# SECURITY PROTOCOL FOR POLLUTION ATTACK USING NETWORK CODING

Kiattikul Sooksomsatarn

Department of Computer Science,
School of Information and Communication Technology, University of Phayao,
Maeka, Muang, Phayao 56000, Thailand

## *ABSTRACT*

*Network coding is a technique for maximizing the use of available bandwidth capacity. We are interested in applying network coding to multimedia content distribution. This is desirable because many popular network applications for content distribution consume high bandwidth and international bandwidth; both are scarce in countries such as New Zealand. Existing work has addressed the use of network coding for content distribution, however work on network coding and security does not consider the trade-off between quality of service and security for multimedia. Network coding is vulnerable to a pollution attack or a packet modification attack. It has detrimental effect particularly on network coding because of specific characteristic of network coding that allows nodes to modify received packets at any time. Many pollution attack defence mechanisms use computationally expensive techniques leading to higher communication cost. Therefore, the focus of this work is on developing protocols to address both open problems and validate the protocols using a combination of formal and simulation techniques. More importantly, our novel contribution is reduction of complexity of algorithms appropriate for streaming content distribution with network coding.*

## *KEYWORDS*

*Network Coding, Pollution Attack Detection, Security Protocol*

## 1. INTRODUCTION

Previous work using homomorphic Message Authentication Codes (homomorphic MACs) to detect corrupted packets, such as, the work by Agrawal and Boneh [1] and Li et al. [2], leverages the observation that if a packet does not belong to the source space, then it is a corrupted packet. The detection works by firstly establishing shared secret keys between the source and the intermediate nodes. Then, using these secret keys, the source node can sign the fixed source space and the intermediate nodes can verify if their received packets belong to the source space. However, the fixed source space can cause another attack called tag-pollution attack since an attacker tries to produce a new valid tag from the fixed source space he/she has received.

In cooperative SpaceMac's detection scheme [3], Le and Markopoulou leverage the observation that a packet sent by an intermediate node must belong to the space spanned by all packets that it received from its parents. For example, consider a subset of nodes in a network shown in Figure 1. A packet sent by *C* must belong to the space spanned by the packets it received from its

Dhinaharan Nagamalai et al. (Eds) : CSEIT, NCS, SPM, NeTCoM - 2018
pp. 209–221, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.81816

parents: *A* and *B*; otherwise, *C* must be polluting the network. Formally, at any moment *t* in the multi-cast session, if an intermediate node *N* sends out a vector *y* then $y \in \Pi_N(t)$; otherwise, *y* is corrupted.

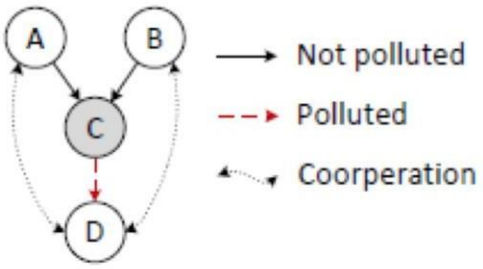


Fig. 1. Pollution Attack Detection in SpaceMac[3] (Note that "coor-peration" should be "cooperation")

Figure 1 illustrates how SpaceMac helps to detect pollution attacks. Using SpaceMac, *A* and *B* are able to sign the expanding space $\Pi_C$ (the received space of *C*) and *D* is able to verify any packet sent by C to see if it belongs to $\Pi_C$. If there is a packet sent by C that is not in $\Pi_C$, the attack is detected by *D*. The cooperation among *A*, *B*, and *D* helps to detect the attack from *C*. Our detection scheme uses loose synchronization by firstly establishing Kerberos ticket only shared between the source and the TCC for preventing tag-pollution and replay attack. Then, using pull-based algorithm, a downstream node can make a tag request. The downstream node must have the valid tag (*SourceSpace*) generated by the source from the original ticket to have an access for content distribution. A destination node or another downstream node do not need to interact with the source anymore if its parent has the valid tag since the parent can generate the new valid tag (*Sub-Space*). There are numbers of TCCs as replicas throughout the network. The upstream nodes can interact with the closest TCC as many times as required further down to the destination taking the role of the source to generate a new tag.

## 2. DEFINITIONS

### 2.1. Packet Identity

Size of content for distributing is definitely different. The content needs to be divided to smaller size called a *packet*. All the packets are the same in size for ease of content distribution using network coding. To establish authentication tag of packets, the packets need to be uniquely identified. The requirement of identifying packets is complicated by the fact that packets could be modified, either maliciously or as a requirement of functionality. Ordinary hash functions are no longer applicable when network coding is applied. A hash value is not robust under modifications of the packet it identifies. Any change to the packet will result in a new identity. A homomorphic hash is a function that can compute the hash of a combined packet from the hashes of the individual packets. With this construction, we can dis- tribute a list of individual hashes to nodes, and they can use those to verify incoming packets once they arrive. Homomorphic Hashing is described in the paper On- the-fly verification of rate less erasure codes for efficient content distribution by Krohn et al. [4] Therefore, we define a homomorphic hash value as a basic form of

identification where homomorphic hash function *HH* is applied to the packet p to produce the identity of the packet $id_p = HH(p)$

## 2.2. Kerberos Ticket

Kerberos is a computer network authentication protocol which works based on tickets to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Kerberos protocol messages are protected against eavesdropping and replay attacks. Kerberos builds on symmetric key cryptography and requires a trusted third party, and optionally may use public-key cryptography during certain phases of authentication. In our protocol, to establish loose synchronization, the source creates and signs a ticket for the packet being sent out. This ticket contains information to identify the packet, as well as any extra terms of the ticket such as duration of the ticket. As the pollution attack detection protocol sends ticket information separately from the packets themselves, there is no need to embed the ticket information in the packets.

## 2.3. Structure of Authentication Tag

In the pollution attack detection protocol, tags are used to represent authentication of origin and are passed from the source to the destination via intermediate nodes. A tag is defined as a tuple:

$$tag = \{A = T_{p,D}^{(n)}, B = pk_D^{(n)}\}_{sk_{TCC}}$$

The parameters of the tag are:

- $A = T_{p,D}^{(n)} = \{id_p = HH(p), Ticket_p\}_{sk_p}$: The one-time ticket signed with the secret key for the packet $sk_p$. This ticket contains information such as the identity of the packet $id_p = HH(p)$ and the original Ticket for the packet generated beforehand $\text{Ticket}_p$.
- $B = pk_D^{(n)}$: The one-time public key created by the downstream node *D* using nonce *n*.
- The parameters (*A and B*) are signed with the secret key of the TCC.

The pollution attack detection protocol uses an encryption scheme. The TCC signs tags and the public signing key of the TCC has to be well-known to verify the signed tags. The public encryption key of the TCC is also well-known so that the source can encrypt messages to send to the TCC. Before the protocol is run for the first time, the TCC generates and publishes its public encryption and signing keys. The notation $A_{skB}$ denotes the message *A* signed using the key *skB* and $A_{pkB}$ denotes the message *A* encrypted using the key *pkB*.

## 2.4. Encryption Scheme

The pollution attack detection protocol requires an encryption scheme that provides indistinguishability under Chosen Plaintext Attacks (IND-CPA), which is a security definition for private- or public-key encryption schemes. At a high level, IND-CPA security means that no adversary can distinguish between different messages, even when allowed to make encryptions and decryptions of its choice.

A cryptosystem is indistinguishable under chosen plaintext attack if every probabilistic polynomial time adversary has only a negligible advantage over random guessing with probability $\frac{1}{2} + \epsilon(k)$, where is a negligible function in the security parameter $k$.

The two most commonly used encryption schemes are the RSA encryption scheme and the El-Gamal encryption scheme. Nonetheless, both encryption schemes are not well applicable to our protocol because of stream ciphered homomorphism. Therefore, Goldwasser-Micali cyyptosystem scheme is used in our protocol.

Goldwasser-Micali (GM) cryptosystem is an asymmetric key encryption algorithm developed by Shafi Goldwasser and Silvio Micali in 1982 [5]. The GM encryption scheme is semantically secure if any probabilistic, polynomial-time algorithm (PPTA) that is given the ciphertext of a certain message m and the message's length, cannot determine any partial information on the message with probability non-negligibly higher than all other PPTA's that only have access to the message length.

Two years later, Goldwasser and Micali subsequently demonstrated that semantic security is equivalent to another definition of security called ciphertext indistinguishability under chosen-plaintext attack [6].

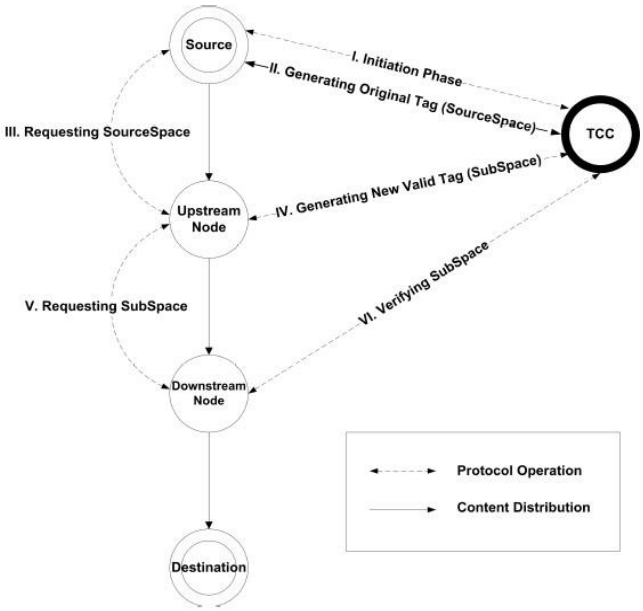


Fig. 2. Main phases of protocol for Pollution Attack Detection

## 3. OVERVIEW OF THE PROTOCOL FOR POLLUTION ATTACK DETECTION (PAD)

The main phases of the pollution attack detection protocol include:

## 3.1. Initiation Phase

The *Source* initiates dividing the content into chunks called packets and assigning their identity, cooperates with the *TCC*, creates initial secret/public keys for the packets. The *TCC* creates *Kerberos Ticket* from received information by using a trap-door function.

## 3.2. Sourcespace Phase

This phase includes two mechanisms:

### 3.2.1. Generating Original Tag (SourceSpace)

Using the Ticket for that packet, the Source generates a one-time ticket for the requesting node and the requested packet beforehand. The one-time ticket is then signed by the secret key only shared between the Source and the *TCC*. The *TCC* creates and sign a authentication tag called SourceSpace.

### 3.2.2. Requesting SourceSpace

The Upstream Node makes a request of original tag generated from the Source for downloading a desire packet. The Source sends the tag to the requesting node with the packet that the node desires**.**

## 3.3. Subspace Phase

This phase includes three mechanisms:

### 3.3.1. Generating New Valid Tag (SubSpace)

The Upstream Node takes the role of the Source creating a propagated new valid tag called SubSpace. If a Downstream Node wishes to redistribute the packet that they have received from the Upstream Node, they can take the role of the Upstream Node and send it to another Downstream Node(s).

### 3.3.2. Requesting SubSpace

This is like Requesting SourceSpace, but the Downstream Node no longer needs to interact with Source because the Upstream Node can also generate the valid tag. However, tag verification is needed in next method**.**

### 3.3.3. Verifying SubSpace

This provides a method for the Downstream Node to check whether the tag of the packet they received is valid.

# 4. DETAILED PROTOCOL FOR PAD

This section identifies the protocol for Pollution Attack Detection in more detail. The protocol consists of three phases: Initiation phase, SourceSpace phase, and SubSpace phase.

## 4.1. Initiation Phase

This section presents the initiation phase of the pollution attack detection protocol.

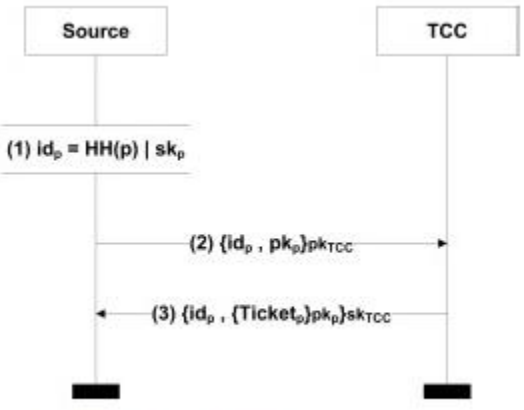


Fig. 3. Handshake between Source and TCC

Figure 3 shows a handshake between Source and TCC for sharing public information of packets. A protocol for content distribution using network coding is defined by a set of five probabilistic, polynomial-time, multi-party algorithms: SetupKey, LooseSync, GenLicense, RegenLicense and VerifyLicense.

### 4.1.1. SetupKey(k)

A probabilistic polynomial time (PPT) algorithm that sets up keys and global parameters necessary for the protocol with security parameter *k*.

### 4.1.2. LooseSync(packet, $pk_{packet}$)

A polynomial time algorithm where the source assigns to packet with some public information $pk_{packet}$ where only the source knows the corresponding private information $sk_{packet}$. Returns 1 or 0 to indicate success or failure of the assignment.

### 4.1.3. GenLicense(packet, data, $sk_{packet}$)

A PPT algorithm which returns license for packet. The algorithm generates the license using the secret information for the packet $sk_{packet}$ and content.

### 4.1.4. RegenLicense(packet, $license_{old}$, $content_{old}$, $content_{new}$)

A PPT algorithm which returns $license_{new}$ for packet. The algorithm generates $license_{new}$ using

$content_{new}$ as well as $license_{old}$ and $content_{old}$ from an existing license for packet.

#### 4.1.5. VerifyLicense(packet, content, license)

A polynomial time algorithm that verifies the correctness of license for packet and content and returns 1 or 0.

The use of the GenLicense and RegenLicense algorithms will result in a set of tags $license_1$, ..., $license_n$ with corresponding content values $content_1$, ..., $content_n$. The following formula express the correctness property:

$$VerifyTag(packet, content_i, license_i) = 1$$

Where $license_1$ = GenLicense(packet, $content_1$, $sk_{packet}$) and $license_i$ = RegenLicense(packet, $license_{i1}$, $content_{i1}$, $content_i$)

### 4.2. Sourcespace Phase

To initially generate a tag, a protocol takes place between the intermediate (downstream) node, the source, and the TCC. The generation of a new tag for a packet by the source takes place in the seven steps shown in Figure 4.

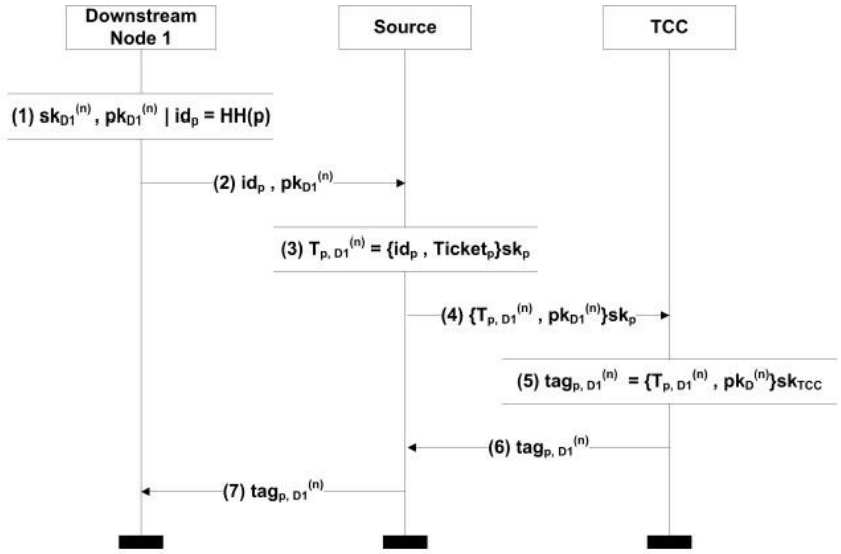


Fig. 4. Source Generating Tag with TCC

### 4.3. Subspace Phase

Now the one-time tag for a packet has been generated, signed, and sent to the downstream node, the downstream node can take the role of the source using this tag to generate a new tag for a

destination or another downstream node without interacting with the source. The generation of a new tag by the upstream node takes place in the seven steps shown in Figure 5.

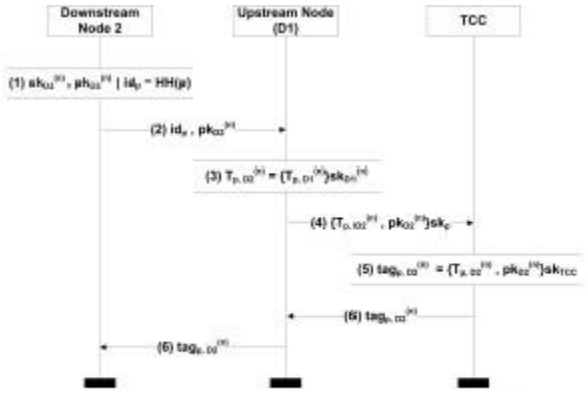


Fig. 5. Upstream Node Generating One-time Tag with TCC for Destination or Another Downstream Node On-the-fly

On-the-fly tag generation has the trade-off between computational/network overheads and security. In content distribution using network coding, homomorphic cryptography is applied to verify Message Authentication Code (MAC) tags without decrypting incoming signed, combined packets.

## 5. SECURITY ANALYSES

This section demonstrates two formal security analyses of the protocol for PAD. Firstly, a reduction to contradiction style of argument is used to show the PAD protocol provides security against modification, fabrication, collusion, and spoofing attacks. Secondly, a formal technique to analyse our security protocol, Communicating Sequential Processing (CSP), is used to formalise our protocol running on Failures-Divergence Refinement (FDR) model checker to show the protocol also provides security against authentication attack.

### 5.1. Security Proof By Contradiction

**Theorem 1.** *The protocol for pollution attack detection provides Secure Content Distribution using Network Coding [7] in the random oracle model provided that the signature scheme used has provable security against existential forgeries under adaptive chosen message attacks and the encryption scheme used has provable security against IND-CCA2 attacks.*

We use a reduction to contradiction style of argument to show the tagged transaction protocol provides security against spoofing, fabrication, network sniffing, and cloning attacks. then make arguments showing the security of the tagged transaction protocol against identity revelation and linkability attacks. In this security analysis, the TGC is assumed to be acting as a trusted third party. Chapter 6 removes this assumption and discusses methods to verify the actions of the TGC. For the security analysis of the identity revelation and linkability properties the following assumptions are made: a perfect anonymous communication channel, an anonymous supplier, and the parties in the protocol not revealing their identities or the identities of the parties with whom they communicate. This security analysis does not consider side channel attacks.

## 5.2. Formal Security Modelling Analysis

### 5.2.1. Modelling the Honest Agents

We now describe how we can model the honest agents running the protocol as CSP processes. We give a parameterised process *Initiator (A, kA)* to represent an agent a running the protocol as initiator and using session key *kA*. The process starts by receiving a message m from the environment, telling it with whom to run the protocol. It then sends an appropriate message 1 *m1* and receives back an appropriate message 2 *m2* containing an arbitrary value for nonce of responder *nB*.

$$Initiator(A, k_A) =$$
$$\begin{array}{ll} \Box & \left( \begin{array}{l} env.A.(Env0, B) \rightarrow \\ send.A.B.A.n_A \rightarrow \\ receive.T.A.\{B.k_{AB}.n_A.n_B\}_{k_{AT}}.m1 \rightarrow \\ send.A.B.m2.\{n_B\}_{k_{AB}} \rightarrow \\ Session(A, B, k_{AB}, n_A, n_B) \end{array} \right) \\ B \in Agent & \\ k_{AB} \in Key & \\ n_B \in Nonce & \\ m \in Message & \end{array}$$

The definition of the responder is similar: the process Responder (B, $n_B$) represents agent B running the protocol as responder using nonce $n_B$. The responder starts by receiving a message 1 m1, from an arbitrary agent a and containing an arbitrary session key k. It then sends back the corresponding message 2 m2.

$$Responder(B, n_B) =$$
$$\begin{array}{ll} \Box & \left( \begin{array}{l} receive.A.B.n_A \rightarrow \\ send.B.T\{B.k_{AB}.n_A.n_B\}_{k_{AT}}.m1 \rightarrow \\ send.A.B.m2.\{n_B\}_{k_{AB}} \rightarrow \\ Session(A, B, k_{AB}, n_A, n_B) \end{array} \right) \\ k_{AB} \in Key & \\ A \in Agent & \\ n_A \in Nonce & \end{array}$$

As noted above, we consider a small system, comprising Alice acting as initiator, using key kA, and Bob acting as responder, using nonce nB. The two agents do not communicate directly: we arrange below for all communications to go via the attacker. We model this as an interleaving.

$$System_0 = Initiator(Alice, k_A) ||| Responder(Bob, n_B).$$

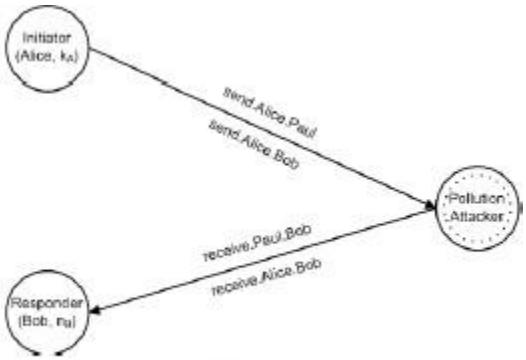


Fig. 6. Pollution Attacker Model

Of course, it is straightforward to consider larger systems, with more agents, or with particular agents running the protocol multiple times, perhaps with different roles.

### 5.2.2. Modeling the Attacker

We now describe how we can model the attacker. The main issue is modeling which messages the attacker can understand and to create. We need to keep track, therefore, of which submessages of protocol messages the attacker knows; we term these Facts:

$$\begin{aligned} Facts = & \{\{pk_B\}_{(sk_A)_k} | k \in SessionKey, A, B \in Agent\} \cup \\ & \{\{k\}_n | k \in SessionKey, n \in Nonce\} \cup \\ & \{\{sk_A\}_k | k \in SessionKey, A \in Agent\} \cup \\ & Agent \cup Nonce \cup SessionKey \cup \\ & SecretKey \cup PublicKey. \end{aligned}$$

If the Attacker knows a fact f and a key *k* then he can encrypt *f* with *k*; if he knows an encrypted message and the corresponding decryption key, he can perform the decryption to obtain the body; if he knows a collection of facts, he can concatenate them together; if he knows a concatenation, he can split it up into the individual components.

### 5.2.3. Requesting SourceSpace

The Upstream Node makes a request of original tag generated from the Source for downloading a desire packet.

## 5.3. Security Analysis

Showing that the pollution attacks detection protocol provides secure network coding consists of showing proofs by contradiction to show security against colluding, packet sniffing, spoofing, and fabrication attacks. If there exists an attacker that can break the security properties of the pollution attacks detection protocol, then this attacker can be used to solve a problem thought to be hard.

In CSP model, we describe the basic technique of CSP model checking of security protocols. We consider a small system running the protocol and conclude a single initiator Alice, who will use the session key *Ka*, and a single responder Bob, who will use the secret *Sb*. We also include a pollution attacker, Paul, who has complete control over the network.

We now consider authentication of the responder to the initiator, and vice versa. More precisely, we consider the following questions:

1) If an initiator *A* completes a run of the protocol, apparently with *A*, then has *A* been running the protocol, apparently with *A*, and do they agree upon the value of the nonce *n* and the session key *k*?

2) If a responder *B* completes a run of the protocol, apparently with *A*, then has *A* been running the protocol, apparently with *A*, and do they agree upon the value of the session key *k*? (Note that *A* can receive no guarantee that he and *A* agree upon *n*, because he cannot be sure that *A* even receives message 2.)

We describe how to test for the authentication property. We introduce new events, as follows:

• The event *Running.InitiatorRole.A.B.k* indicates that *A* thinks that she is running the protocol as initiator, apparently with *B*, using session key *k*.

• The event *Complete.ResponderRole.B.A.k* indicates that *B* thinks he has completed a run of the protocol as responder, apparently with *A*, using session key *k*.

We will then check that whenever the latter event occurs, the former event has previously occurred. We arrange for initiator *A* to perform the Running event when she sends message 1, and we arrange for responder *B* to perform the Complete event when he sends message 2; we hide all other events.

$$
\begin{aligned}
AuthSystem_0 = \\
&System[Running.InitiatorRole.A.B.k/ \\
&send.A.B.(m1, \{pk_B b\}_{(sk_A)_k}), \\
&Complete.ResponderRole.B.A.k/ \\
&send.B.A.(m2, \{k\}n) \\
&A, B \in Agent, k \in SessionKey, n \in Nonce] \\
&(\Sigma - alphaAuthSystem),
\end{aligned}
$$

$$
\begin{aligned}
alphaAuthSystem = \\
&\{Running.InitiatorRole.A.B, \\
&Complete.ResponderRole.B.A \\
&A, B \in Honest.
\end{aligned}
$$

More generally, the Complete event is performed at the last step in the protocol taken by that agent, and the Running event is performed when the agent sends a message that should be causally linked to the other agent receiving a message. Recall that we want to check that whenever a responder A performs a Complete event concerning initiator A, then a has previously performed a corresponding Running event concerning B. We therefore consider the following specification process, which allows only such traces

$$
\begin{aligned}
AuthSpec = \\
&Running.InitiatorRole?A.B.k \rightarrow \\
&Chaos(\{Complete.ResponderRole.B.A.k\}).
\end{aligned}
$$

Note that this specification allows B to perform an arbitrary number of Complete events corresponding to a single Running event, and so does not insist that there is a one-one relationship between the runs of A and the runs of B. We could test for such a relationship by replacing the Chaos(Complete.ResponderRole.B.A.k) by Complete.ResponderRole.B.A.k → STOP. We can use FDR to test the renement

$$AuthSpec \sqsubseteq_T AuthSystem.$$

(The above renement test is appropriate since *AuthSystem* performs at most a single *Running* event; for a system that could perform *N* such events, we would replace the left-hand side of the renement test by an interleaving of N copies of *AuthSpec*.) FDR nds that this renement does not hold, and returns the following witness trace:

$$< Complete.ResponderRole.Bob.Alice.k_A >$$

Bob thinks he has completed a run of the protocol with Alice, but Alice did not think that she was running the protocol with Bob. We can again use the FDR debugger to nd the corresponding trace of System:

$$\begin{aligned} < env.Alice.(Env0, Paul) \\ & send.Alice.Paul. \\ & (m1, \{pk_P\}_{\{sk_A\}_{k_A}}) \\ & receive.Alice.Bob.(m1, \{pk_B\}_{\{sk_A\}_{k_A}}) \\ & send.Bob.Alice.(m2, \{k_{A1}\}_{n_B}) > . \end{aligned}$$

We can test whether the responder is authenticated to the initiator (item 1 above) in a similar way. FDR nds no attack in this case. It is interesting to consider what guarantees the responder does receive from the protocol. We claim that if responder B completes a run of the protocol, apparently with A, then A has been running the protocol, and that they agree upon the value of the session key k. Note though that A might have been running the protocol with some agent C other than B, and so performed a Running.InitiatorRole.Alice.C.k event. We can test this condition using the renement check

$$AlivenessSpec \sqsubseteq_T SystemAliveness,$$

, where

$$\begin{aligned} AlivenessSpec = \\ & Running.InitiatorRole.Alice?C?k \rightarrow \\ & Chaos(\{Complete.ResponderRole.B.Alice.k \\ & | B \in Agent\}), \\ \\ SystemAliveness = \\ & AuthSystem_0 \backslash (\Sigma - alphaSystemAliveness), \\ \\ alphaSystemAliveness = \\ & \{Running.InitiatorRole.A.B, \\ & Complete.ResponderRole.B.A \\ & | A \in Honest, B \in Agent\}. \end{aligned}$$

## 6. CONCLUSIONS

Using a proof by contradiction we have shown that the pollution attacks detection protocol provides protection against colluding, packet sniffing, spoofing, and fabrication attacks in the random oracle model. Further security analysis has shown that the pollution attacks detection protocol also provides protection against pollution attacks as we test authentication property of the responder to the initiator for, and vice versa. We claim that if the initiator A completes a run of the protocol, apparently with B, then B has been running the protocol, and they do agree upon the value of the session key k. Therefore, FDR finds no authentication attacks in this case.

### REFERENCES

[1] S. Agrawal and D. Boneh, (2009) “Homomorphic macs: Macbased integrity for network coding,” in Proceedings of the 7th International Conference on Applied Cryptography and Network Security, ser. ACNS ’09. Berlin, Heidelberg: Springer-Verlag, pp. 292–305.

[2] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, (2010) “Ripple authentication for network coding,” in INFOCOM, 2010 Proceedings IEEE, pp. 1–9.

[3] A. Le and A. Markopoulou, (2012) “Cooperative defense against pollution attacks in network coding using spacemac,” Selected Areas in Communications, IEEE Journal on, vol. 30, no. 2, pp. 442–449.

[4] M. Krohn, M. Freedman, and D. Mazieres, (2004) "On-the-fly verification of rateless erasure codes for efficient content distribution," in Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on, pp. 226–240.

[5] S. Goldwasser and S. Micali, (1982) "Probabilistic encryption and how to play mental poker keeping secret all partial information," in Proc. 14th Symposium on Theory of Computing, p. 365377.

[6] S. Goldwasser and S. Micali , (1984) "Probabilistic encryption," Journal of Computer and System Sciences, vol. 28, no. 2, p. 270299.

[7] K. Sooksomsatarn, I. Welch, and W. Seah, (2010) "Secure content distribution using network coding," in Proc. 8th New Zealand Computer Science Research Student Conference.