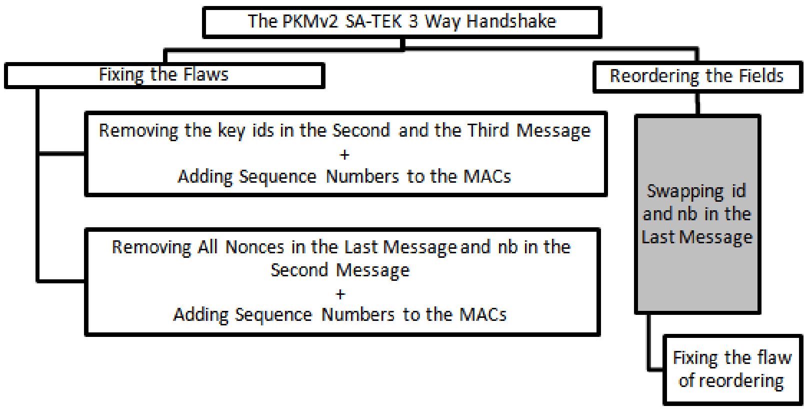
The PKMv2 SA-TEK 3 Way Handshake
Fixing the Flaws
Reordering the Fields
Removing the key ids in the Second and the Third Message + Adding Sequence Numbers to the MACs
Removing All Nonces in the Last Message and nb in the Second Message + Adding Sequence Numbers to the MACs
Swapping id and nb in the Last Message
Fixing the flaw of reordering

Fig. 4. Further Experiments

6 Conclusion

The contribution of this paper is two-fold. First, we establish the security of the Privacy and Key Management protocol PKMv2 of the latest version of the WiMAX standard, the IEEE 802.16e-2005. Second, we show how the protocol can be simplified without violating the security properties.

The PKMv2 SA-TEK 3-Way Handshake protocol was developed as a response to the identification of various flaws in the earlier version. Several security elements were added, not only to fix known flaws, but also to cater for other possible flaws. Our analysis shows that the resulting protocol is indeed secure, but also that it was over-secured by introducing redundant fields.

Our approach is based on the LySa calculus and the corresponding analysis tool, the LySaTool. The LySa framework provides an intuitive protocol description and an automated verification of the security properties authentication and confidentiality. Therefore LySa is a suitable choice for investigating protocols such as the classical security protocols [15], large-scale systems [18] and even voting protocols [12].

As we mentioned in the static analysis section, an error found by analysis does not always imply that the protocol actually has a flaw. A successful run on the analysis on the other hand, guarantees that the protocol does not violate the security properties. Thus, our experiments that yielded no violations are versions of the protocol that are guaranteed secure, whereas the experiments that returned violations would require further investigation to determine if these actually corresponded to an attack.

The success of our approach shows that is a viable method for iteratively simplifying a secure protocol. Only alterations that does not introduce possible attacks are kept and tested in combinations with other safe alterations. During development, we carried out many experiments, but only the ones we found of most interest were included in the paper. The result is a simplified version of PKMv2 SA-TEK 3-Way Handshake protocol, guaranteed to be secure, but without various redundant fields.

References

1. Johnston, D., Walker, J.: Overview of IEEE 802.16 Security. IEEE Security & Privacy Magazine Vol. 2, Issue: 3, (2004) 40–48
2. IEEE Std 802.16e-2005, 2006. Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1
3. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Springer-Verlag (1996) 147–166
4. Lowe, G.: An attack on the needham-schroeder public-key authentication protocol. Information Processing Letters **56(3)** (1995) 131–133
5. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Communications of the ACM. **21(12)** (1978) 993–999
6. Mitchell, J. C., Mitchell, M., Stern, U.: Automated Analysis of Cryptographic Protocols Using Murphi. IEEE Symposium on Security and Privacy. (1997) 141–151
7. Shmatikov, V., Stern, U.: Efficient finite-state analysis for large security protocols. Communications of the ACM. (1998) 106–115
8. Corin, R., Saptawijaya, A., Etalle, S.: A logic for constraint-based security protocol analysis. IEEE Symposium on Security and Privacy. (2006) 155–168
9. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory. **29(12)** (1983) 198–208

10. Bodei, C., Buchholtz, M., Degano, P., Nielson, H.R., Nielson, F.: Static Validation of Security Protocols. Journal of Computer Security. (2004) 347–390
11. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Automatic validation of protocol narration. Proceedings of the 16th Computer Security Foundations Workshop. (2003) 126–140
12. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag
13. Abadi, M., Gordon, A. D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation. **148(1)** (1999) 1–70
14. Datta, A., He, C., Mitchell, J. C., Roy, A., Sundararajan, M.: 802.16e Notes. IETF Liasons. (2005)
15. Buchholtz, M., Nielson, H.R., Nielson, F.: A calculus for control flow analysis of security protocols. International Journal on Information Security. **2(3-4)** (2004) 145–167
16. Milner, R.: Communicating and mobile systems: the pi-calculus. Cambridge University Press, fifth edition.
17. Nielsen, C.R., Andersen, E.H., Nielson, H.R.: Static Validation of a Voting Protocol. Nordic Journal of Computing . (2006) 98–116
18. Hansen, S.M., Skriver, J., Nielson, H.R.: Using Static Analysis to Validate the SAML Single Sign-On Protocol. Proceedings of the 2005 Workshop on Issues in the theory of Security . (2005) 27–40

A Protocol Narrations

Table 7. PKMv2 without nb in message 3

1. A $\rightarrow$ **B:** $id, na, MAC\{id, na\}_K$
2. B $\rightarrow$ **A:** $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A $\rightarrow$ **B:** $na, id, T, MAC\{na, id, T\}_K$

Table 8. PKMv2 without na in message 3

1. A $\rightarrow$ **B:** $id, na, MAC\{id, na\}_K$
2. B $\rightarrow$ **A:** $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A $\rightarrow$ **B:** $nb, id, T, MAC\{nb, id, T\}_K$

Table 9. PKMv2 without na and nb in message 3

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A → B: $id, T, MAC\{id, T\}_K$

Table 10. PKMv2 without id in message 3

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A → B: $na, nb, T, MAC\{na, nb, T\}_K$

Table 11. PKMv2 without ids in message 2 and 3

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, nb, S, MAC\{na, nb, S\}_K$
3. A → B: $na, nb, T, MAC\{na, nb, T\}_K$

Table 12. PKMv2 without id and nb in message 3

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A → B: $na, T, MAC\{na, T\}_K$

Table 13. PKMv2 without id and na in message 3

1. A → B: $id, na, MAC\{id, na\}_K$
2. B → A: $na, id, nb, S, MAC\{na, id, nb, S\}_K$
3. A → B: $nb, T, MAC\{nb, T\}_K$