USENIX'23 Artifact Appendix: Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations

Cas Cremers
CISPA Helmholtz Center for Information Security

Charlie Jacomme
Inria Paris, France

Aurora Naska
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

This artifact includes formal models and proofs of the Signal protocol with an abstraction of its session-handling layer Sesame and ratcheting mechanism of the Double Ratchet. We prove that Signal with a session-handling layer does not achieve the Post-Compromise Security (PCS) guarantee, *i.e, the conversation is secure after a healing phase following the compromise*, although it holds in a single-session Signal. Following this, we propose and model a mechanism on how to restore PCS, and in a second step how to detect clone attackers in the conversation.

We provide four models of Signal: a) single-session Signal from the literature, where the PCS guarantee holds, b) Signal with its session-handling layer Sesame, where an attacker breaks PCS, c) Signal with our PCS-fix, with the restored PCS guarantee, and d) Signal with a clone detection mechanism, that soundly detects the clone's activity, *i.e, detection without any false positives*.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

Our models and proofs are accessible for inspection and reproduction at https://github.com/sesame-symbolic-model/sesame-model. To clone the repository, the user should run:

```
$ git clone --branch "sesame-model-v1"
https://github.com/sesame-symbolic-model/
sesame-model.git
```

A.2.3 Hardware dependencies

We have run our experiments on a Intel(R) Xeon(R) CPU E5-4650L 2.60GHz server with 756GB of RAM, and 4 threads per Tamarin call. The time of the experiments' execution might vary depending on your machine's CPU and RAM.

A.2.4 Software dependencies

To evaluate our models, you need the Tamarin prover[1] tool version v1.7.1. As shown in Appendix A.3.1, either you can download a docker image with the preinstalled Tamarin prover version, or compile it from scratch using the provided source files of the used version.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

The following instructions have been tested on a Linux machine, and there may be slight variations on how to install and start the docker client on other systems. There are two ways to install the Tamarin prover and reproduce our results:

- **Using Docker** We provide via dockerhub an anonymous docker with the required preinstalled version of Tamarin.
  1. The user should install the Docker Engine on their machine as instructed in https://docs.docker.com/engine/install.
  2. Fetch the provided Tamarin image via:
     ```
     $ docker pull sesameproof/tamarin
     ```
  3. From the cloned repository, or one that has the Sesame folder inside of it, run:
     ```
     $ docker run -it -v "$PWD:/opt/case-studies"
     sesameproof/tamarin bash
     ```

[1] https://tamarin-prover.github.io

This should give you a shell, where the commands *tamarin-prover* of the experiments can be executed.

- **Compiling Tamarin from source** We provide in the repository the folder `tamarin-prover.zip` containing the source files for the correct version of Tamarin, with installation instructions at https://tamarin-prover.github.io/manual/book/002_installation.html.

A.3.2 Basic Test

To test that the Tamarin Prover is installed correctly, run:

```
$ tamarin-prover test
```

You should see the following message in the terminal:

```
*** TEST SUMMARY ***
All tests successful.
The tamarin-prover should work as  intended.
:-) happy proving (-:
```

A.4 Evaluation workflow

A.4.1 Major Claims

In the following we list the main properties of our formal analysis:

(PCS in single-session): Single-session Signal (*Double Ratchet* model) achieves the PCS guarantee in a conversation between two users, i.e., the conversation is secure after the healing phase and the clone attacker cannot inject or decrypt new messages. In this case, session-based and conversation-based PCS collapse and are both proven by experiment **(E1)** and described in Section 5.3.

(PCS violation in Sesame): Signal with the session-handling layer Sesame (*Sesame* model) does not achieve conversation-based PCS. We show an attack where a clone can impersonate the victim by using the compromised session, even though the honest parties heal after the compromise. The violation is shown in experiment **(E2)** and described in Section 5.4.

(Restored PCS): Signal with session-handling and proposed PCS-fix from Section 4.1 (*Sesame with sequential sessions* model) restores conversation-based PCS and locks the attacker out of the conversation. The property is proven in experiment **(E3)** and described in Section 5.5.

(Sound Clone Detection): In Sesame with sequential sessions and the proposed clone detection mechanism in Section 4.1 (*Sesame with sequential sessions + warning message* model), the parties can soundly detect a clone attacker that impersonates the victim by initiating new sessions with the compromise key. The property is shown in experiment **(E4)** and described in Section 5.5.

A summary of the results is reported in Table 2 of our paper.

A.4.2 Experiments

We now show the experiment steps needed to reproduce our results and prove the main claims we listed in the previous section.

Preparation Before running our experiments, the user should have followed one of the installation steps from Appendix A.3.1. In case of **using Docker**, the user runs the command from the second step to get a shell for the execution of `tamarin-prover` commands. In the second case of **compiling from source**, the user opens a terminal inside the `Sesame` folder in the cloned repository.

(E1): [5 human-minutes + ~1 compute-minute]

We prove the session-based and conversation-based PCS on the Signal model with a single-session restriction.

Execution: In the shell from the preparation step, the user executes:

```
$ tamarin-prover --prove Sesame/
DoubleRatchet.spthy
```

Results: The user should see as a result a list with all the proven properties of the model printed on the terminal. In particular, notice the `PCS` and `PCS_conversation` lemmas, which correspond to session-based and conversation-based PCS. In addition, the results include all other helper lemmas and sanity checks on the model. We present in the following a snippet of the expected output, while we provide the full results at the beginning of the `DoubleRatchet.spthy` file in the repository.

```
==================================================
summary of summaries:
analyzed: DoubleRatchet.spthy

...
PCS (all-traces): verified (9 steps)
StartChainPreceededByAssociate (all-traces):
     verified (119 steps)
SameTidKey (all-traces): verified (597 steps)
PCS_Conversation (all-traces): verified (61 steps)
==================================================
```

(E2): [5 human-minutes + 2.4 compute-seconds]

We show the attack on conversation-based PCS. An attacker can perform a step in the protocol using the compromised session state, even though the parties have healed after the compromise. This is possible since the healing happens in another session of the conversation.

Execution: In the shell, the user executes:

```
$ tamarin-prover Sesame/
Sesame_PCSAttack.spthy
```

Note, that the file `./Sesame_PCSAttack.spthy` contains a stored proof of the attack, therefore we verify the proof without the --prove flag.

[Optional] [5 human-minutes + ∼1 compute-minute]

Using docker: To use Tamarin in the interactive mode, the user needs to run docker with an additional parameter that allows access to the IP address outside of docker:

```
$ docker run -p 3001:3001 -it -v "$PWD:/opt/
case-studies" sesameproof/tamarin bash
```

Then, run Tamarin with an extra argument to make the tool listen on any IP address:

```
$ tamarin-prover interactive
Sesame_PCSAttack.spthy  -i='*4'
```

Compiling Tamarin from source: The user needs to run the tool in interactive mode within the directory with the file as follows:

```
$ tamarin-prover interactive
Sesame_PCSAttack.spthy
```

In the end, in both cases the user opens the interface at 127.0.0.1:3001, loads the theory sesame with origin source ./Sesame_PCSAttack.spthy and clicks on the **SOLVED** green keyword of the attack trace.

Results: The user should see as a result a list with the proven attack_pcs lemma printed on the terminal. The expected output is the following:

```
====================================================
summary of summaries:
analyzed: Sesame_PCSAttack.spthy

...
attack_pcs (exists-trace): verified (80 steps)
====================================================
```

[Optional] In Figure 1, we show the attack trace on conversation-based PCS.

(E3): [5 human-minutes + ∼2 compute-minutes]

We prove the session-based and conversation-based PCS on the Signal with sequential sessions model.

Execution: In the shell from the preparation step, the user executes:

```
$ tamarin-prover --prove Sesame/
Sesame_Solution_RestoredPCS.spthy
```

Results: The user should see as a result a list of all the proven properties of the model printed on the terminal. The proven properties include the session-based PCS and PCS_conversation lemmas, as well as other helper and sanity lemmas. We present in the following a snippet of the expected output, while we provide the full results at the beginning of the Sesame_Solution_RestoredPCS.spthy file in the repository.

```
====================================================
summary of summaries:
analyzed: Sesame_Solution_RestoredPCS.spthy

...
PCS (all-traces): verified (9 steps)
SameRootKeyForTid (all-traces): verified (18 steps)
StartPrev (all-traces): verified (78 steps)
distinct_tid (all-traces): verified (53 steps)
SamePartner (all-traces): verified (44 steps)
PCS_Conversation (all-traces): verified (146 steps)
====================================================
```

(E4): [5 human-minutes + ∼1 compute-minute]

We prove the soundness of the clone-detection mechanism on Sesame with sequential sessions and warning message model. Informally, if an honest party detects a clone, there indeed was an attacker that cloned the honest device.

Execution: In the shell from the preparation step, the user executes:

```
$ tamarin-prover --prove Sesame/
Sesame_CloneDetection.spthy
```

Results: The user should see as a result a list of all the proven properties of the model printed on the terminal. In particular, notice the verification of the main guarantee cd_soundness. We present in the following a snippet of the expected output, while we provide the full results at the beginning of the Sesame_CloneDetection.spthy file in the repository.

```
====================================================
summary of summaries:
analyzed: Sesame_CloneDetection.spthy

...
current_origin (all-traces): verified (429 steps)
CompromiseBeforeStart (all-traces): verified (14
     steps)
cd_soundness (all-traces): verified (115 steps)
====================================================
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.

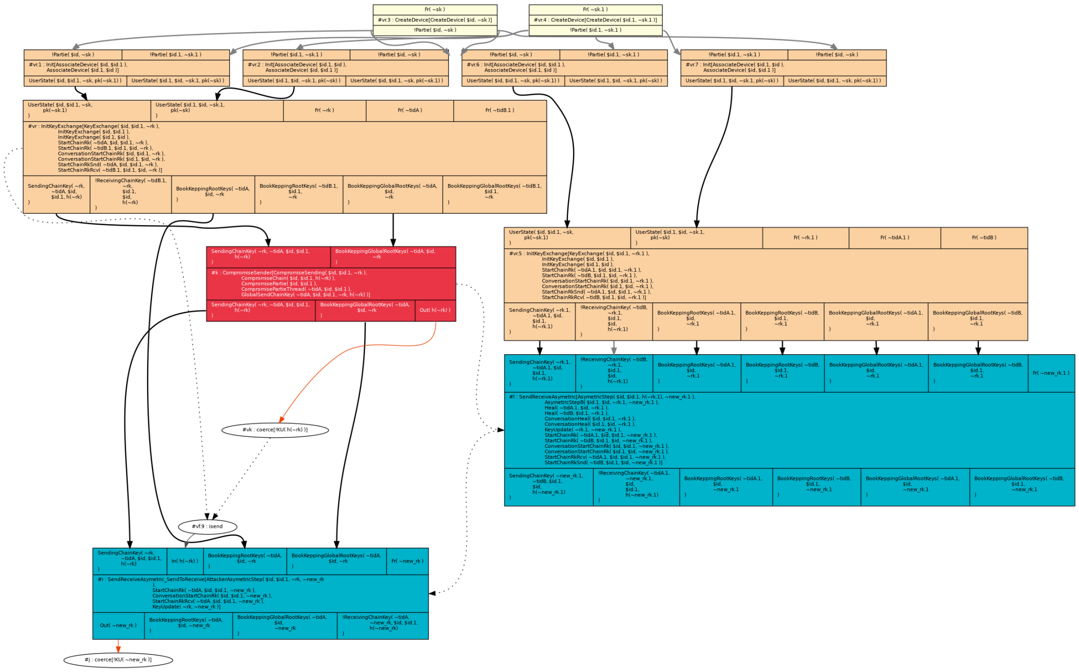
Fr(~sk)
#vr.3 : CreateDevice[CreateDevice($id, ~sk)]
!Partie($id, ~sk)
Fr(~sk.1)
#vr.4 : CreateDevice[CreateDevice($id.1, ~sk.1)]
!Partie($id.1, ~sk.1)
!Partie($id, ~sk)
!Partie($id.1, ~sk.1)
#vr.1 : Init[AssociateDevice($id, $id.1), AssociateDevice($id.1, $id)]
UserState($id, $id.1, ~sk, pk(~sk.1))
UserState($id.1, $id, ~sk.1, pk(~sk))
!Partie($id.1, ~sk.1)
!Partie($id, ~sk)
#vr.2 : Init[AssociateDevice($id.1, $id), AssociateDevice($id, $id.1)]
UserState($id.1, $id, ~sk.1, pk(~sk))
UserState($id, $id.1, ~sk, pk(~sk.1))
!Partie($id, ~sk)
!Partie($id.1, ~sk.1)
#vr.6 : Init[AssociateDevice($id, $id.1), AssociateDevice($id.1, $id)]
UserState($id, $id.1, ~sk, pk(~sk.1))
UserState($id.1, $id, ~sk.1, pk(~sk))
!Partie($id.1, ~sk.1)
!Partie($id, ~sk)
#vr.7 : Init[AssociateDevice($id.1, $id), AssociateDevice($id, $id.1)]
UserState($id.1, $id, ~sk.1, pk(~sk))
UserState($id, $id.1, ~sk, pk(~sk.1))
UserState($id, $id.1, ~sk, pk(~sk.1))
UserState($id.1, $id, ~sk.1, pk(~sk))
Fr(~rk)
Fr(~tidA)
Fr(~tidB.1)
#vr : InitKeyExchange[KeyExchange($id, $id.1, ~rk), InitKeyExchange($id, $id.1), InitKeyExchange($id.1, $id), StartChainRk(~tidA, $id, $id.1, ~rk), StartChainRk(~tidB.1, $id.1, $id, ~rk), ConversationStartChainRk($id, $id.1, ~rk), ConversationStartChainRk($id.1, $id, ~rk), StartChainRkSnd(~tidA, $id, $id.1, ~rk), StartChainRkRcv(~tidB.1, $id.1, $id, ~rk)]
SendingChainKey(~rk, ~tidA, $id, $id.1, h(~rk))
!ReceivingChainKey(~tidB.1, ~rk, $id.1, $id, h(~rk))
BookKeppingRootKeys(~tidA, $id, ~rk)
BookKeppingRootKeys(~tidB.1, $id.1, ~rk)
BookKeppingGlobalRootKeys(~tidA, $id, ~rk)
BookKeppingGlobalRootKeys(~tidB.1, $id.1, ~rk)
SendingChainKey(~rk, ~tidA, $id, $id.1, h(~rk))
BookKeppingGlobalRootKeys(~tidA, $id, ~rk)
#k : CompromiseSender[CompromiseSending($id, $id.1, ~rk), CompromiseChain($id, $id.1, h(~rk)), CompromisePartie($id, $id.1), CompromisePartieThread(~tidA, $id, $id.1), GlobalSendChainKey(~tidA, $id, $id.1, ~rk, h(~rk))]
SendingChainKey(~rk, ~tidA, $id, $id.1, h(~rk))
BookKeppingGlobalRootKeys(~tidA, $id, ~rk)
Out(h(~rk))
UserState($id, $id.1, ~sk, pk(~sk.1))
UserState($id.1, $id, ~sk.1, pk(~sk))
Fr(~rk.1)
Fr(~tidA.1)
Fr(~tidB)
#vr.5 : InitKeyExchange[KeyExchange($id, $id.1, ~rk.1), InitKeyExchange($id, $id.1), InitKeyExchange($id.1, $id), StartChainRk(~tidA.1, $id, $id.1, ~rk.1), StartChainRk(~tidB, $id.1, $id, ~rk.1), ConversationStartChainRk($id, $id.1, ~rk.1), ConversationStartChainRk($id.1, $id, ~rk.1), StartChainRkSnd(~tidA.1, $id, $id.1, ~rk.1), StartChainRkRcv(~tidB, $id.1, $id, ~rk.1)]
SendingChainKey(~rk.1, ~tidA.1, $id, $id.1, h(~rk.1))
!ReceivingChainKey(~tidB, ~rk.1, $id.1, $id, h(~rk.1))
BookKeppingRootKeys(~tidA.1, $id, ~rk.1)
BookKeppingRootKeys(~tidB, $id.1, ~rk.1)
BookKeppingGlobalRootKeys(~tidA.1, $id, ~rk.1)
BookKeppingGlobalRootKeys(~tidB, $id.1, ~rk.1)
SendingChainKey(~rk.1, ~tidA.1, $id, $id.1, h(~rk.1))
!ReceivingChainKey(~tidB, ~rk.1, $id.1, $id, h(~rk.1))
BookKeppingRootKeys(~tidA.1, $id, ~rk.1)
BookKeppingRootKeys(~tidB, $id.1, ~rk.1)
BookKeppingGlobalRootKeys(~tidA.1, $id, ~rk.1)
BookKeppingGlobalRootKeys(~tidB, $id.1, ~rk.1)
Fr(~new_rk.1)
#l : SendReceiveAsymetric[AsymetricStep($id, $id.1, h(~rk.1), ~new_rk.1), AsymetricStepB($id.1, $id, ~rk.1, ~new_rk.1), Heal(~tidA.1, $id, ~rk.1), Heal(~tidB, $id.1, ~rk.1), ConversationHeal($id, $id.1, ~rk.1), ConversationHeal($id.1, $id, ~rk.1), KeyUpdate(~rk.1, ~new_rk.1), StartChainRk(~tidA.1, $id, $id.1, ~new_rk.1), StartChainRk(~tidB, $id.1, $id, ~new_rk.1), ConversationStartChainRk($id, $id.1, ~new_rk.1), ConversationStartChainRk($id.1, $id, ~new_rk.1), StartChainRkRcv(~tidA.1, $id, $id.1, ~new_rk.1), StartChainRkSnd(~tidB, $id.1, $id, ~new_rk.1)]
SendingChainKey(~new_rk.1, ~tidB, $id.1, $id, h(~new_rk.1))
!ReceivingChainKey(~tidA.1, ~new_rk.1, $id, $id.1, h(~new_rk.1))
BookKeppingRootKeys(~tidA.1, $id, ~new_rk.1)
BookKeppingRootKeys(~tidB, $id.1, ~new_rk.1)
BookKeppingGlobalRootKeys(~tidA.1, $id, ~new_rk.1)
BookKeppingGlobalRootKeys(~tidB, $id.1, ~new_rk.1)
#vk : coerce[!KU(h(~rk))]
#vf.9 : isend
SendingChainKey(~rk, ~tidA, $id, $id.1, h(~rk))
In(h(~rk))
BookKeppingRootKeys(~tidA, $id, ~rk)
BookKeppingGlobalRootKeys(~tidA, $id, ~rk)
Fr(~new_rk)
#i : SendReceiveAsymetric_SendToReceive[AttackerAsymetricStep($id, $id.1, ~rk, ~new_rk), StartChainRk(~tidA, $id, $id.1, ~new_rk), ConversationStartChainRk($id, $id.1, ~new_rk), StartChainRkRcv(~tidA, $id, $id.1, ~new_rk), KeyUpdate(~rk, ~new_rk)]
Out(~new_rk)
BookKeppingRootKeys(~tidA, $id, ~new_rk)
BookKeppingGlobalRootKeys(~tidA, $id, ~new_rk)
!ReceivingChainKey(~tidA, ~new_rk, $id, $id.1, h(~new_rk))
#j : coerce[!KU(~new_rk)]

Figure 1: Tamarin output graph of the attack on conversation-based PCS. The attacker can forward the state of a compromised session after the parties have healed in another session.