Anonymity in Network Connections for Mobile Communication

Ragne Elisabeth Henriksen

Master of Science in Physics and Mathematics
Submission date: June 2012
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

I would like to extend a thank you to my supervisor Kristian Gjøsteen. I could not have written this thesis without your help and am grateful that you said yes to be my supervisor. Thank you for helping me understand the theory on which this thesis is built, for having patience and for keeping me motivated.

Abstract

This thesis summarizes an existing protocol, that we have chosen to call the *Token Key Agreement* protocol. It then goes on to introduce two new protocols we have chosen to name the *Symmetric Key Agreement* protocol and the *Asymmetric Key Agreement* protocol. We are working within the UC framework, and as such introduce ideal functionalities and protocol descriptions for the protocols. For the first protocol we also introduce a simulated adversary. Further, the paper includes an overview of the security offered by the three protocols.

Sammendrag

Denne oppgaven oppsummerer en eksisterende protokoll som vi har valgt å kalle *Token Nøkkelutvekslings* protokollen. Den fortsetter deretter med å introdusere to nye protokoller vi har valgt å kalle *Symmetrisk Nøkkelutvekslings* protokollen og *Asymmetrisk Nøkkelutvekslings* protokollen. Vi jobber innenfor UC rammeverket, og utleder således ideelle funksjonaliteter for de to protokollene, protokollbeskrivelser og for den første protokollen konstruerer vi også en simulert angriper. Videre inneholder oppgaven en oversikt over sikkerheten som tilbys av de tre protokollene.

Contents

Chapter 1

Introduction

The word anonymity is derived from the Greek word anonymia meaning “nameless”. Anonymity in its most general form means that a persons identity remains unknown. In today’s society obtaining anonymity in the digital world has become a widely discussed problem. The concept of anonymity comes into play in those cases in which we want to keep secret the identity of the participants of a certain event. There are several situations in which this property may be needed or desirable; for instance: voting, web surfing, anonymous donations, and posting on bulletin boards. This paper will talk about how to obtain anonymity in network connections in mobile communication.

The paper will begin by introducing the model we will be working within. It then goes on to introduce a protocol called the *Symmetric Key Agreement* protocol in Chapter 3. This protocol is the main work of the paper. The chapter starts by providing a full protocol description in Section 3.1. It then goes on to provide a simulator for the protocol in Section 3.2. Lastly, the chapter includes a security analysis of the protocol.

Chapter 4 introduces a protocol we have named the *Asymetric Key Agreement* protocol. The chapter has the same buildup as the previous one. However, we do not introduce a simulator for this protocol.

Lastly, in Chapter 5 we give a summarized version of the protocol suggested in the article *Secure and Anonymous Network Connection in Mobile Communications* by Gjøsteen, Petrides, and Steine [GPS12]. This is the protocol that the first two protocols are built on.

The three protocols will be shown to have varying security properties and efficiencies and the paper concludes with a discussion of the advantages and disadvantages of the three protocols.

1.1 Prerequisites

The reader is assumed to be familiar with the Universal Composability (UC) framework. Due to space requirements information on the UC framework is not included in this paper, but can be found in [Can00]. We will not be using Cannettis framework with dummy variables when describing our protocols, but rather a similar framework that can be found in the article, *A Novel Framework for Protocol Analysis* by Gjøsteen, Petrides and Steine [GPS11]. This framework is an adaption of Cannetti's framework. The main difference between the two frameworks is that whilst Canetti uses a dynamic setting, where ITMs come into existence only when they are needed, the novel framework considers a fixed number of ITMs where communication is regulated by a fixed communication graph. The new framework uses a message queue to activate ITMs. This lets every ITM submit several messages into the message queue for later activation. Thus, the new framework lets ITM's send messages to themselves, and it allows them to reactivate themselves when all the previously queued messages have been processed. In Cannetti's framework self activation is not possible. Using a fixed system of ITMs also aids when depicting protocol machines handling multiple sessions.

1.2 Abbreviations and Terminology

We will use IMSI to mean International Mobile Subscriber Identity and TMSI to mean Temporary Mobile Subscriber Identity. The IMSI is integrated on the phone's simcard whilst the TMSI is generated by the MNO. Further, we will abbreviate the Token Key Agreement protocol with TKA, the Symmetric Key Agreement protocol with SKA and the Asymmetric Key Agreement protocol with AKA.

Chapter 2

The Model

We will now give a brief description of the model we will be working within. Although this is not common practice we will assume that when utilizing a cellphone, a user, U communicates with a mobile network operator, MNO, which in turn communicates with U's service provider, SP, as shown in Figure 2.1. U and SP only ever communicate with each other through the MNO. This is a model where MNOs deal only with maintaining the network infrastructure and SPs deal with user subscriptions for network access.

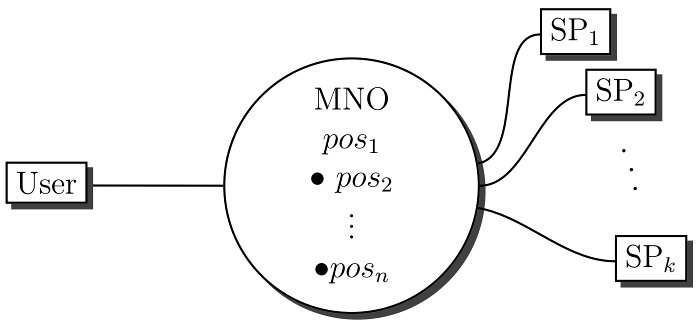

Figure 2.1: A graphical depiction of a mobile communication network with n base stations and k service providers.

2.1 The Radio Link Functionality and the Secure Channel Functionality

Users and the mobile network operator communicate through an insecure channel. We have modeled this channel with a functionality called $\mathcal{F}_{RL}$, for radio link. The functionality can be found in Figure 2.2. Service providers and the mobile network operator on the other hand, communicate through a secure channel as modeled by the functionality $\mathcal{F}_{sec}$, for secure. This functionality can be found in Figure 2.3. Both of these functionalities are taken from a draft of the paper *Secure and Anonymous Network Connection in Mobile Communications* from January 11th of 2012.

```
On ($\mathbf{Listen}$, $pos$) from $\mathcal{A}$
    1. Record $pos$ as corrupted and hand over ($\mathbf{Listening}$, $pos$) to $\mathcal{A}$.
On ($\mathbf{Enter}$, $pos$) from $U$
    1. Record $U$ as present at $pos$.
On ($\mathbf{Leave}$, $pos$) from $U$
    1. Remove the record of $U$'s presence at $pos$.
On ($\mathbf{Send}$, $sid$, $m$, $pos$, $N$) from $U$
    1. Stop if $U$ is not present at $pos$.
    2. If $pos$ is corrupted then hand over ($\mathbf{Send}$, $sid$, $m$, $pos$, $N$) to $\mathcal{A}$.
    3. Else send ($\mathbf{Recv}$, $sid$, $m$, $pos$) to $N$.
On ($\mathbf{Send}$, $sid$, $m$, $pos$) from $N$
    1. If $pos$ is corrupted then hand over ($\mathbf{Send}$, $sid$, $m$, $N$ $pos$) to $\mathcal{A}$.
    2. Else send ($\mathbf{Recv}$, $sid$, $m$, $pos$) to $U$, for all honest users $U$ present
       at $pos$.
On ($\mathbf{Send}$, $sid$, $m$, $pos$, $N$) from $\mathcal{A}$
    1. Stop if $pos$ is not corrupted, otherwise send ($\mathbf{Recv}$, $sid$, $m$, $pos$)
       to $N$.
On ($\mathbf{Send}$, $sid$, $m$, $pos$) from $\mathcal{A}$
    1. Stop if $pos$ is not corrupted, otherwise send ($\mathbf{Recv}$, $sid$, $m$, $pos$)
       to $U$, for all honest users $U$ present at $pos$.
```

Figure 2.2: The radio link functionality $\mathcal{F}_{RL}$.

```
On (**Send**, $sid$, $m$, $X_1$, $X_2$) from $\mathcal{A}$
    1. Stop if $X_1$ is honest or $X_2$ is corrupted, otherwise send
       (**Recv**, $sid$, $m$, $X_1$) to $X_2$.
On (**Send**, $sid$, $m$, $X_1$) from honest $X_2$
    1. If $X_1$ is honest then send (**Recv**, $sid$, $m$, $X_2$) to $X_1$.
    2. Else hand over (**Send**, $sid$, $m$, $X_2$, $X_1$) to $\mathcal{A}$.
```

Figure 2.3: The secure channel functionality $\mathcal{F}_{sec}$.

2.2 The Corruption Classes

Since the radio link functionality is not secure an adversary $\mathcal{A}$ can listen in on the channel at a given position. We have modeled this by letting the position, *pos*, be corrupt whenever someone is listening or intercepting the radio link channel. An execution of a protocol may start without anyone listening at *pos* only to have *pos* become corrupted during the execution. This can easily be modeled through the simulators for the three protocols, but should be kept in mind when reading the security analysis for the different protocols.

Further, U, N and S may all be corrupt. Whenever either N or U is corrupt *pos* will be corrupt as well. We have restricted ourselves to be operating with static corruption of players to simplify the model. We have included a table of the 10 corruption classes in Figure 2.4. One can easily check that these are in fact all the corruption classes.

C.C.	*pos*	U	N	S
0	Honest	Honest	Honest	Honest
1	Corrupt	Honest	Honest	Honest
2	Corrupt	Honest	Corrupt	Honest
3	Honest	Honest	Honest	Corrupt
4	Corrupt	Honest	Honest	Corrupt
5	Corrupt	Honest	Corrupt	Corrupt
6	Corrupt	Corrupt	Honest	Honest
7	Corrupt	Corrupt	Corrupt	Honest
8	Corrupt	Corrupt	Honest	Corrupt
9	Corrupt	Corrupt	Corrupt	Corrupt

Figure 2.4: The different corruption classes (C.C.).

2.3 The Ideal Functionality

In the chapters to come we will introduce three protocols whose goal it is to achieve secure and anonymous mobile communication. These three protocols can a be realized by the same ideal functionality, namely $\mathcal{F}_{KE}$. Two of the three protocols make use of tokens. In these two protocols every user has a token that he has been given by his service provider. He uses this token to make himself known to the service provider. If a session fails before the user receives a new token he will have to reuse the token he has the next time he tries to start a session. An adversary may then try to link different attempts at starting a session in order to trace a user. We have modeled this by allowing the simulators for these two protocols to send (**Deny Linkable**, *id*) (or (**Deny Unlinkable**, *id*) in the cases where the token is updated but the protocol aborts) to the ideal functionality. The third protocol does not make use of tokens, and when an execution fails the simulator will send (**Deny**, *id*) to the ideal functionality.

We wish for the three ideal functionalities to be equal. This way it will be easy to build on top of them if desirable. As such it will be possible to send (**Deny Linkable**, *id*), (**Deny Unlinkable**, *id*) and (**Deny**, *id*) to the ideal functionality in all of the protocols. However, the simulators specify what happens for each protocol, and although it is possible for the ideal functionalities to receive all three the simulators will only send either the two first, or the third message to the ideal functionality.

A protocol securely executes a given task if an adversary $\mathcal{A}$ can not gain anything more from an attack on a real execution of the protocol than from an attack on an ideal process where the parties simply hand their inputs to a trusted party, $\mathcal{F}_{KE}$, with the appropriate functionality and obtain their outputs from it without any further interaction [CLOS02]. Thus, it is not problematic to use the same ideal functionality for all three protocols. The ideal functionality is described in Figures 2.5 and 2.6.

Further, the ideal functionality makes use of a function called the *leak* function. This function leaks information to the adversary as specified by the ideal functionality. The information it leaks depends on what corruption class we are in. The leak function will be the same for all three protocols. Thus, we could have made it a part of the ideal functionality. However, so that it will be possible to build other protocols on top of the same functionality we have avoided doing this. The leak function is described in Figure 2.7.

In our model a user, U tells the functionality $\mathcal{F}_{KE}$ that he wishes to establish a key with the MNO at a given position. $\mathcal{A}$ is informed about the attempt. $\mathcal{A}$ can the decide what happens. In all corruption classes besides C.C.0 $\mathcal{A}$ is free to perform denial of service (DoS) attacks. Thus, the adversary may allow the protocol to complete, to partially complete or he may interfere in other ways. An adversary is hence also allowed to change the order messages are received in for some of the corruption classes. What $\mathcal{A}$ is able to do depends on what corruption class we are in, and how the users previous attempt at key agreement ended. Every attempted protocol execution is identified using a session identifier (sid). It should also be noted that U will never run 2 instances of the protocol in parallel.

In the two protocols where we make use of tokens, tokens will be uncorrelated. However, and adversary may through DoS attacks or by interfering prevent a user from receiving his new token. This would allow both the network and an adversary to trace a user. However, the adversary would be tracing an anonymous user. Further, the only way a service provider could find a particular users location if if he is listening at pos.

2.4 Authenticated Encryption with Associated Data

In the two protocols we construct we use authenticated encryption with associated data (AEAD). This is a method for authenticating data where some of the data, *adata*, is authenticated whilst some of the data, *cdata*, is both encrypted and authenticated. We write that

- $c \leftarrow \mathcal{E}(k; adata; cdata)$.
- $cdata/\perp \leftarrow \mathcal{D}(k; adata; c)$.

The scheme was formalized by Phillip Rogaway in 2002 in the article *Authenticated-Encryption with Associated-Data* [Rog02].

On (**Enter**, pos, N) from honest U
1. Stop if a record (id, pos, U, N, S) exists. Else if no record (U, id) exists, generate random identifier id, in either case record (id, pos, U, N, S) where S is U's SP and send (**Enter**, **Leak**($class$, id, pos, U, N, S, k')) to $\mathcal{S}$

On (**Leave**, pos) from U
1. Stop if (id, pos, U, N, S) is not recorded, otherwise hand over (**Leave**, id) to $\mathcal{S}$ and erase the record.

On (**Deny linkable**, id) **or** (**Deny unlinkable**, id) from $\mathcal{S}$
1. Stop if (id, pos, U, N, S) is not recorded, otherwise remove it and send (**Est**. **failed linkable**) **or** (**Est**. **failed unlinkable**) to U. Additionally, if S is honest then record (U, id) **or** remove any such record.

On (**Deny**, id) from $\mathcal{S}$
1. Stop if (id, pos, U, N, S) is not recorded, otherwise remove it and send (**Est**. **failed**) to U.

On (**Listen**, pos) from $\mathcal{S}$
1. From every record containing pos that is in C.C.0 **or** 3, collect (id, N, S) **or** (id, U, N), record pos as corrupted and hand over the collected data together with pos and **Leak**($class$, id, pos, U, N, S, k') to $\mathcal{A}$.

On (**Est**. **User**, k, id) from $\mathcal{S}$
1. Stop if (id, pos, U, N, S) is not recorded, otherwise:
   - (i) If in C.C.0, 3 or 4 then generate random key $\hat{k}$, replace the record by (id, pos, U, N, S, $\hat{k}$) and send (**Est**., $\hat{k}$, pos, N) to U
   - (ii) Else if in C.C.2 **or** 5 then send (**Est**., k, pos, N) to U and either replace the record by (id, U, N, S) **or** remove it.

On (**Est**. **SP**, id) from $\mathcal{S}$
1. Stop if neither (id, U, N, S) **nor** (id, pos, U, N, S, k) is not recorded. Otherwise, either remove the record **or** replace it by (id, pos, N, S, k), and send (**Est**., U, N) to S.

On (**Est**. **SP**, id, U, N, S) from $\mathcal{S}$
1. Stop if not in C.C.6 or 7. Otherwise, in the first case record (id, N, S) and in both cases send (**Est**., U, N) to S.

Figure 2.5: The ideal functionality $\mathcal{F}_{KE}$.

On (**Est**. **MNO**, k, id) from $\mathcal{S}$

1. Stop if (id, pos, N, S, $\hat{k}$) is not recorded, otherwise remove it and send (**Est**., $\hat{k}$, pos, S) to N.

On (**Est**. **MNO**, k, id, pos) **<u>or</u>** (**Est**. **MNO**, k, pos, N, S) from $\mathcal{S}$

1. Stop unless (id, N, S) is recorded **<u>or</u>** pos and S are corrupted. In the first case remove the record and in both cases send (**Est**., k, pos, S) to N.

Figure 2.6: The ideal functionality $\mathcal{F}_{KE}$ continued.

$$\mathbf{leak}(class, id, pos, U, N, S) = \begin{cases} (id) & \text{if } class = 0 \\ (id, pos, S) & \text{if } class = 1 \\ (id, pos, N, S) & \text{if } class = 2 \\ (id, U, N, S) & \text{if } class = 3 \\ (id, pos, U, N, S) & \text{if } class = 4,\ 5,\ 6,\ 7,\ 8 \text{ or } 9 \end{cases}$$

Figure 2.7: The leakage function for TKA, SKA and AKA.

Chapter 3

The Symmetric Key Agreement Protocol

The symmetric key agreement protocol is the main work of this thesis. The protocol makes use of tokens and a symmetric encryption function. The user, U, and the service provider, S, share a secret key k_{US}. This chapter starts of by describing the protocol in Section 3.1. It then provides an ideal-process adversary, namely the simulator $\mathcal{S}_{SKA}$. We wish to show that there exists a $\mathcal{S}_{SKA}$ such that the environment $\mathcal{Z}$ can not tell whether it is interacting with $\mathcal{F}_{KE}$ and $\mathcal{S}_{SKA}$ in the ideal process, or with π_{SKA} in the real-life model with non-negligible probability. This would normally be done through a series of games. However, we have chosen to show that the protocol π_{SKA} only has n possible executions. We then show that we can simulate all n executions through $\mathcal{S}_{SKA}$, thus concluding that the protocol realizes the ideal functionality. This is done in Section 3.3 and concludes the chapter.

3.1 The Protocol

An overview of the protocol is provided in Figure 3.1. A full protocol description then follows in Figures 3.2, 3.3 and 3.4. We would like to note $e = (sid, pos, n_1, n_2, k, T, N, S, c, \mu_1)$, where $\mu_1 = \mathcal{E}(k; pos, c;)$. We have named the variable e for everything, as the variables contained in e are everything that both U and N knows.

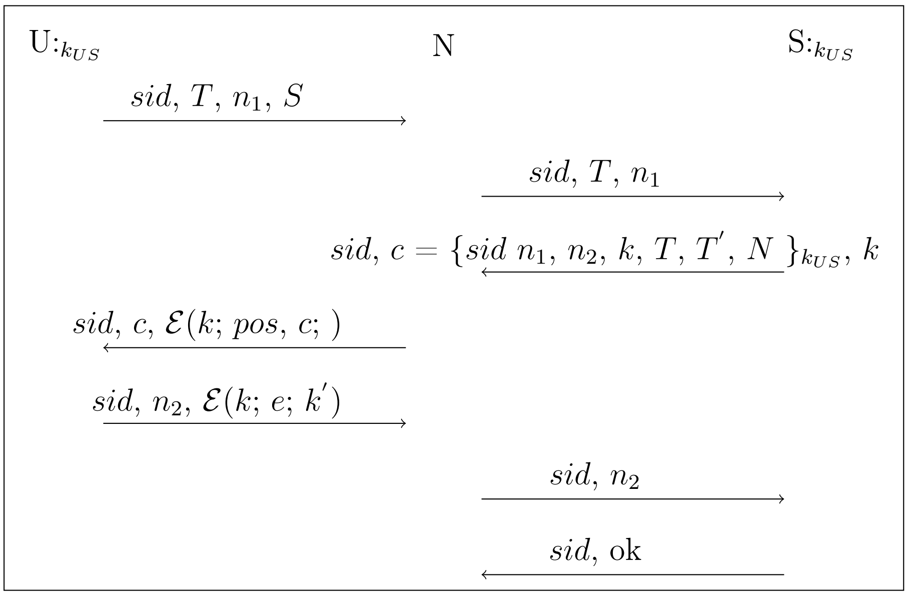
U:k_{US}
N
S:k_{US}
sid, T, n_1, S
sid, T, n_1
sid, $c = \{sid\ n_1, n_2, k, T, T^{'}, N\}_{k_{US}}$, k
sid, c, $\mathcal{E}(k; pos, c;)$
sid, n_2, $\mathcal{E}(k; e; k^{'})$
sid, n_2
sid, ok

Figure 3.1: Summary of the anonymous symmetric key establishment protocol π_{SKA}. Communication between users and MNOs is via $\mathcal{F}_{RL}$ and between MNOs and SPs via $\mathcal{F}_{sec}$.

```
**Stored**: $S$ , $k_{US}$, $T$ = Token, $N$, $k^{'}$
On (**Enter**, $pos$, $N$) from $\mathcal{Z}$:
    1. If nothing is recorded then generate random nonce $n_1$ and session
       identifier $sid$, record ($sid$, $pos$, $N$, $T$, $n_1$) and send (**Enter**, $pos$)
       and (**Send**, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$, $N$) to $\mathcal{F}_{RL}$.
On (**Leave**, $pos$) from $\mathcal{Z}$:
    1. Send (**Leave**, $pos$) to $\mathcal{F}_{RL}$.
    2. If a record with $pos$ exists remove it and output
       (**Est**. **failed unlinkable**) if token was updated or
       (**Est**. **failed linkable**) otherwise.
On (**Recv**, $sid$, $m$, $pos$) from $\mathcal{F}_{RL}$:
    1. If $m = (sid, c, \mu_1)$ where $\mathbf{Dec}_{k_{US}}(c) = (sid, n_1, n_2, k, T, T^{'}, N)$
       and ($sid$, $pos$, $N$, $T$, $n_1$) is recorded, update $T \leftarrow T^{'}$. If in addition
       $\mu_1 = \mathcal{E}(k; pos, c; )$ replace the record with
       ($sid$, $pos$, $N$, $T$, $n_1$, $n_2$, $c$, $\mu_1$), set $\mu_2 \leftarrow \mathcal{E}(k; e; k^{'})$,
       output (**Est**., $k^{'}$, $pos$, $N$) and send
       (**Send**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$, $N$) to $\mathcal{F}_{RL}$.
    2. Else do as second point of (**Leave**, $pos$) above.
```

Figure 3.2: The anonymous key establishment protocol π_{SKA}, Part 1: User.

```
Stored:
On (Recv, $sid$, $m$, $pos$) from $\mathcal{F}_{RL}$:
    1. If $m = (sid, T, n_1, S)$ and $sid$ is not on any record, record
       $(sid, T, n_1, pos, S)$ and send (Send, $sid$, $(sid, T, n_1)$, $S$) to $\mathcal{F}_{sec}$.
    2. Else if $m = (sid, n_2, \mu_2)$ and $(sid, T, n_1, pos, S, c, \mu_1, k)$ is
       recorded and $\mathcal{D}(k; e; \mu_2) \neq \perp$, set
       $k' \leftarrow \mathcal{D}(k; e; \mu_2)$, replace the previous record by
       $(sid, T, n_1, pos, S, n_2, c, \mu_1, \mu_2, k, k')$ and send
       (Send, $sid$, $(sid, n_2)$, $S$) to $\mathcal{F}_{sec}$.
On (Recv, $sid$, $m$, $pos$) from $\mathcal{F}_{sec}$:
    1. If $m = (sid, c, k)$ and there is a record $(sid, T, n_1, pos, S)$, set
       $\mu_1 \leftarrow \mathcal{E}(k; pos; c)$, change the record to
       $(sid, T, n_1, pos, S, c, \mu_1, k)$ and send (Send, $sid$, $(sid, c, \mu_1)$, $S$)
       to $\mathcal{F}_{RL}$.
    2. Else if $m = (sid, \text{ok})$ and there is a record
       $(sid, T, n_1, pos, S, n_2, c, \mu_1, \mu_2, k, k')$ then remove it and output
       (Est., $k'$, $pos$, $S$).
```

Figure 3.3: The anonymous key establishment protocol π_{SKA}, Part 2: MNO.

```
Stored: $k_T$, $\{U; k_{US}\}$
On (Recv, $sid$, $m$, $N$) from $\mathcal{F}_{sec}$:
    1. If $m = (sid, T, n_1)$ and $sid$ is not on any record, record
       $(sid, T, n_1, N)$ and $\mathbf{Dec}_{k_T}(T) = U$ then generate random nonce $n_2$
       and token $T_1 = \mathbf{Enc}_{k_T}(U)$, set $c \leftarrow \mathbf{Enc}_{k_{US}}(sid, n_1, n_2, k, T, T_1, N)$,
       record $(sid, T, n_1, n_2, U, N)$ and send (Send, $sid$, $(sid, c, k)$, $N$)
       to $\mathcal{F}_{sec}$.
    2. Else if $m = (sid, n_2)$ and $(sid, T, n_1, n_2, U, N)$ is recorded then
       remove it, send (Send, $sid$ $(sid, \text{ok})$, $N$) to $\mathcal{F}_{sec}$ and output
       (Est., $U$, $N$).
```

Figure 3.4: The anonymous key establishment protocol π_{SKA}, Part 3: SP.

3.2 The Simulator

This simulator is not based on the one that exists for the TAP protocol [GPS12]. Instead we have described what happens in each corruption class, and have modeled the possible transitions between corruption classes towards the end of the simulator. We will talk about the corruption classes as being classes. For instance, we will denote something that is in C.C.0 as being in Class 0.

Within the simulator we will use $\mathcal{F}_{RL}$ and $\mathcal{F}_{sec}$ to describe simulated functionalities that are the same as $\mathcal{F}_{RL}$ and $\mathcal{F}_{sec}$ described in Figure 2.2 and Figure 2.3.

Further, we will assume that a ciphertext is on the same form as random noise. We know that $\varepsilon_{k_{US}}(0^{l_i}) \sim \varepsilon_{k_{US}}(m_i)$ where $|m_i| = l_i$ and $\varepsilon_{k_{US}}(m_i)$ means the encryption of message m_i using key k_{US}. Thus, it would not be difficult producing noise that looks like an encryption. For simplicity we will in the simulator $\mathcal{S}_{SKA}$ use the term *generate random string X of ciphertext* to describe this, thus assuming that $\{0, 1\}^{l_i}$ $\sim \varepsilon_{k_{US}}(m_i)$.

The simulator contains two storage tapes. The first one is called **Sessions** and it keeps track of every session or attempted session. The next one is called **Corrupt** and keeps track of which positions are corrupt.

In every new command of the simulator it should say, verify that there exists a record containing *sid* on **Sessions**, else ignore. To save space we have omitted this, but it should be noted that this is an implicit requirement.

The simulator follows on the next pages. We have included figures that summarize the executions in the different classes. This should make it easier to understand the simulator and visualize what happens. We have used i as a counter, and i_L and i_U to keep track of when the protocol fails linkable and unlinkable.

Lastly, we would like to mention that when we write $\mathcal{A}/\mathcal{F}$ we mean '$\mathcal{A}$ through $\mathcal{F}$.

The simulator $\mathcal{S}_{SKA}$

```
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 0$:
    **On** (ENTER, $id$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, -, -, -), where
         $i = 0$, and send ($i$, $sid$) to self.

    **On** ($i$, $sid$) from self:
      1. If $i = 9$
         Generate random string of ciphertext $k^{'}$, update
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, -, -, -, $S$, -, -, -, -, $k^{'}$, -, -, -, -)
         and send (**Est**. **User**, $k^{'}$, $id$) to $\mathcal{F}_{KE}$.
      2. Else if $i = 14$
         Send (**Est**. **SP**, $id$) to $\mathcal{F}_{KE}$.
      3. Else if $i = 17$
         Send (**Est**. **MNO**, $k$, $id$) to $\mathcal{F}_{KE}$ and stop.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    **On** any other input, ignore.
```

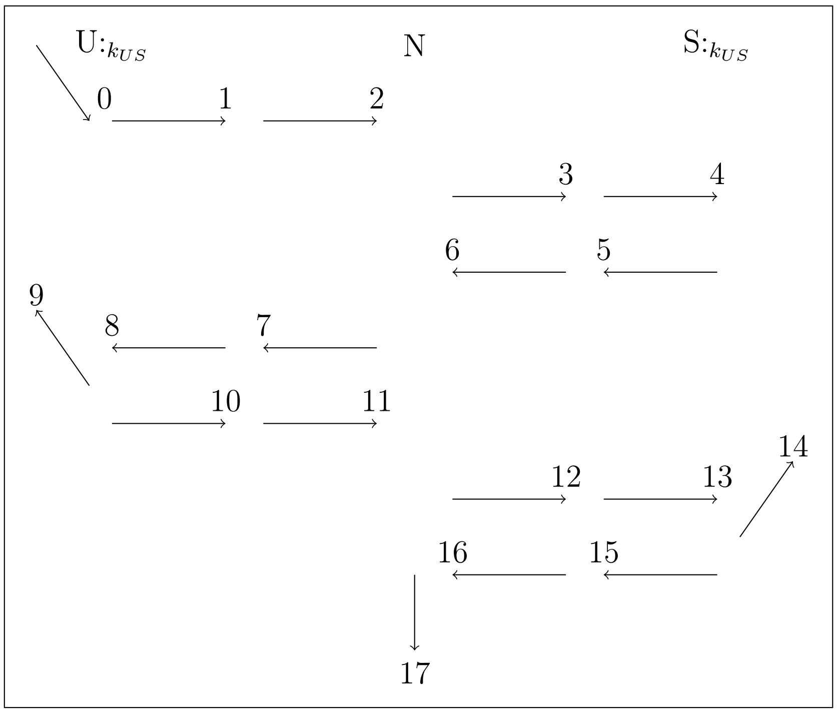

Figure 3.5: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 0.

```
The simulator $\mathcal{S}_{SKA}$ continued
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 1$:
    **On** (ENTER, $id$, $pos$, $S$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, $pos$, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on **Sessions** and send ($i$, $sid$) to self.

    **On** ($i$, $sid$) from self:
      1. If $i = 1$
         Generate random $T$ of Token length, and nonce $n_1$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$, $N$) to $\mathcal{F}_{RL}$ and
         stop.
      3. Else if $i = 7$
         Generate random strings $c$ and $\mu_1$ of cipher text, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, -, $\mu_1$, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $c$, $\mu_1$), $pos$) to $\mathcal{F}_{RL}$ and stop.
      4. Else if $i = 9$
         (i)   If $i_L = 9$ send (**Deny Linkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (ii)  Else if $i_U = 9$ send (**Deny Unlinkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (iii) Else generate random string $k'$ of ciphertext, store
               ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, -, $\mu_1$, $k'$, -, -, $i_U$, $i_L$)
               on **Session** and send (**Est**. **User**, $k'$, $id$) to $\mathcal{F}_{KE}$.
      5. Else if $i = 10$
         Generate random nonce $n_2$ and string of ciphertext $\mu_2$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, $k$, $\mu_1$, $k'$, $\mu_2$, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$, $N$) to $\mathcal{F}_{RL}$ and stop.
      6. Else if $i = 14$
         Send (**Est**. **SP**, $id$) to $\mathcal{F}_{KE}$.
      7. Else if $i = 17$
         Send (**Est**. **MNO**, $k'$, $id$) to $\mathcal{F}_{KE}$ and stop.
      8. Set $i \leftarrow i + 1$.
      9. Send ($i$, $sid$) to self.
```

```
The simulator $\mathcal{S}_{SKA}$ continued
While $class = 1$:
    On (Recv, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Check that $i = 2$, else ignore.
      2. Check that $T$ is a valid token, else ignore.
      3. If $T$ is a valid token, but $T$, $n_1$ or $S$ do not match those on $sid$'s
         record on Sessions update $i_L \leftarrow 9$.
      4. Else if no record containing $sid$ exists, store
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on Sessions.
      5. Set $i \leftarrow i + 1$.
      6. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $c$, $\mu_1$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Check that $i = 8$, else ignore.
      2. If a record
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, $c$, -, $\mu_1$, -, -, -, $i_U$, $i_L$)
         exists (i.e. there is no value for $id$), ignore.
      3. Check that $c$ matches that on $sid$'s record on Sessions,
         else set $i_L \leftarrow 9$.
      4. Check that $\mu_1$ matches that on $sid$'s record on Sessions,
         else set $i_U \leftarrow 9$.
      5. Set $i \leftarrow i + 1$.
      6. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Check that $i = 11$, else ignore.
      2. Check that $\mu_2$ matches that on $sid$'s record on Sessions,
         else ignore.
      3. Check that $n_2$ matches that on $sid$'s record on Sessions,
         else ignore.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    On any other input, ignore.
```

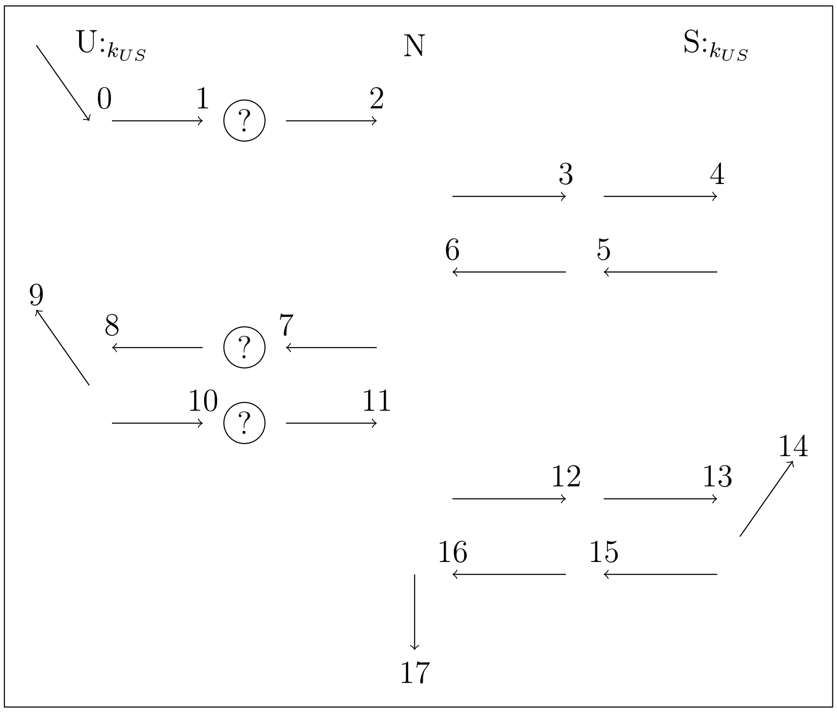

Figure 3.6: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 1.

The simulator $\mathcal{S}_{SKA}$ continued

```
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 2$:
    **On** (ENTER, $id$, $pos$, $N$, $S$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, $pos$, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on **Sessions** and send ($i$, $sid$) to self.

    **On** ($i$, $sid$) from self:
      1. If $i = 1$
         Generate random $T$ of Token length, and nonce $n_1$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$, $N$) to $\mathcal{F}_{RL}$ and
         stop.
      2. Else if $i = 3$
         Generate random string $c$ of cipher text and key $k$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, $k$, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $c$, $k$), $N$) to $\mathcal{F}_{sec}$ and stop.
      3. Else if $i = 5$
         (i)   If $i_L = 5$ send (**Deny Linkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (ii)  If $i_U = 5$ send (**Deny Unlinkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (iii) Else generate random key $k'$, store
               ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, $k$, $\mu_1$, $k'$, -, -, $i_U$, $i_L$)
               on **Sessions** and send (**Est**. **User**, $k'$, $id$) to $\mathcal{F}_{KE}$.
      4. Else if $i = 6$
         Let $\mu_2 \leftarrow \mathcal{E}(k; e; k')$ and generate nonce $n_2$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, $k$, $\mu_1$, $k'$, $\mu_2$, $n_2$, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$, $N$) to $\mathcal{F}_{RL}$ and stop.
      5. Else if $i = 8$
         Send (**Est**. **SP**, $id$) to $\mathcal{F}_{KE}$.
      6. Else if $i = 9$
         Send (**Send**, $sid$, ($sid$, ok), $N$) to $\mathcal{F}_{sec}$ and stop.
      7. Set $i \leftarrow i + 1$.
      8. Send ($i$, $sid$) to self.
```

```
The simulator $\mathcal{S}_{SKA}$ continued
While $class = 2$:
    On (Recv, $sid$ ($sid$, $T$, $n_1$), $N$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 2$ else ignore.
      2. Check that $T$ is a valid token, else ignore.
      3. If $T$ is a valid token, but $T$ or $n_1$ do not match those on $sid$'s
         record on Sessions update $i_L \leftarrow 5$.
      4. Else if no record containing $sid$ exists, store
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on Sessions.
      5. Set $i \leftarrow i + 1$.
      6. Send ($i$, $sid$) to self.

    On (Recv, $sid$ ($sid$, $c$, $\mu_1$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Check that $i = 4$ else ignore.
      2. If a record
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, $c$, -, $\mu_1$, -, -, -, $i_U$, $i_L$)
         exists (i.e. there is no value for $id$), ignore.
      3. Else if $c$ does not match that on $sid$'s record on Sessions set
         $i_L \leftarrow 5$.
      4. If $\mu_1 \neq \mathcal{E}(k; pos, c; )$ set $i_U \leftarrow 5$.
      5. Store ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, $c$, $k$, $\mu_1$, -, -, -, $i_U$, $i_L$)
         on Sessions.
      6. Set $i \leftarrow i + 1$.
      7. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $n_2$), $N$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 7$, else ignore.
      2. Check that $n_2$ matches that on $sid$'s record on Sessions, else
         ignore.
      3. Set $i \leftarrow i + 1$.
      4. Send ($i$, $sid$) to self.

    On any other input, ignore.
```

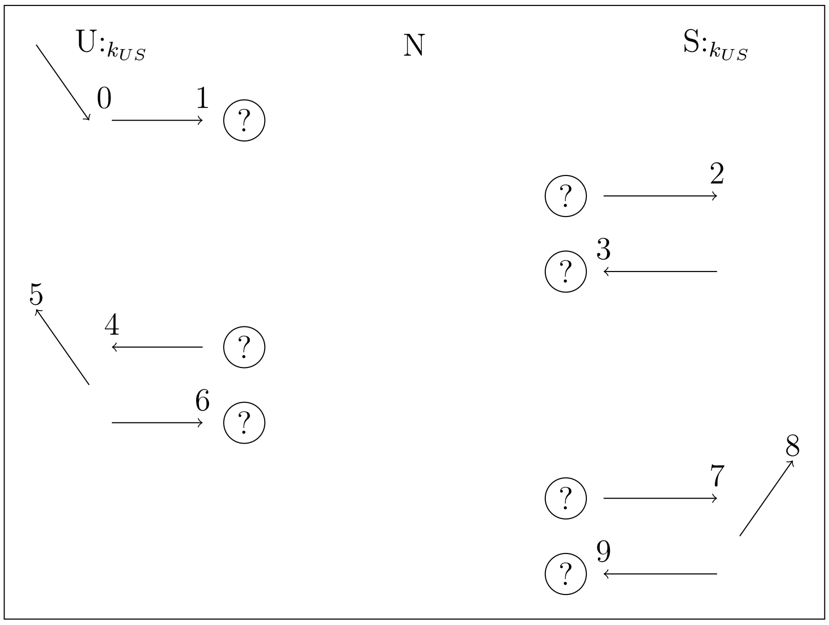

Figure 3.7: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 2.

```
The simulator $\mathcal{S}_{SKA}$ continued
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 3$:
    **On** (ENTER, $id$, $U$, $N$, $S$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on **Sessions** and send ($i$, $sid$) to self.

    **On** ($i$, $sid$) from self:
      1. If $i = 3$
         Generate random $T$ of Token length, and nonce $n_1$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $T$, $n_1$), $S$) to $\mathcal{F}_{sec}$ and stop.
      2. Else if $i = 7$
         (i)  If $i_L = 7$ send (**Deny Linkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (ii) Else generate random string of ciphertext $k'$, store $k'$ on
              **Sessions** and send (**Est**. **User**, $k'$, $id$) to $\mathcal{F}_{KE}$.
      3. Else if $i = 10$
         Send (**Send**, $sid$, ($sid$, $n_2$), $S$) to $\mathcal{F}_{sec}$ and stop.
      4. Else if $i = 12$
         Send (**Est**. **MNO**, $k'$, $id$) to $\mathcal{F}_{KE}$ and stop.
      5. Set $i \leftarrow i + 1$.
      6. Send ($i$, $sid$) to self.

    **On** (**Recv**, $sid$, ($sid$, $c$, $k$), $S$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 4$, else ignore.
      2. Verify that $dec_{k_{US}}(c) = (sid, n_1, n_2, k, T, T', N)$, and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, else set $i_L \leftarrow 7$.
      3. Set $i \leftarrow i + 1$.
      4. Send ($i$, $sid$) to self.
```

```
The simulator $\mathcal{S}_{SKA}$ continued
While $class = 3$:
    On (Recv, $sid$, ($sid$, ok), $N$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 11$, else ignore.
      2. Check that a record
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, -, $k'$, -, $n_2$, $i_U$, $i_L$)
         exists, else ignore.
      3. Set $i \leftarrow i + 1$.
      4. Send ($i$, $sid$) to self.

    On any other input, ignore.
```

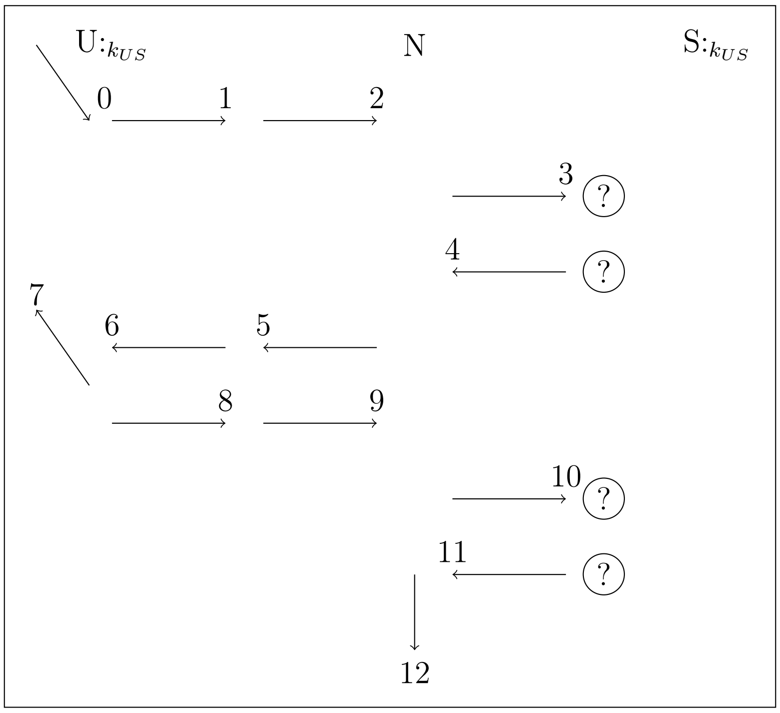

Figure 3.8: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 3.

The simulator $\mathcal{S}_{SKA}$ continued

```
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 4$:
    **On** (ENTER, $id$, $pos$, $U$, $N$, $S$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on **Sessions** and send ($i$, $sid$) to self.

    **On** ($i$, $sid$) from self:
      1. If $i = 1$
         Generate random $T$ of token length and nonce $n_1$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$) on
         **Sessions**, send (**Send**, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$, $N$) to $\mathcal{F}_{RL}$ and
         stop.
      2. Else if $i = 3$
         Send (**Send**, $sid$, ($sid$, $T$, $n_1$), $S$) to $\mathcal{F}_{sec}$ and stop.
      3. Else if $i = 5$
         Let $\mu_1 \leftarrow \mathcal{E}(k; pos, c; )$ , store $\mu_1$ on **Sessions**, send
         (**Send**, $sid$, ($sid$, $c$, $\mu_1$), $pos$) to $\mathcal{F}_{RL}$ and stop.
      4. Else if $i = 7$
         (i)   If $i_L = 7$ send (**Deny Linkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (ii)  Else if $i_U = 7$ send (**Deny Unlinkable**, $id$) to $\mathcal{F}_{KE}$ and stop.
         (iii) Else generate random key $k^{'}$, store $k^{'}$ on **Sessions** and send
               (**Est**. $User$, $k^{'}$, $id$) to $\mathcal{F}_{KE}$.
      5. Else if $i = 8$
         Let $\mu_2 \leftarrow \mathcal{E}(k; e; k^{'})$, store $\mu_2$ on **Sessions**, send
         (**Send**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$, $N$) to $\mathcal{F}_{RL}$ and stop.
      6. Else if $i = 10$
         Send (**Send**, $sid$, ($sid$, $n_2$), $S$) to $\mathcal{F}_{sec}$ and stop.
      7. Else if $i = 12$
         Send (**Est**. **MNO**, $k^{'}$, $pos$, $N$, $S$) and stop.
      8. Set $i \leftarrow i + 1$.
      9. Send ($i$, $sid$) to self.
```

```
The simulator $\mathcal{S}_{SKA}$ continued
While $class = 4$:
    On (Recv, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 2$, else ignore.
      2. Verify that $T$ is a valid token, else ignore.
      3. Verify that $T$, $n_1$ and $S$ match those on $sid$'s record on Sessions,
         else set $i_L \leftarrow 7$ and update $T$ and $n_1$ on $sid$'s record on Sessions.
      4. Else if no record containing $sid$ exists, store
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on Sessions.
      5. Set $i \leftarrow i + 1$.
      6. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $c$, $k$), $S$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 4$, else ignore.
      2. If a record
         (-, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, -, $n_1$, $S$, $c$, -, $\mu_1$, -, -, -, $i_U$, $i_L$)
         exists (i.e. there is no value for $id$), ignore.
      3. Else verify that $dec_{k_{US}}(c) = (sid, n_1, n_2, k, T, T', N)$ and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, -, -, -, $n_2$, $i_U$, $i_L$) on
         Sessions, else set $i_L \leftarrow 7$.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $c$, $\mu_1$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 6$, else ignore.
      2. Verify that $c$ matches that on $sid$'s record on Sessions else set
         $i_L \leftarrow 7$.
      3. Verify that $\mu_1$ matches that on $sid$'s record on Sessions, else set
         $i_U \leftarrow 7$.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.
```

The simulator $\mathcal{S}_{SKA}$ continued

```
While $class = 4$:
    **On** (**Recv**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 9$, else ignore.
      2. Verify that $\mu_2$ and $n_2$ match those on $sid$'s record on **Sessions**,
         else ignore.
      3. Set $i \leftarrow i + 1$.
      4. Send ($i$, $sid$) to self.

    **On** (**Recv**, $sid$, ($sid$, ok), $S$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 11$, else ignore.
      2. Set $i \leftarrow i + 1$.
      3. Send ($i$, $sid$) to self.

    **On** any other input, ignore.
```

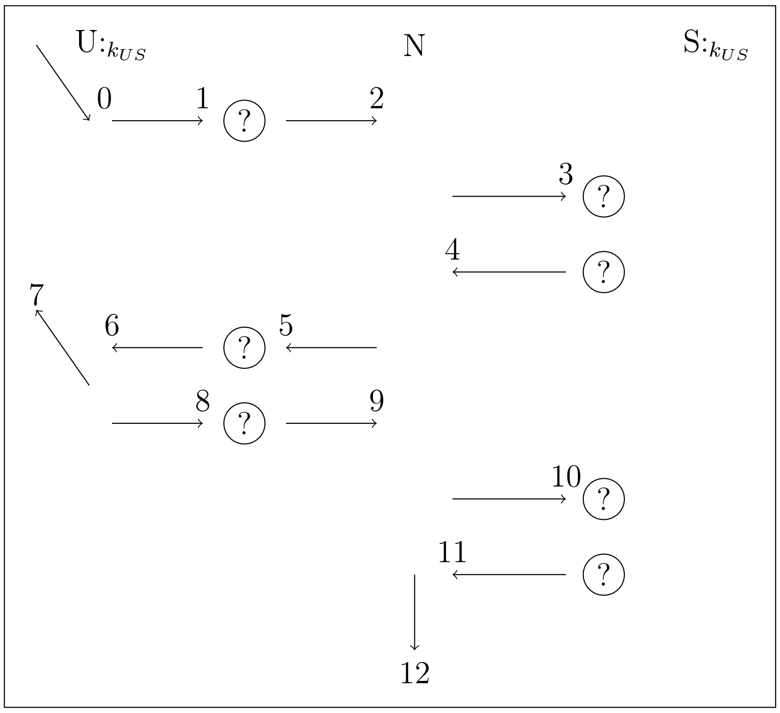

Figure 3.9: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 4.

```
The simulator $\mathcal{S}_{SKA}$ continued
Stored : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 5$:
    On (ENTER, $id$, $pos$, $U$, $N$, $S$) from $\mathcal{F}_{KE}$:
      1. Generate fresh session identifier $sid$ and store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, -, -, -, $S$, -, -, -, -, -, -, $i_U$, $i_L$), where
         $i = 0$, $i_U = 0$ and $i_L = 0$ on Sessions and send ($i$, $sid$) to self.

    On ($i$, $sid$) from self:
      1. If $i = 1$
         Generate random $T$ of Token length, and nonce $n_1$, store
         ($id$, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -, $i_U$, $i_L$) on
         Sessions, send (Send, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$, $N$) to $\mathcal{F}_{RL}$ and
         stop.
      2. Else if $i = 3$
         (i)   If $i_L = 3$ send (Deny Linkable, $id$) to $\mathcal{F}_{KE}$ and stop.
         (ii)  Else if $i_U = 3$ send (Deny Unlinkable, $id$) to $\mathcal{F}_{KE}$ and stop.
         (iii) Else generate random key $k'$, store $k'$ and send
               (Est. User, $k'$, $id$) to $\mathcal{F}_{KE}$.
      3. Else if $i = 4$
         Let $\mu_2 \leftarrow \mathcal{E}(k; e; k')$, store
         ($id$, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, $\mu_1$, $k'$, $\mu_2$, $n_2$, $i_U$,
         $i_L$) on Sessions, send (Send, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$, $N$) to $\mathcal{F}_{RL}$
         and stop.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    On (Recv, $sid$, ($sid$, $c$, $\mu_1$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 2$, else ignore.
      2. Verify that $dec_{k_{US}}(c) = (sid, n_1, n_2, k, T, T', N)$, and store
         ($id$, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, $\mu_1$, -, -, $n_2$, $i_U$, $i_L$)
         on Sessions, else set $i_L \leftarrow 3$.
      3. Verify that $\mu_1 = \mathcal{E}(k; pos, c; )$, else set $i_U \leftarrow 3$.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    On any other input, ignore.
```

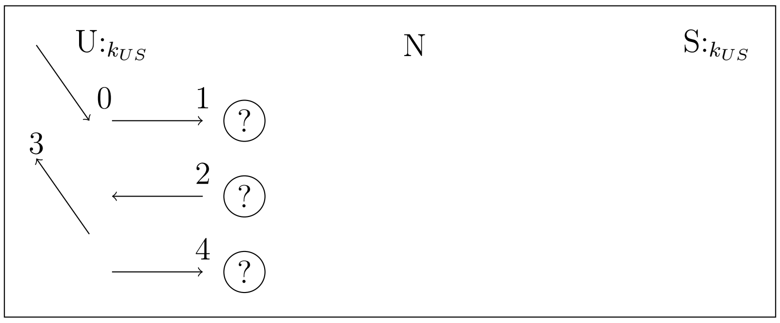

Figure 3.10: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 5.

```
The simulator $\mathcal{S}_{SKA}$ continued
Stored : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 6$:
    On $(i, sid)$ from self:
      1. If $i = 5$
         Generate random $T'$ of token length, nonce $n_2$ and key $k$, set
         $c \leftarrow enc_{k_{US}}(sid, n_1, n_2, k, T, T', N)$, let $\mu_1 \leftarrow \mathcal{E}(k; pos, c; )$,
         store $(id, sid, class, i, pos, U, N, T, T', n_1, S, c, k, \mu_1, -, -, n_2)$
         on Sessions, send (Send, $sid$, $(sid, c, \mu_1)$, $pos$) to $\mathcal{F}_{RL}$ and stop.
      2. Else if $i = 9$
         Send (Est. SP, $id$, $U$, $N$, $S$) to $\mathcal{F}_{KE}$.
      3. Else if $i = 12$
         Send (Est. MNO, $k'$, $id$, $pos$) to $\mathcal{F}_{KE}$ and stop.
      4. Set $i \leftarrow i + 1$.
      5. Send $(i, sid)$ to self.

    On (Recv, $sid$, $(sid, T, n_1, S)$, $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 0$, else ignore.
      2. Store (-, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -) where
         $i = 0$ on Sessions.
      3. Verify that $T$ is a valid token, else ignore.
      4. Set $i \leftarrow i + 1$.
      5. Send $(i, sid)$ to self.

    On (Recv, $sid$, $(sid, n_2, \mu_2)$, $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 6$, else ignore.
      2. Verify that $\mu_2 = \mathcal{E}(k; e; k')$ and store $k'$ on $sid$'s
         record on Sessions, else ignore.
      3. Verify that $n_2$ matches that on $sid$'s record on Session, else
         ignore.
      4. Set $i \leftarrow i + 1$.
      5. Send $(i, sid)$ to self.

    On any other input, ignore.
```

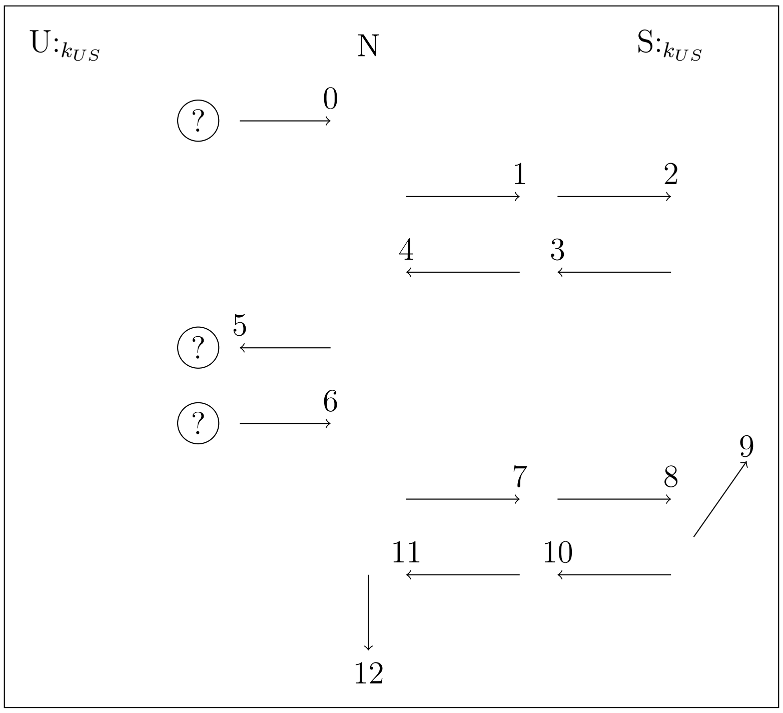

Figure 3.11: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 6.

The simulator $\mathcal{S}_{SKA}$ continued

```
**Stored** : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 7$:
    **On** $(i,\ sid)$ from self:
      1. If $i = 1$
         Generate random $T^{'}$ of token length, nonce $n_2$ and key $k$,
         set $c \leftarrow enc_{k_{US}}(n_1, n_2, k, T, T^{'}, N)$, store
         $(id,\ sid,\ class,\ i,$ -, $U,\ N,\ T,\ T^{'},\ n_1,\ S,\ c,\ k,$ -, -, -, $n_2)$ on
         **Sessions**, send (**Send**, $sid$, $(sid,\ c,\ k)$, $N$) to $\mathcal{F}_{sec}$ and stop.
      2. Else if $i = 3$
         Send (**Est**. **SP**, $id$, $U$, $N$, $S$) to $\mathcal{F}_{KE}$.
      3. Else if $i = 4$
         Send (**Send**, $sid$, $(sid,$ ok$)$, $N$) to $\mathcal{F}_{sec}$ and stop.
      4. Set $i \leftarrow i + 1$.
      5. Send $(i,\ sid)$ to self.

    **On** (**Recv**, $sid$, $(sid,\ T,\ n_1)$, $N$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Verify that $i = 0$, else ignore.
      2. Store (-, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -) where
         $i = 0$ on **Sessions**.
      3. Verify that $T$ is a valid token, else ignore.
      4. Set $i \leftarrow i + 1$.
      5. Send $(i,\ sid)$ to self.

    **On** (**Recv**, $sid$, $(sid,\ n_2)$, $N$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Verify that $i = 2$, else ignore.
      2. Verify that $n_2$ matches that on $sid$'s record on **Session**, else
      ignore.
      3. Set $i \leftarrow i + 1$.
      4. Send $(i,\ sid)$ to self.

    **On** any other input, ignore.
```

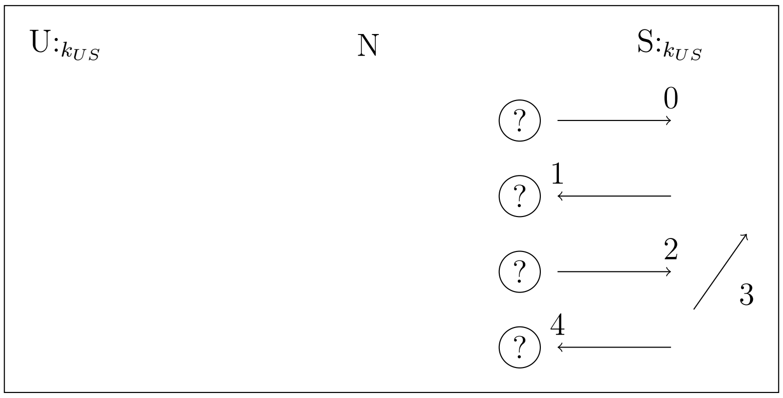

Figure 3.12: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 7.

```
The simulator $\mathcal{S}_{SKA}$ continued
Stored : $U$, $N$, $S$, $\mathcal{F}_{RL}$, $\mathcal{F}_{sec}$, $\{k_{US}; U, S\}$.
While $class = 8$:
    On $(i, sid)$ from self:
      1. If $i = 1$
         Send (Send, $sid$, $(sid, T, n_1)$, $S$) to $\mathcal{F}_{sec}$ and stop.
      2. Else if $i = 3$
         Let $\mu_1 \leftarrow \mathcal{E}(k; pos, c; )$ , store $\mu_1$ on Sessions, send
         (Send, $sid$, $(sid, c, \mu_1)$, $pos$) to $\mathcal{F}_{RL}$ and stop.
      3. Else if $i = 5$
         Send (Send, $sid$, $(sid, n_2)$, $S$) to $\mathcal{F}_{sec}$ and stop.
      4. Else if $i = 7$
         Send (Est. MNO, $k'$, $pos$, $N$, $S$) to $\mathcal{F}_{KE}$ and stop.
      5. Set $i \leftarrow i + 1$.
      6. Send $(i, sid)$ to self.
```

The simulator $\mathcal{S}_{SKA}$ continued

```
While $class = 8$:
    **On** (**Recv**, $sid$, ($sid$, $T$, $n_1$, $S$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 0$, else ignore.
      2. Verify that $T$ is a valid token, else ignore.
      3. Store (-, $sid$, $class$, $i$, -, $U$, $N$, $T$, -, $n_1$, $S$, -, -, -, -, -, -) where
         $i = 0$ on **Sessions**.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    **On** (**Recv**, $sid$, ($sid$, $c$, $k$), $S$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 2$, else ignore.
      2. Store $c$ and $k$ on $sid$'s record on **Sessions**.
      3. Set $i \leftarrow i + 1$.
      4. Send ($i$, $sid$) to self.

    **On** (**Recv**, $sid$, ($sid$, $n_2$, $\mu_2$), $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Verify that $i = 4$, else ignore.
      2. Verify that $\mu_2 = \mathcal{E}(k; e; k')$, else ignore.
      3. Store $k'$, $n_2$ and $\mu_2$ on $sid$'s record on **Sessions**.
      4. Set $i \leftarrow i + 1$.
      5. Send ($i$, $sid$) to self.

    **On** (**Recv**, $sid$, ($sid$, ok), $S$) from $\mathcal{A}/\mathcal{F}_{sec}$:
      1. Check that $i = 6$, else ignore.
      2. Set $i \leftarrow i + 1$.
      3. Send ($i$, $sid$) to self.

    **On** any other input, ignore.
```

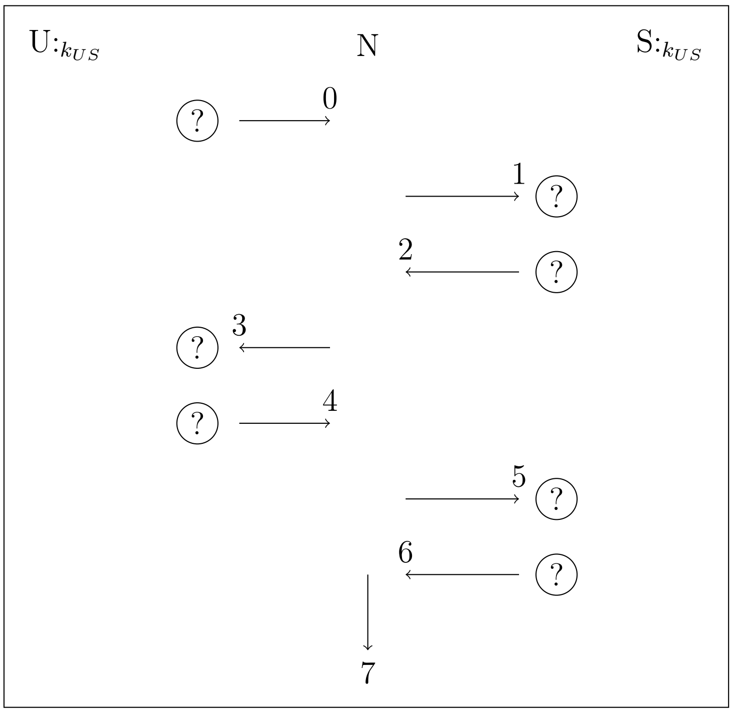

Figure 3.13: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 8.

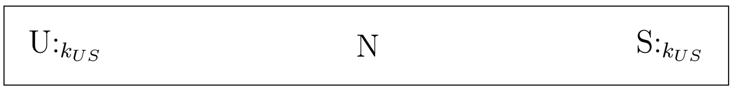

Figure 3.14: Summary of the anonymous symmetric key establishment protocol π_{SKA} for Class 9.

```
The simulator $\mathcal{S}_{SKA}$ continued
For all classes:
    On (LEAVE) from $\mathcal{F}_{KE}$:
      1. Send (LEAVE) to self.

    On (ENTER) or (LEAVE) from self:
      1. Stop.

    On (LEAVE, $id$) from $\mathcal{F}_{KE}$ with ($id$, $\cdots$ , $class$, $\cdots$) recorded:
      1. If
         ($id$, $sid$, $class$, $i$, $pos$, $U$, $N$, $T$, $T'$, $n_1$, $S$, $c$, $k$, $\mu_1$, $sk$, $k'$, $\mu_2$, $n_2$,
         $i_U$, $i_L$) is recorded then send (LEAVE) to self, remove the entry
         and hand over (DENY, $id$) to $\mathcal{F}_{KE}$.

    On (LISTEN, $pos$) from $\mathcal{A}/\mathcal{F}_{RL}$:
      1. Store ($pos$) on Corrupt and on $\mathcal{F}_{RL}$'s Corrupt, set
         $class \leftarrow class + 1$, and hand over (LISTEN, $pos$) to $\mathcal{F}_{KE}$.
```

3.3 Security Analysis

We will now show that the executions only can take on n possible forms. If we can simulate all of the possible forms we have proved that the protocol realizes the ideal functionality. We will do this class wise, starting with Class 1 as the reasoning for Class 0 is trivial. Further, we will omit Class 9 as the adversary is free to do whatever he wants in this class. We wish to find every possible fragment of the protocol execution for all classes. Before we begin the security analysis we will introduce some terms that will be used throughout the analysis. More detailed descriptions of these terms along with proofs of reductions between the different notions can be found in [BDPR98] (notions for public encryption) and [BDJR97] (notions for symmetric encryption).

We will start by describing INT-CTXT security. Assume that an adversary $\mathcal{A}$ wants to break an encryption scheme where $enc_k(m) = c$ and $dec_k(enc_k(m)) = dec_k(c) = m$. We let $\mathcal{A}$ be free to encrypt any plaintext m using the given encryption function. For the encryption function to be INT-CTXT secure, an adversary should not from the previous gain any advantage in being able to produce a cipher-text that is different to the ones he has already encrypted. That is, $\mathcal{A}$ should not be able to produce a c so that $\mathcal{D}_k(c) \neq \perp$.

We will now go on to describe RoR-CCA (Real or Random) security. An adversary $\mathcal{A}$ wants to decide if an encrypted message, c is an encryption of random noise, or an encryption of a set of messages he chooses. $\mathcal{A}$ is allowed to see encryptions of any plaintext message, and decryptions of any ciphertext besides the ones in question. If the adversary is not able to distinguish $\mathcal{E}_k(m)$ from $\mathcal{E}_k(noise)$ for a given encryption scheme, the scheme is RoR-CCA secure.

We are using a functionality called $\mathcal{F}_{\text{shared key enc.}}$ to encrypt data under k_{US}. This functionality is described in [GPS12]. This means that only an honest U or an honest S will be able to encrypt and decrypt data using k_{US}. We will assume this encryption function as well as the AEAD function to be both IND-CCA and RoR-CCA secure.

When doing a security analysis one should base it on the given protocol. In all three protocols described in this paper it is specified what record must be stored for a given message to be accepted. If the correct record is not stored the received message will simply be ignored. For instance, the protocol π_{SKA} of Figure 3.2 requires (sid, T, n_1, n_2, U, N) to be stored for (sid, n_2) to be accepted. The record (sid, T, n_1, n_2, U, N) is stored when S receives (sid, T, n_1) from N and creates a ($c = \{sid$, n_1, n_2, k, T, T', $N\}_{k_{US}}$) that it sends to N as a part of (sid, c, k). Thus, the two statements

- S received (sid, n_2) from $\mathcal{A}$ for a record where (sid, T, n_1, n_2, U, N) is recorded

- S received (sid, n_2) from $\mathcal{A}$ for a record where he already sent (sid, c, k) to N

are equivalent. With this in mind we will proceed with the security analysis.

3.3.1 Class 1

We will start with Class 1. There are two possible tracks that the protocol might follow in this class. The first track is shown in Figure 3.15.

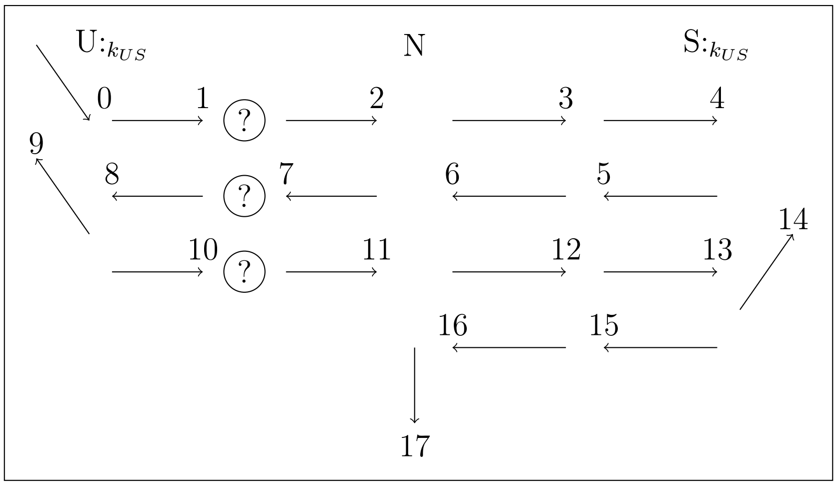

Figure 3.15: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 1.

If the protocol follows this track one possibility is that N accepts. Let us look at what we know when N accepts.

N accepts:

- N received (sid, ok) from S for a record where he already sent (sid, n_2) to S.
- Since S sent (sid, ok) to N, S must have accepted.

S accepts:

- S received (sid, n_2) for a record where he already sent (sid, c, k) to N.
- N received (sid, n_2, $\mathcal{E}(k;\ e;\ k^{'})$) from $pos.$ and N accepted it.
- Due to RoR-CCA only S and U know n_2 which is encrypted using k_{US} as apart of c.
- U sent n_2.

- U accepted.

U accepts:

- U received (sid, c, $\mathcal{E}(k$; pos, c;)), where the decryption of c matches a record, from $\mathcal{A}/pos$ for a record where he already sent (sid, T, n_1, S) to $\mathcal{A}/pos$.
- Only S and U know k_{US}, and hence due to INT-CTXT c is made by S.
- $\exists$ a session on S with $sid, n_1, n_2, k, T, T^{'}, N$ like those in c.
- At this point only U, N and S know k, and thus due to INT-CTXT N made $\mathcal{E}(k$; pos, c;).
- N received (sid, c, k) from S.
- S received (sid, T, n_1) from N.
- N received (sid, T, n_1, S) from $\mathcal{A}/pos$.
- Since U accepted he agrees with S on wanting to start a session with the variables contained in c.
- Only U and N know $k^{'}$.

All of this must have happened in this exact way if N accepts. Between S and N accepting, nothing could have gone wrong, and thus, as long as S accepts, we also find ourselves in Track 1. The protocol will for Class 1 only ever abort in Step 9 of Figure 3.15. It may output either (**Est**. **Failed Linkable**) or (**Est**. **Failed Unlinkable**).

U outputs **Est**. **Failed Unlinkable** in Step 9 in Figure 3.15:

- U received a (sid, c, $\mathcal{E}(k$; pos, c;)) from $\mathcal{A}/pos$, where c was correct but the AEAD was wrong.
- Due to INT-CTXT, c is created by S.
- Since S created c, S must have received (sid, T, n_1) from N. Also, S must have sent (sid, c, k) to N.
- N sent (sid, c, $\mathcal{E}(k$; pos, c_1;)) to $\mathcal{A}/pos$.
- Since N sent (sid, T, n_1) to S, N must have received (sid, T, n_1, S) from $\mathcal{A}/pos$.
- Since U accepted c, U must have tried to start a session with the same n_1, and thus sent (sid, T, n_1, S) to $\mathcal{A}/pos$.

All of this must have happened in this exact way if U outputs **Est**. **Failed Unlinkable** in Step 9 of Figure 3.15. We then find ourselves in the top part of Track 1. Let us look at what must have happened if U outputs **Est**. **Failed Linkable** in Step 9 of Figure 3.15.

U outputs **Est**. **Failed Linkable** in Step 9 in Figure 3.15:

- U received a (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) from $\mathcal{A}/pos$, where the decryption of c using k_{US} is not on the form {sid, n_1, -, -, T, -, -}
- There are two things that might have happened here. Either $\mathcal{A}$ sent something to N in Step 1 of Figure 3.15 or he sent (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) to U directly as shown in Figure 3.16.
- Let us start by looking at what must have happened if he sent something to N:
- $\mathcal{A}$ must have sent (sid, T, any nonce, S) to N. If sid was not correct the message would just have been ignored. If T or S was wrong the message would have been ignored in Step 4 of the protocol.
- N received (sid, T, n_1, S) and sent (sid, T, n_1) to S.
- S sent (sid, c, k) to N.
- N sent (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) to $\mathcal{A}/pos$.

When receiving (sid, T, n_1, S) from U, $\mathcal{A}$ could chose to send (sid, random binary string of appropriate length, random binary string of appropriate length) to U. The way the protocol is written U decrypts c before even looking at the AEAD. Thus, when c turned out to be wrong U would simply output **Est**. **Failed Linkable**. This is consistent with the simulator and is depicted in Figure 3.16. However, $\mathcal{A}$ will at any point during an execution that starts in Track 1 in Class 1 be able to send something to U to make U output **Est**. **Failed Linkable**. This is still a part of Track 1 and we have only included Figure 3.16 in order to be able to explain what happens in an easier manner.

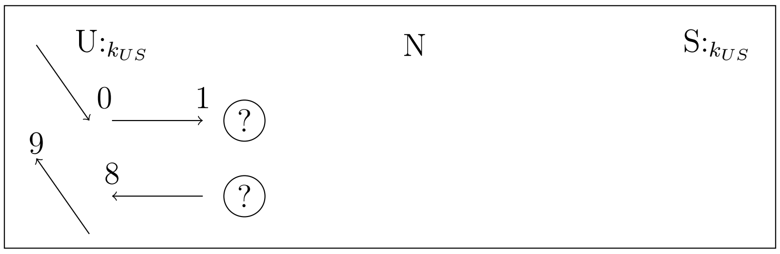

Figure 3.16: A part of Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 1.

These are the only things that could happen when U outputs **Est**. **Failed Linkable**.

The way the protocol is written it is possible for an adversary present at pos to send (sid, T, n_1, S) to N without having received it from U first. The only thing he has to make sure of is that the sid he sends does not already exist on any record. This is depicted in Figure 3.17 as Track 2.

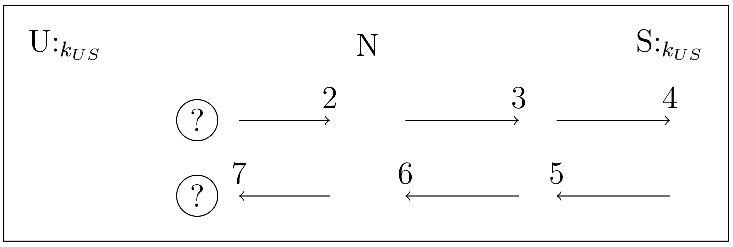

Figure 3.17: Track 2 of the anonymous symmetric key establishment protocol π_{SKA} for Class 1.

$\mathcal{A}$ sends (sid, T, n_1, S) to N as shown in Figure 3.17:

- N sends (sid, T, n_1) to S.
- S receives it and sends (sid, c, k) to N.
- N sends (sid, c, $\mathcal{E}(k$; pos, c;) to $\mathcal{A}$.

Since sid does not exist on any of U's records, $\mathcal{A}$ will in this instance never be able to send anything to U in step 8 of Figure 3.15 that U will accept. Thus, in this track $\mathcal{A}$ will just send (sid, T, n_1, S) to N and the receive (sid, c, $\mathcal{E}(k$; pos, c;)) some time after he sends the first message. This is all that ever will happen in Track 2.

A last possibility is that the protocol starts its execution in Track 1. At any point $\mathcal{A}$ could send (sid, bit string, bit string) to U. The execution will then split up so that part of it becomes equivalent with the part of Track 1 that is shown in Figure 3.16, and the other part of it becomes equivalent to Track 2 shown in Figure 3.17.

We have now described all the scenario that might take place in Class 1. All of the above scenarios can be simulated by $\mathcal{S}_{SKA}$, and hence, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 1.

3.3.2 Class 2

In Class 2 pos and N are corrupt. Let us start by looking at what guarantees we have if S accepts:

S accepts:

- S sends (sid, ok) to $\mathcal{A}$.
- S must have received (sid, n_2) from $\mathcal{A}$ for a record where he already sent (sid, c, k) to $\mathcal{A}$.

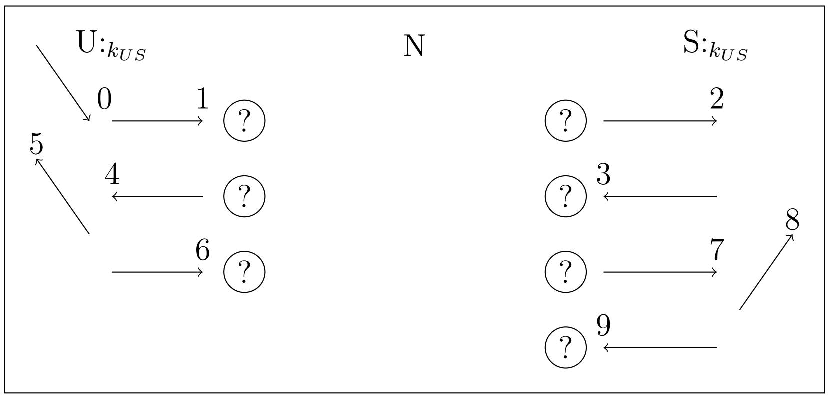

Figure 3.18: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 2.

- $\mathcal{A}$ does not have access to k_{US} and thus can not decrypt c. Due to RoR-CCA it can therefore not know n_2.
- U must have accepted.

U accepts:

- U must have received a $(sid,\ c,\ \mathcal{E}(k;\ pos,\ c;\ ))$ from $\mathcal{A}$ for a record where he already sent $(sid,\ T,\ n_1,\ S)$ to $\mathcal{A}$.
- Due to INT-CTXT S must have created c.
- S and U agree on all the entries in c.
- S must have sent $(sid,\ c,\ k)$ to $\mathcal{A}$.
- S must have received $(sid,\ T,\ n_1)$ from $\mathcal{A}$.

Everything must have happened exactly as described above if S accepts. It is easy to see that this execution follows Track 1. For Class 2 the protocol will only ever abort in Step 5 of Figure 3.18. Let us see what we know when the protocol outputs **Est**. **Failed Unlinkable** or **Est**. **Failed Linkable** in Step 5 of Figure 3.18.

U outputs **Est**. **Failed Unlinkable** in Step 5 of Figure 3.18:

- U received a $(sid,\ c,\ \mathcal{E}(k;\ pos,\ c;\ ))$ for a record where he already sent $(sid,\ T,\ n_1,\ S)$ to $\mathcal{A}$, where c was correct but the AEAD was wrong.
- Since c is correct sid, n_1 and T must be contained in c.
- Due to INT-CTXT S created c.
- Since S created c, S received $(sid,\ T,\ n_1)$ from $\mathcal{A}$.
- S sent $(sid,\ c,\ k)$ to $\mathcal{A}$.

All of this must have happened if U outputs **Est. Failed Unlinkable**. This execution is simply a part of Track 1 and as such, can be simulated.

U outputs **Est. Failed Linkable** in Step 5 of Figure 3.18:

- U received a $(sid,\ c,\ \mathcal{E}(k;\ pos,\ c;\ ))$ where c was wrong.
- U must have sent $(sid,\ T,\ n_1,\ S)$ to $\mathcal{A}$ or the message would have been ignored.

There are now two options. Either, $\mathcal{A}$ sent something to S, or not. Let us start by looking at what must have happened if $\mathcal{A}$ sent something to S;

S must have received $(sid,\ T,\ n_1)$. T must be a valid token as S would have just ignored the message otherwise. Further, S must have sent $(sid,\ c,\ k)$ to $\mathcal{A}$.

If this is the case we are again only operating in a smaller part of Track 1. Track 1 is covered by the simulator. Let us go on to look at what happens when $\mathcal{A}$ simply sends $(sid$, bit string, bit string$)$ to U without involving S. When c turns out to be wrong U will simply output **Est. Failed Linkable**. This is depicted in Figure 3.19 and will simply be a part of Track 1. As we stated in Section 3.3.1, $\mathcal{A}$ will be free to send a message to U at any point after Step 1 and before Step 4. This will cause U to abort, and will split up the execution so that part of it follows the part of Track 1 that is shown in Figure 3.19, and the other part Track 2 which is depicted in Figure 3.20.

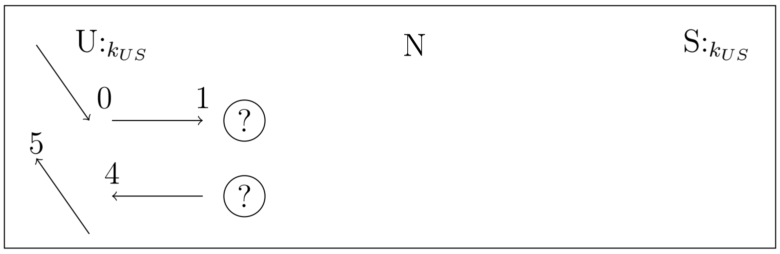

Figure 3.19: Part of Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 2.

The protocol does not require for *sid* to be stored when S receives $(sid,\ T,\ n_1)$. As such it would be possible for $\mathcal{A}$ to send $(sid,\ T,\ n_1)$ to S without U having attempted to start a session. This is shown in Figure 3.20 and we have, as stated, named it Track 2. S will have to send a valid T to S so that the message is not simply ignored. This track can also be simulated by $\mathcal{S}_{SKA}$.

The two tracks described above are the only possible tracks for Class 2. Since they are all consistent with the simulator $\mathcal{S}_{SKA}$, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 2.

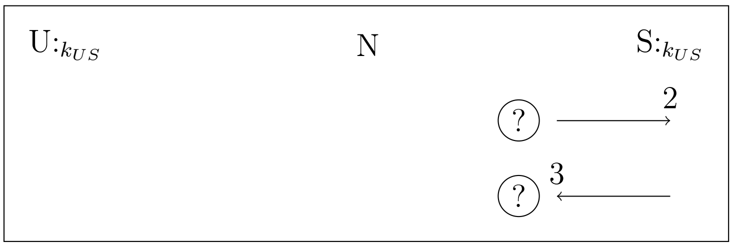

Figure 3.20: Track 2 of the anonymous symmetric key establishment protocol π_{SKA} for Class 2.

3.3.3 Class 3

In Class 3 only S is corrupt. An adversary will never be able to send anything to N for a given *sid* before it has received (*sid*, T, n_1) for that *sid* from N. Thus, the only track that exists for Class 3 is Track 1 described in Figure 3.21.

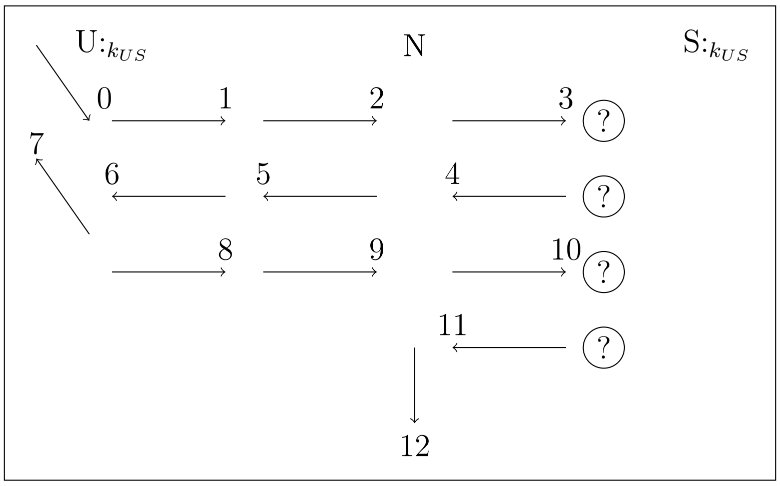

Figure 3.21: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 3.

Let us look at what guarantees we have if N accepts.

N accepts:

- N must have received (*sid*, ok) from $\mathcal{A}$ for a record where he already sent (*sid*, n_2) to $\mathcal{A}$.
- Since N sent (*sid*, n_2) to $\mathcal{A}$, he must have received (*sid*, n_2, $\mathcal{E}(k;\, e;\, k')$) from U.

- U accepted.
- N and U will output the same k' as no one will have been able to change it between it leaving U and reaching N. They will also be the only ones who know k'.

U accepts:

- U received a $(sid, c, \mathcal{E}(k; pos, c;))$ from N that it accepted.
- U must have sent (sid, T, n_1, S) to N or it would not have accepted.
- Since N sent $(sid, c, \mathcal{E}(k; pos, c;))$ to U, N must have received (sid, c, k) from $\mathcal{A}$ for a record where he already sent (sid, T, n_1) to $\mathcal{A}$.
- Since N sent (sid, T, n_1) to $\mathcal{A}$ he must have received (sid, T, n_1, S) from U.

Things must have happened exactly as described above when N accepts. We can see that this is consistent with Track 1. Since Track 1 can be simulated by $\mathcal{S}_{SKA}$ everything is okay so far.

There is only one way the protocol can fail in Class 3, and that is if $\mathcal{A}$ sends the wrong (sid, c, k) to N as all other input from $\mathcal{A}$ that does not lead to N accepting is ignored. The message will then go on until U receives it in Step 7 of Figure 3.21. U will then send (**Deny Linkable**, id) to $\mathcal{F}_{KE}$.

U outputs **Est**. **Failed Linkable** in Step 7 of Figure 3.21:

- U has received a $(sid, c, \mathcal{E}(k; pos, c;))$ from N that is wrong. That is, the decryption of c is not on the format $(sid, n_1, -, -, T, -, N)$.
- Since N sent $(sid, c, \mathcal{E}(k; pos, c;))$ to U, N must have received (sid, c, k) from $\mathcal{A}$.
- Since N did not just ignore the message from $\mathcal{A}$, N must have sent (sid, T, n_1) to $\mathcal{A}$.
- Since N sent the above to $\mathcal{A}$, N must have received (sid, T, n_1, S) from U.

We can see that this is simply a part of Track 1, and thus it is consistent with the simulator. Hence, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ within Class 3.

3.3.4 Class 4

In Class 4 both S and pos are corrupt. This gives the adversary a definite advantage. Clearly, the protocol may execute in a normal manner and follow the path depicted in Figure 3.22. We have chosen to call this path Track 1.

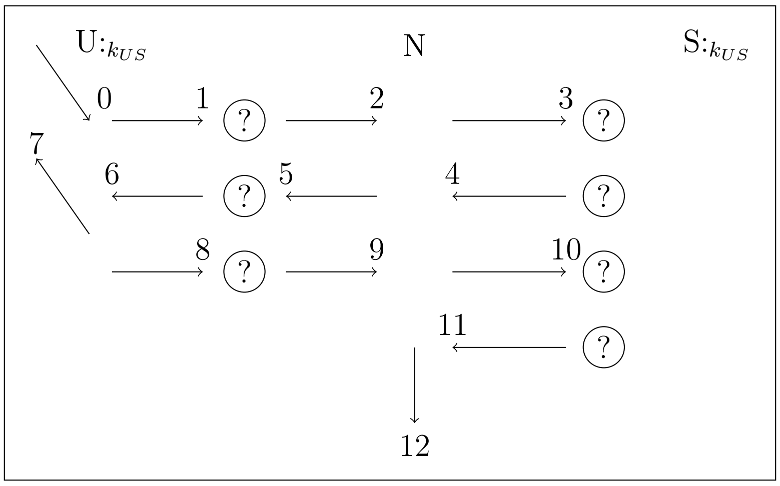

Figure 3.22: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 4.

Let us go on to have a look at what might happen when N accepts.

N accepts:

- N received (sid, ok) from S for a sid where he had already sent (sid, n_2) to $\mathcal{A}$.
- N must have received (sid, n_2, $\mathcal{E}(k;\, e;\, k')$) from $\mathcal{A}$ for a record where he already sent (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$ to $\mathcal{A}$. Since $\mathcal{A}$ has knows k, n_2 and e he could have created this message.
- N must have received (sid, c, k) from $\mathcal{A}$ for a record where he already sent (sid, T, n_1) to $\mathcal{A}$.
- N must have received(sid, T, n_1, S) from $\mathcal{A}$.
- U has not necessarily been involved in this protocol at all.

We can see that only N and $\mathcal{A}$ needs to be present for N to accept. This is depicted in Figure 3.23 and we have named the track the protocol follows Track 2.

Let us however assume that U has in fact been present.

N accepts with U present:

- N received (sid, ok) from $\mathcal{A}$ for a sid where he had already sent (sid, n_2) to $\mathcal{A}$.
- N must have received (sid, n_2, $\mathcal{E}(k;\, e;\, k')$) from $\mathcal{A}$.
- Since pos is collaborating with S they know k, and $\mathcal{A}/pos$ would have been able to alter this AEAD, and so this AEAD is not necessarily the same one U sent to $\mathcal{A}$.

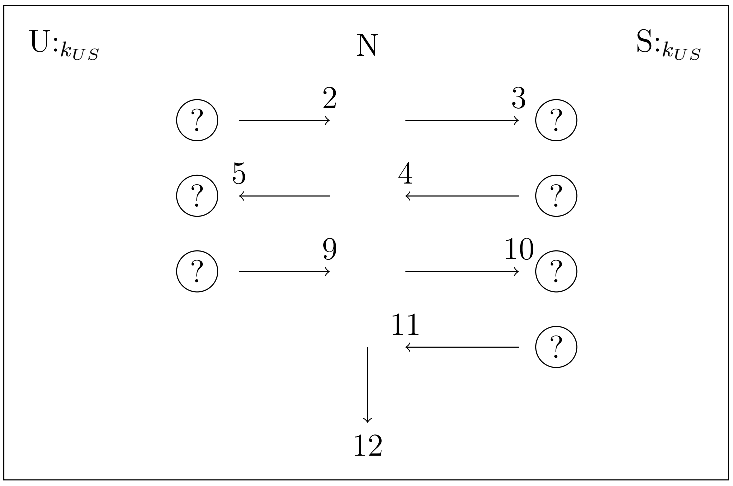

Figure 3.23: Track 2 of the anonymous symmetric key establishment protocol π_{SKA} for Class 4.

- N and U might not have the same value for k'.
- Neither N nor U will know that *pos* and S were dishonest.
- N and U will output different values for k' whilst thinking they agree on k'.

We will talk more about this security breach in the Chapter 6. As long as both N and U are present during an execution, the execution will follow Track 1. Track 1 can be simulated by $\mathcal{S}_{SKA}$.

We will now go on to see what guarantees we have if U accepts.

U accepts:

- U received (sid, c, $\mathcal{E}(k;\ pos,\ c;\ )$) from $\mathcal{A}$ where both c and the AEAD were correct.
- U sent (sid, T, n_1, S) to $\mathcal{A}$, or else he would not have accepted.

This is all we know when U accepts. Thus, N does not need to be involved in order for $\mathcal{A}$ to get U to accept. If this is the case the execution will follow the path shown in Figure 3.24. This path is still a part of Track 1. As discussed for the other classes, an adversary $\mathcal{A}$ can send a message to U at any point in between Steps 1 and 6. Thus the path depicted in Figure 3.24 is a part of Track 1.

In Class 4 the protocol has 2 starting points, Step 0 and Step 2 of Figure 3.22. If it starts in Step 0 it will follow Track 1. If it starts in Step 2 it will follow Track 2. As we have talked about in the previous sections, an execution could just as well start

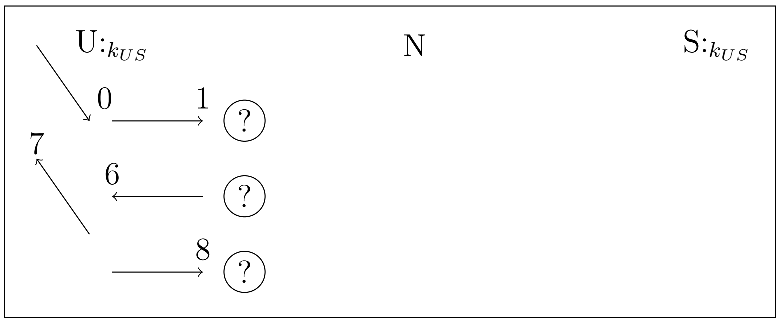

Figure 3.24: Part of Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 4.

out following Track 1 and later on split so that part of the execution runs as shown in Figure 3.23 and part of it as shown in Figure 3.24. This is not problematic. Both tracks and the transition between them are consistent with the simulator $\mathcal{S}_{SKA}$.

Let us now go on to look at what happens when U outputs (**Est**. **Failed Linkable**) or (**Est**. **Failed Unlinkable**) in Step 7 of Figure 3.22.

U outputs (**Est**. **Failed Linkable**) in Step 7 of Figure 3.22:

- U received a (sid, c, $\mathcal{E}(k$; pos, c;)) from $\mathcal{A}$ where the decryption of c was not on the form $\{sid$, n_1, -, -, T, -, $N\}$.
- Since the above message was not ignored U must have sent (sid, T, n_1, S) to $\mathcal{A}$.
- This is all we know. We can see that this execution is consistent with the part of Track 1 that is depicted in Figure 3.24.

U outputs (**Est**. **Failed Unlinkable**) in Step 7 of Figure 3.22:

- U received a (sid, c, $\mathcal{E}(k$; pos, c;)) from $\mathcal{A}$ where c was correct but the AEAD was wrong. Thus, it can be simulated by $\mathcal{S}_{SKA}$.
- Since the above message was not ignored U must have sent (sid, T, n_1, S) to $\mathcal{A}$.
- This is all we know. We can see that this execution is consistent with the part of Track 1 that is depicted in Figure 3.24. Thus, it can be simulated by $\mathcal{S}_{SKA}$.

N might be involved in these executions. If that is the case the execution will be consistent with Track 1 as shown in Figure 3.22. Also here the execution might split up into one execution following Track 1 and one execution following Track 2. As discussed this is not problematic. Thus, all possible executions leading to U

outputting (**Est**. **Failed Linkable**) or (**Est**. **Failed Unlinkable**) can be simulated by $\mathcal{S}_{SKA}$.

As before, we can classify all the executions of the protocol for Class 4, and all possibilities are covered by the simulator, $\mathcal{S}_{SKA}$. Hence, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 4.

3.3.5 Class 5

In Class 5 only U is honest. Thus, the only possible track is depicted in Figure 3.25.

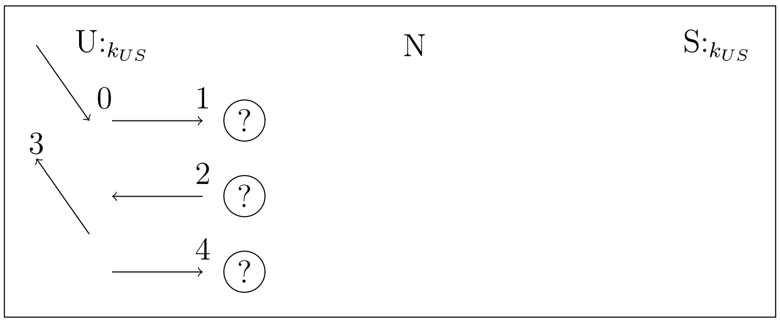

Figure 3.25: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 5.

Let us have a look at what happens when U accepts.

U accepts:

- U received a (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) from $\mathcal{A}$ that U accepted.
- U sent (sid, T, n_1 S) to $\mathcal{A}$ or he would not have accepted.
- U will output a $k^{'}$ that N will be able to alter.
- U will send (sid, n_2, $\mathcal{E}(k;\, e;\, k^{'})$) to $\mathcal{A}$.

This execution clearly follows Track 1 and is covered by the simulator. Let us now see what happens if U outputs (**Est**. **Failed Unlinkable**) or (**Est**. **Failed Linkable**) in Step 3 of Figure 3.25.

U outputs (**Est**. **Failed Unlinkable**) in Step 3 of Figure 3.25:

- U received (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) where c was correct, but the AEAD was wrong.
- Since the message was not ignored U must have sent (sid, T, n_1, S) to $\mathcal{A}$.

- This execution is covered by Track 1.

U outputs (**Est**. **Failed Linkable**) in Step 3 of Figure 3.25:

- U received $(sid, c, \mathcal{E}(k; pos, c;))$ where c was wrong.
- Since the message was not ignored U must have sent (sid, T, n_1, S) to $\mathcal{A}$.
- This execution is covered by Track 1.

Again, we can simulate all executions that might occur, and the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 5.

3.3.6 Class 6

In Class 6 only U is corrupt. We will show that the only exists one possible path an execution may follow in Class 6. We have named this path Track 1 and it is depicted in Figure 3.26. Let us start by having a look at what happens if N accepts.

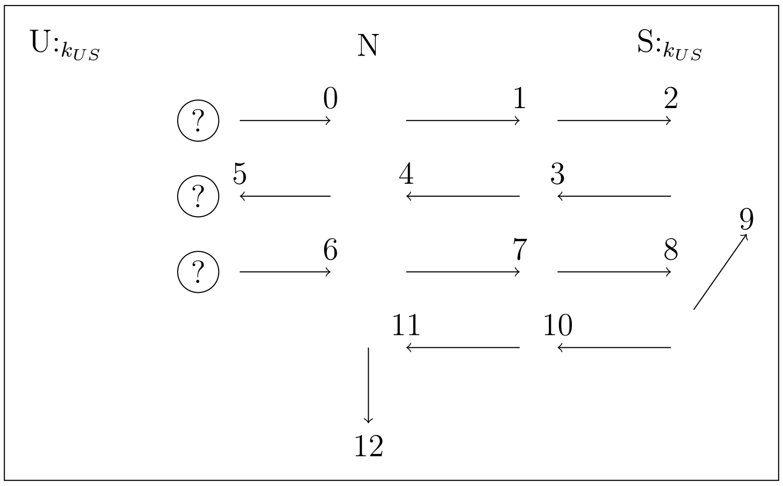

Figure 3.26: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 6.

N accepts:

- N received (sid, ok) from S for a record where he already sent (sid, n_2) to S.
- S accepted.

S accepts:

- S received (sid, n_2) from N for a record where he already sent (sid, c, k) to N.
- Since N sent (sid, n_2) to S, N must have received (sid, n_2, $\mathcal{E}(k;\, e;\, k')$) from $\mathcal{A}$.
- N must have sent (sid, c, $\mathcal{E}(k;\, pos,\, c;\,)$) to $\mathcal{A}$ or he would not have sent (sid, n_2) to S when he received (sid, n_2, $\mathcal{E}(k;\, e;\, k')$) from $\mathcal{A}$.
- N must have received (sid, c, k) from S.
- S must have received (sid, T, n_1) from N.
- N must have received (sid, T, n_1, S) from $\mathcal{A}$.

This follows Track 1 and is consistent with the simulator. This class as mentioned only has 1 track. $\mathcal{A}$ will never be able to send a wrong message to N that is not ignored. He could chose to stop after he receives the message from N in Step 5 of Figure 3.26. However, this would only give us a smaller portion of Track 1. Since U is the corrupt party we do not have to simulate the protocol aborting. This will be done by the adversary. Thus, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ also for Class 6.

3.3.7 Class 7

In Class 7 only S is honest.

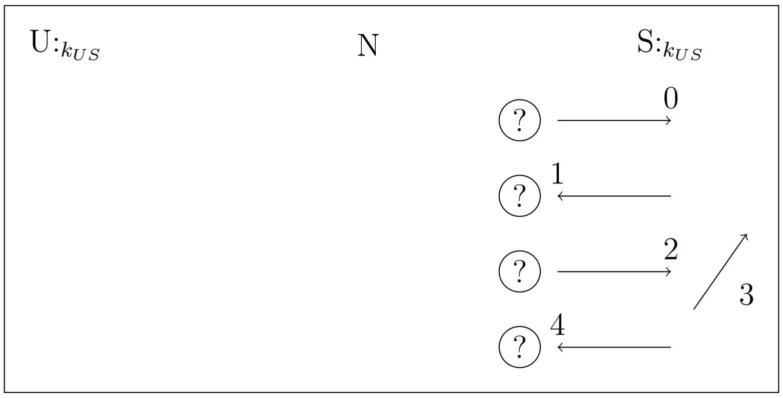

Figure 3.27: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 7.

Clearly, an adversary will not be able to start an execution in Step 2 of Figure 3.27. An attempt at this would simply be ignored since the appropriate record containing sid would not exist. Thus, there are only two possible executions that may take place here. The adversary will have to send (sid, T, n_1) to S in Step 0 of Figure 3.27. S

would then send back (sid, c, k). The adversary could chose to stop the execution here, or he could send (sid, n_2) to S. In the latter case S would accept and send (sid, ok) to $\mathcal{A}$. Both of these executions are covered by the simulator. We thus conclude that the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 7.

3.3.8 Class 8

In Class 8 only N is honest.

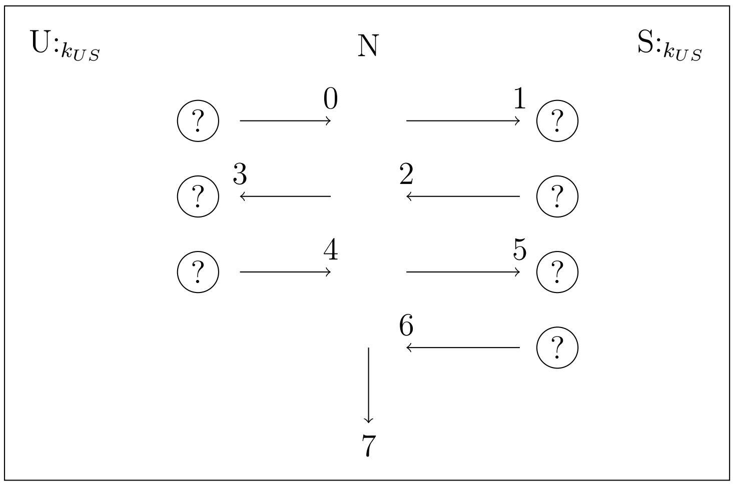

Figure 3.28: Track 1 of the anonymous symmetric key establishment protocol π_{SKA} for Class 8.

The only entry point in Class 8 is Step 0 of Figure 3.28. Let us look at what might happen when $\mathcal{A}$ starts a session. $\mathcal{A}$ sends (sid, T, n_1, S) to N. N will then send (sid, T, n_1) to $\mathcal{A}$. $\mathcal{A}$ could now choose not to answer and the protocol would stop here. However, if he answers the only response that will not be ignored is (sid, bit string, key) upon which N would send (sid, bit string, $\mathcal{E}$(key, pos, bit string) to $\mathcal{A}$. Again, $\mathcal{A}$ could chose to not answer and the protocol would stop. If he does answer the only reply that will be accepted is (sid, nonce, $\mathcal{E}$(key, e, new key)). Upon receiving this N will store the new key and send (sid, nonce) to $\mathcal{A}$. Also here $\mathcal{A}$ could chose to end the execution. However, does he answer the only message that will not be ignored is (sid, ok). When receiving this N will accept and output (**Est.**, new key, pos, S).

All of the possibilities above are parts of the track shown in Figure 3.28, namely Track 1. Track 1 is consistent with the simulator. Hence, the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 8 as well. We have now shown that the protocol

π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for all the classes. We thus conclude that the protocol π_{SKA} realizes the ideal functionality $\mathcal{F}_{KE}$.

Chapter 4

The Asymmetric Key Agreement Protocol

We will now go on to look at a similar protocol that uses an asymmetric encryption function instead of tokens. The main difference between this protocol and the previous one is that this protocol does not make it possible to link different attempts at starting a session to one another. We will start by describing the protocol in Section 4.1. We then perform a security analysis of the protocol in Section 4.3

4.1 The Protocol

When describing this protocol we will use ek_S to represent the public encryption key belonging to S, dk_S to mean the decryption key belonging to S and as before, k_{US} to mean a shared secret key between U and S. We would also like to note $e = (sid,$ pos, n_2, c_0, c_1, k, N, S, $\mu_1)$, where $\mu_1 = \mathcal{E}(k;\, pos,\, c_1;\,)$.

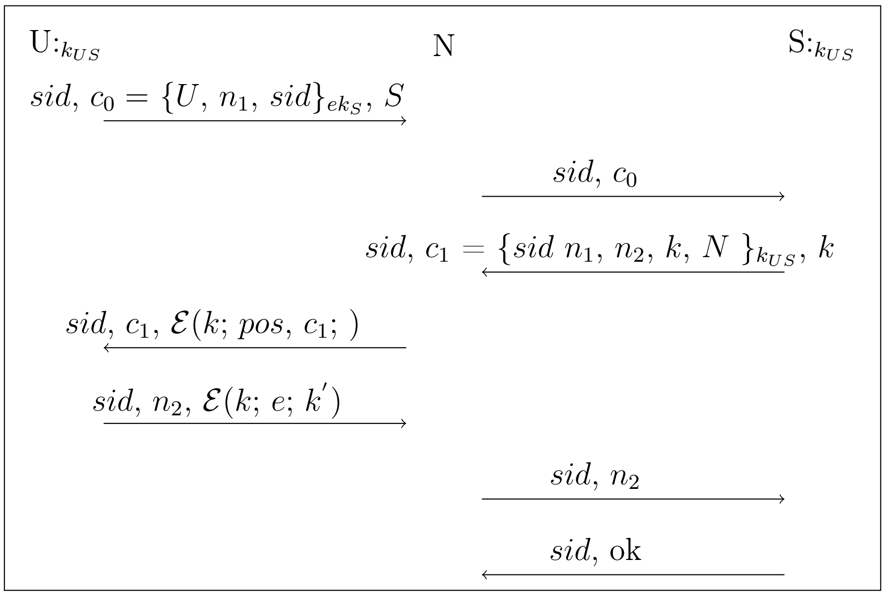
U:k_{US}
N
S:k_{US}
$sid,\, c_0 = \{U,\, n_1,\, sid\}_{ek_S},\, S$
$sid,\, c_0$
$sid,\, c_1 = \{sid\ n_1,\, n_2,\, k,\, N\ \}_{k_{US}},\, k$
$sid,\, c_1,\, \mathcal{E}(k;\, pos,\, c_1;\,)$
$sid,\, n_2,\, \mathcal{E}(k;\, e;\, k')$
$sid,\, n_2$
sid, ok

Figure 4.1: Summary of the anonymous symmetric key establishment protocol π_{AKE}. Communication between users and MNOs is via $\mathcal{F}_{RL}$ and between MNOs and SPs via $\mathcal{F}_{sec}$.

One drawback of this protocol is that since ek_S is public an adversary is able to construct c_0. Public key decryption is a relatively expensive operation and it is not optimal that the adversary is able to send any number of entries to S to decrypt.

```
**Stored**: $S$, $k_{US}$, $N$, $k^{'}$
On (**Enter**, $pos$, $N$) from $\mathcal{Z}$:
    1. If nothing is recorded then generate random nonce $n_1$ and session
         identifier $sid$, let $c_0 = \{U, n_1, sid\}_{ek(S)}$ record $(sid, pos, N, n_1)$ and
         send (**Enter**, $pos$) and (**Send**, $sid$, $(sid, c_0, S)$, $pos$, $N$) to $\mathcal{F}_{RL}$.
On (**Leave**, $pos$) from $\mathcal{Z}$:
    1. Send (**Leave**, $pos$) to $\mathcal{F}_{RL}$.
    2. If a record with $pos$ exists remove it and output (**Est**. **failed**).
On (**Recv**, $sid$, $m$, $pos$) from $\mathcal{F}_{RL}$:
    1. If $m = (sid, c, \mu_1)$ where $\mathbf{Dec}_{k_{US}}(c_1) = (sid, n_1, n_2, k, N)$,
         $\mu_1 = \mathcal{E}(k; pos, c_1; )$ and $(sid, pos, N, n_1)$ is recorded, replace it
         by $(sid, pos, N, n_1, n_2, c_1, \mu_1)$, set $\mu_2 \leftarrow \mathcal{E}(k; e; k^{'})$,
         output (**Est**., $k^{'}$, $pos$, $N$) and send
         (**Send**, $sid$, $(sid, n_2, \mu_2)$, $pos$, $N$) to $\mathcal{F}_{RL}$.
    2. Else do as second point of (**Leave**, $pos$) above.
```

Figure 4.2: The anonymous key establishment protocol π_{AKE}, Part 1: User.

```
**Stored**:
On (**Recv**, $sid$, $m$, $pos$) from $\mathcal{F}_{RL}$:
    1. If $m = (sid,\ c_0,\ S)$ and $sid$ is not on any record, record
       $(sid,\ pos,\ S,\ c_0)$ and send (**Send**, $sid$, $(sid,\ c_0)$, $S$) to $\mathcal{F}_{sec}$.
    2. Else if $m = (sid,\ n_2,\ \mu_2)$ and $(sid,\ pos,\ S,\ c_1,\ \mu_1,\ k)$ is
       recorded check that $\mathcal{D}(k;\ e;\ \mu_2) \neq \perp$, set
       $k' \leftarrow \mathcal{D}(k;\ e;\ \mu_2)$, replace the previous record by
       $(sid,\ c_0,\ pos,\ S,\ n_2,\ c_1,\ \mu_1,\ \mu_2,\ k,\ k')$ and send
       (**Send**, $sid$, $(sid,\ n_2)$, $S$) to $\mathcal{F}_{sec}$.
On (**Recv**, $sid$, $m$, $pos$) from $\mathcal{F}_{sec}$:
    1. If $m = (sid,\ c_1,\ k)$ and there is a record $(sid,\ pos,\ S,\ c_0)$, set
       $\mu_1 \leftarrow \mathcal{E}(k;\ pos,\ c_1;\ )$, change the record to
       $(sid,\ pos,\ S,\ c_1,\ \mu_1,\ k)$ and send (**Send**, $sid$, $(sid,\ c_1,\ \mu_1)$, $S$)
       to $\mathcal{F}_{RL}$.
    2. Else if $m = (sid,\ \text{ok})$ and there is a record
       $(sid,\ c_0,\ pos,\ S,\ n_2,\ c_1,\ \mu_1,\ \mu_2,\ k,\ k')$ then remove it and output
       (**Est.**, $k'$, $pos$, $S$).
```

Figure 4.3: The anonymous key establishment protocol π_{AKE}, Part 2: MNO.

```
**Stored**: $\{U, k_{US}\}$
On (**Recv**, $sid$, $m$, $N$) from $\mathcal{F}_{sec}$:
    1. If $m = (sid,\ c_0)$ and $sid$ is not on any record, record $(sid,\ c_0,\ N)$,
       then generate random nonce $n_2$ and set
       $c_1 \leftarrow \mathbf{Enc}_{k_{US}}(sid, n_1, n_2, k, N)$, record $(sid,\ n_2,\ U,\ N)$ and send
       (**Send**, $sid$, $(sid,\ c_1,\ k)$, $N$) to $\mathcal{F}_{RL}$.
    2. Else if $m = (sid,\ n_2)$ and $(sid,\ n_2,\ U,\ N)$ is recorded then remove it,
       send (**Send**, $sid$ $(sid,\ \text{ok})$, $N$) to $\mathcal{F}_{sec}$ and output (**Est.**, $U$, $N$).
```

Figure 4.4: The anonymous key establishment protocol π_{AKE}, Part 3: SP.

4.2 The Simulator

The simulator for this protocol will be very similar to the one provided for the SKA protocol. We have chosen not to construct this simulator. It will however be easy to create if basing it on the diagrams accompanying the simulator described in Section 3.2. Hence, when performing the security analysis in Section 4.3 we simply state whether an execution is possible to simulate.

4.3 Security Analysis

We will now have a look at the security properties of this protocol. We will do an analysis that is equivalent to the one performed for the SKA protocol in Section 3.3. We will again look at the classes (excluding 0 and 9) in ascending order. For each class we will describe all possible executions, and state that these are possible to simulate. Thus, showing that the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$. The AKA protocol is very similar to the SKA protocol, and as such we have not included diagrams showing the different tracks as we did for the SKA protocol. This is as the diagrams will look the exact same as those provided in Section 3.3. Hence, the reader will simply be referred to the diagrams in Section 3.3.

As opposed to what happens in π_{SKA}, a message received together with a AEAD that is wrong will simply be ignored by the recipient in this protocol. This should be kept in mind when reading the security analysis.

4.3.1 Class 1

We will start with Class 1. There are two possible tracks that the protocol might follow in this class. The first track is shown in Figure 3.15. If the protocol follows this track one possibility is that N accepts. Let us look at what guarantees we have when N accepts.

N accepts:

- N received (sid, ok) from S for a record where he already sent (sid, n_2) to S.
- Since S sent (sid, ok) to N, S must have accepted.

S accepts:

- S received (sid, n_2) from N for a record where he already sent (sid, c_1, k) to N.

- N received (sid, n_2, $\mathcal{E}(k;\ e;\ k^{'})$) from $\mathcal{A}/pos$ and N accepted it.
- Due to RoR-CCA only S and U know n_2 which is encrypted using k_{US} as apart of c_1.
- U sent n_2.
- U accepted.

U accepts:

- U received (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$), where the decryption of c_1 matches a record, from $\mathcal{A}/pos$ for a record where he already sent (sid, c_0, S) to $\mathcal{A}/pos$.
- Only S and U know k_{US}, and hence due to INT-CTXT c_1 is made by S.
- $\exists$ a session on S with sid, n_1, n_2, k, N like those in c_1.
- At this point only U, N and S know k, and thus due to INT-CTXT N made $\mathcal{E}(k;\ pos,\ c_1;\ )$.
- N received (sid, c_1, k) from S.
- S received (sid, c_0) from N.
- N received (sid, c_0, S) from $\mathcal{A}/pos$.
- Since U accepted he agrees with S on wanting to start a session with the variables contained in c_1.
- Only U and N know $k^{'}$.

All of this must have happened in this exact way if N accepts. Between S and N accepting, nothing could have gone wrong, and thus, as long as S accepts, we also find ourselves in Track 1. The protocol will for Class 1 only ever abort in Step 9 of Figure 3.15.

U aborts in Step 9 in Figure 3.15:

- U received a (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) from $\mathcal{A}/pos$ where the decryption of c_1 using k_{US} is not on the form $\{sid,\ n_1,\ \text{-},\ \text{-},\ N\}$
- There are two things that might have happened here. Either $\mathcal{A}$ sent something to N in Step 1 of Figure 3.15 or he sent (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) to U directly as shown in Figure 3.16.
- Let us start by looking at what happens if he sends something to N:
- $\mathcal{A}$ must have sent (sid, c_0, S) to N where $c_0 = \{U,\ n_1,\ sid\}_{ek_S}$. Since ek_S is a public key anyone can make a c_0. If sid was not correct the message would just have been ignored. If c_0 was wrong the message would have been ignored in Step 4 of the protocol.
- N received (sid, c_0, S) from $\mathcal{A}/pos$ and sent (sid, c_0) to S.
- S sent (sid, c_1, k) to N.
- N sent (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) to $\mathcal{A}/pos$.

When receiving (sid, c_0, S) from U, $\mathcal{A}$ could chose to send (sid, bit string, bit string) to U. The way the protocol is written U decrypts c_1 before even looking at the AEAD. Thus, when c_1 turned out to be wrong U would simply output **Est**. **Failed**. This is depicted in Figure 3.16. However, $\mathcal{A}$ will at any point during an execution that starts in Track 1 in Class 1 be able to send something to U to make U abort. This will still be a part of Track 1.

As for the SKA protocol, the way this protocol is written it is also possible for an adversary present at pos to send (sid, c_0, S) to N without having received it from U first. The only thing he has to make sure of is that the sid he sends does not already exist on any record. This is depicted in Figure 3.17 as Track 2.

$\mathcal{A}$ sends (sid, c_0, S) to N as shown in Figure 3.17:

- N sends (sid, c_0) to S.
- S receives it and sends (sid, c_1, k) to N.
- N sends (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) to $\mathcal{A}$.

Since sid does not exist on any of U's records, $\mathcal{A}$ will in this instance never be able to send anything to U in step 8 of Figure 3.15 that U will accept. Thus, in this track $\mathcal{A}$ will just send (sid, c_0, S) to N and the receive (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) some time after he sends the first message. This is all that ever will happen in Track 2.

Here as for the SKA protocol it is also a possibility that the protocol starts its execution in Track 1 only to have U receive (sid, bit string, bit string) from $\mathcal{A}$. The execution will then split up so that part of it becomes equivalent with the part of Track 1 that is shown in Figure 3.16, and the other part of it becomes equivalent to Track 2 shown in Figure 3.17.

We have now described all the scenario that might take place in Class 1. All of the above scenarios can be simulated, and hence, the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 1.

4.3.2 Class 2

In Class 2 N and pos are corrupt. Let us start by looking at what guarantees we have if S accepts:

S accepts:

- S sends (sid, ok) to $\mathcal{A}$.
- S must have received (sid, n_2) from $\mathcal{A}$ for a record where he already sent (sid, c_1, k) to $\mathcal{A}$.

- $\mathcal{A}$ does not have access to k_{US} and thus can not decrypt c. Due to RoR-CCA it will therefore not know n_2.
- U sent n_2.
- U must have accepted.

U accepts:

- U must have received a (sid, c_1, $\mathcal{E}(k$; pos, c_1;)) from $\mathcal{A}$ for a record where he already sent (sid, c_0, S) to $\mathcal{A}$.
- Due to INT-CTXT S must have created c_1. Further, n_1 is decrypted under ek_S and only S can decrypt it. n_1 must have been a part of c_1. Due to RoR-CCA N can not see n_1 and thus, S must have received (sid, c_0).
- S and U agree on all the entries in c_1.
- S must have sent (sid, c_1, k) to $\mathcal{A}$.

Everything must have happened exactly as described above if S accepts. It is easy to see that this execution follows Track 1. For Class 2 the protocol will only ever abort in Step 5 of Figure 3.18. Let us see what we know when the protocol outputs **Est. Failed** in Step 5 of Figure 3.18.

U aborts in Step 5 of Figure 3.18:

- U received a (sid, c_1, $\mathcal{E}(k$; pos, c_1;)) where c_1 was wrong.
- U must have sent (sid, c_0, S) to $\mathcal{A}$ or the message would have been ignored.

There are now two options. Either, $\mathcal{A}$ sent something to S, or not. Let us start by looking at what must have happened if $\mathcal{A}$ sent something to S;

S must have received (sid, c_0). c_0 must be on the form $\{U, n_1, sid\}_{ek_S}$ as S would have just ignored the message otherwise. Further, S must have sent (sid, c_1, k) to $\mathcal{A}$.

If this is the case we are again only operating in a smaller part of Track 1. Track 1 can be simulated. As we stated in Section 3.3.1, $\mathcal{A}$ will be free to send a message to U at any point after Step 1 and before Step 4. This will cause U to abort, and will split up the execution so that part of it follows the part of Track 1 that is shown in Figure 3.19, and the other part Track 2 which is introduced below.

The protocol does not require for sid to be stored when S receives (sid, c_0). As such it would be possible for $\mathcal{A}$ to send (sid, c_0) to S without U having attempted to start a session. This is shown in Figure 3.20 and we have named it Track 2. S will have to send a c_0 that is on the correct format to S so that the message is not simply ignored. This instance can also be simulated.

The two tracks described above are the only possible tracks for Class 2. Since both of these can be simulated, the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ within Class 2.

4.3.3 Class 3

In Class 3 only S is corrupt. The only track that exists for Class 3 is Track 1. Track 1 is depicted in Figure 3.21. Let us start by looking at what guarantees we have if N accepts.

N accepts:

- N must have received (sid, ok) from $\mathcal{A}$ for a record where he already sent (sid, n_2) to $\mathcal{A}$.
- Since N sent (sid, n_2) to $\mathcal{A}$, he must have received (sid, n_2, $\mathcal{E}(k;\ e;\ k')$) from U.
- U accepted.
- N and U will output the same k' as no one will have been able to change it between it leaving U and reaching N. They will also be the only parties who know k'.

U accepts:

- U received a (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) from N that it accepted.
- U must have sent (sid, c_0, S) to N or it would not have accepted.
- Since N sent (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) to U, N must have received (sid, c_1, k) from $\mathcal{A}$ for a record where he already sent (sid, c_0) to $\mathcal{A}$.
- Since N sent (sid, c_0) to $\mathcal{A}$ he must have received (sid, c_0, S) from U.

Things must have happened exactly as described above when N accepts. We can see that this is consistent with Track 1. It is possible to simulate Track 1 and everything is okay so far.

The only way the protocol can fail in Class 3 is if $\mathcal{A}$ sends (sid, c_1, k) where c_1 is wrong to N. Any other input from $\mathcal{A}$ that does not lead to N accepting is ignored. The message will then go on until U receives it in Step 7 of Figure 3.21. U will then send (**Deny**, id) to $\mathcal{F}_{KE}$.

U aborts in Step 7 of Figure 3.21:

- U has received a (sid, c_1, $\mathcal{E}(k;\ pos,\ c_1;\ )$) from N that is wrong. That is, the decryption of c_1 is not on the format (sid, n_1, -, -, N).

- Since N sent $(sid, c_1, \mathcal{E}(k; pos, c_1;))$ to U, N must have received (sid, c_1, k) from $\mathcal{A}$.
- Since N did not just ignore the message from $\mathcal{A}$, N must have sent (sid, c_0) to $\mathcal{A}$.
- Since N sent the above to $\mathcal{A}$, N must have received (sid, c_0, S) from U.

We can see that this as well is simply a part of Track 1, and hence can be simulated. Thus, the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ also for Class 3.

4.3.4 Class 4

In Class 4 both S and pos are corrupt. This gives the adversary an advantage. Clearly, the protocol may execute in a normal manner and follow the path depicted in Figure 3.22. We have chosen to call this path Track 1. Let us go on to have a look at what might happen when N accepts.

N accepts:

- N received (sid, ok) from S for a sid where he had already sent (sid, n_2) to $\mathcal{A}$.
- N must have received $(sid, n_2, \mathcal{E}(k; e; k'))$ from $\mathcal{A}$ for a record where he already sent $(sid, c_1, \mathcal{E}(k; pos, c_1;))$ to $\mathcal{A}$.
- N must have received (sid, c_1, k) from $\mathcal{A}$ for a record where he already sent (sid, c_0) to $\mathcal{A}$.
- N must have received(sid, c_0, S) from $\mathcal{A}$.
- U has not necessarily been involved in this execution at all.

We can easily see that only N and $\mathcal{A}$ need to be present for N to accept. This is depicted in Figure 3.23 and we have named the track the protocol follows Track 2. Let us however assume that U has in fact been present.

N accepts with U present:

- N received (sid, ok) from $\mathcal{A}$ for a sid where he had already sent (sid, n_2) to $\mathcal{A}$.
- N must have received $(sid, n_2, \mathcal{E}(k; e; k'))$ from $\mathcal{A}$.
- Since pos is collaborating with S they know k, and $\mathcal{A}/pos$ would have been able to alter this AEAD, and so this AEAD is not necessarily the same one U sent to $\mathcal{A}/pos$.
- N and U might not have the same value for k'.
- Neither N nor U will know that pos and S were dishonest.
- N and U will output different values for k' whilst thinking they agree on k'.

As long as both N and U are present during an execution, the execution will follow Track 1. It is possible to simulate Track 1.

We will now go on to see what guarantees we have if U accepts.

U accepts:

- U received $(sid,\, c_1,\, \mathcal{E}(k;\, pos,\, c_1;\,))$ from $\mathcal{A}$ where both c_1 and the AEAD were correct.
- U sent $(sid,\, c_0,\, S)$ to $\mathcal{A}$, or else he would not have accepted.

This is all we know when U accepts. Thus, N does not need to be involved in order for $\mathcal{A}$ to get U to accept. If this is the case the execution will follow the path shown in Figure 3.24. This path is as discussed earlier still a part of Track 1.

In Class 4 the protocol has 2 starting points, Step 0 and Step 2 of Figure 3.22. If it starts in Step 0 it will follow Track 1. If it starts in Step 2 it will follow Track 2. As we have talked about in the previous sections, an execution could just as well start out following Track 1 and later on split so that part of the execution runs as shown in Figure 3.23 and part of it as shown in Figure 3.24. This can be simulated.

Let us now go on to look at what happens when U outputs (**Est**. **Failed**) in Step 7 of Figure 3.22.

U aborts in Step 7 of Figure 3.22:

- U received a $(sid,\, c,\, \mathcal{E}(k;\, pos,\, c;\,))$ from $\mathcal{A}$ where c or the AEAD was wrong.
- Since the above message was not ignored U must have sent $(sid,\, c_0,\, S)$ to $\mathcal{A}$.
- This is all we know. We can see that this execution is consistent with the part of Track 1 that is depicted in Figure 3.24. However, N might be involved in these executions. If that is the case the execution will be consistent with Track 1 as shown in Figure 3.22. Also here the execution might split up into one execution following Track 1 and one execution following Track 2. As discussed this is not problematic. Thus, all possible executions leading to U aborting can be simulated.

As before, we can classify all the executions of the protocol for Class 4, and all possibilities can be simulated. Hence, the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 4.

4.3.5 Class 5

In Class 5 only U is honest. Thus, the only possible track is depicted in Figure 3.25 and will be referred to as Track 1. Let us start by having a look at what happens when U accepts.

U accepts:

- U received a $(sid,\ c_1,\ \mathcal{E}(k;\ pos,\ c_1;\ ))$ from $\mathcal{A}$ that U accepted.
- U sent $(sid,\ c_0,\ S)$ to $\mathcal{A}$ or he would not have accepted.
- U will output a $k^{'}$ that $\mathcal{A}$ will be able to alter.
- U will send $(sid,\ n_2,\ \mathcal{E}(k;\ e;\ k^{'}))$ to $\mathcal{A}$.

This execution clearly follows Track 1 and is covered by the simulator. Let us now see what happens if U outputs (**Est**. **Failed**) in Step 3 of Figure 3.25.

U aborts in Step 3 of Figure 3.25:

- U received $(sid,\ c_1,\ \mathcal{E}(k;\ pos,\ c_1;\ ))$ where c_1 was correct, but the AEAD was wrong.
- Since the message was not ignored U must have sent $(sid,\ c_0,\ S)$ to $\mathcal{A}$.
- This execution is covered by Track 1.

U outputs (**Est**. **Failed Linkable**) in Step 3 of Figure 3.25:

- U received $(sid,\ c_1,\ \mathcal{E}(k;\ pos,\ c_1;\ ))$ where c_1 was wrong.
- Since the message was not ignored U must have sent $(sid,\ c_0,\ S)$ to $\mathcal{A}$.
- This execution is covered by Track 1.

Track 1 can be simulated. Hence, we can simulate all executions that might occur, and the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ in Class 5.

4.3.6 Class 6

In Class 6 only U is corrupt. We will show that the only exists one possible path an execution may follow in Class 6. We have named this path Track 1 and it is depicted in Figure 3.26. Let us start by having a look at what happens if N accepts.

N accepts:

- N received $(sid,\ ok)$ from S for a record where he already sent $(sid,\ n_2)$ to S.

- S accepted.

S accepts:

- S received (sid, n_2) from N for a record where he already sent (sid, c_1, k) to N.
- Since N sent (sid, n_2) to S, N must have received (sid, n_2, $\mathcal{E}(k; e; k^{'})$) from $\mathcal{A}$.
- N must have sent (sid, c_1, $\mathcal{E}(k; pos, c_1;)$) to $\mathcal{A}$ or he would not have sent (sid, n_2) to S when he received (sid, n_2, $\mathcal{E}(k; e; k^{'})$) from $\mathcal{A}$.
- N must have received (sid, c_1, k) from S.
- S must have received (sid, c_0) from N.
- N must have received (sid, c_0, S) from $\mathcal{A}$.

This execution follows Track 1 and as such can be simulated. $\mathcal{A}$ will never be able to send a wrong message to N that is not ignored. He could chose to stop after he receives the message from N in Step 5 of Figure 3.26. However, this simply leaves us in the top part of Track 1. Thus this class as mentioned only has 1 track and the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ also for Class 6.

4.3.7 Class 7

In Class 7 only S is honest. An adversary will not be able to start an execution in Step 2 of Figure 3.27. An attempt at this would simply be ignored since the appropriate record containing sid would not exist. Thus, there are only two possible executions that may take place here. The adversary will have to send (sid, c_0) to S in Step 0 of Figure 3.27 to start an execution. S would then send back (sid, c_1, k). The adversary could then choose to stop the execution by not sending anything to $\mathcal{S}$. Alternatively he could send (sid, n_2) to S. In the latter case S would accept and send (sid, ok) to $\mathcal{A}$. Both of these executions can be simulated. We thus conclude that the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 7.

4.3.8 Class 8

In Class 8 only N is honest. Thus, the only entry point in Class 8 is Step 0 of Figure 3.28. Let us look at what might happen when $\mathcal{A}$ starts a session. $\mathcal{A}$ sends (sid, c_0, S) to N. N will then send (sid, c_0) to $\mathcal{A}$. $\mathcal{A}$ could now choose not to answer and the protocol would stop here. However, if he answers the only response that will not be ignored is (sid, bit string, key) upon which N would send (sid, bit string, $\mathcal{E}$(key, pos, bit string) to $\mathcal{A}$. Again, $\mathcal{A}$ could chose to not answer and the protocol would stop. If he does answer the only reply that will be accepted is (sid, nonce, $\mathcal{E}$(key, e, new key)). Upon receiving this N will store the new key and send (sid, nonce) to $\mathcal{A}$. Also here $\mathcal{A}$ could chose to end the execution. If he answers the only message that will not be ignored is (sid, ok). When receiving this N will accept and output (**Est**., new key, pos, S).

All of the possibilities above are parts of the track shown in Figure 3.28, namely Track 1. Track 1 can be simulated. Hence, the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 8 as well. We have now concluded that the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$ for all the classes. We thus conclude that the protocol π_{AKA} realizes the ideal functionality $\mathcal{F}_{KE}$.

Chapter 5

Summary of the Token Key Agreement Protocol

Today, most networks identify users using their IMSIs. Thus, a user does not remain anonymous. It is suggested in [GPS12] that TMSIs, temporary identities generated by the MNO, should be introduced in order to help a user obtain anonymity. This chapter only provides a summary of the mentioned protocol in order to be able to perform comparisons. A more detailed description of the protocol can be found in [GPS12].

It should be mentioned that the simulator in the paper describing the full TAP protocol looks very different to the simulator we have introduced for our Symmetric Key Agreement protocol in Section 3.2. This is as we do not have a maximum page limit, and thus, have attempted to create a systematic simulator whilst not concerning ourselves with the simulators length. Furthermore, the work done in this paper concerning the TAP protocol is based on a draft of the paper [GPS12] dating from 11 of January 2012. The article has now been revised and adjustments made. These adjustments are not included in this paper.

5.1 The Protocol

Figure 5.1 provides a summary of the anonymous key establishment protocol π_{KE}. The complete protocol can as mentioned be found in [GPS12] along with the original ideal functionality and the simulator. We have as mentioned changed the ideal functionality and described it in terms of a *leak* function (Figure 2.7). This is done in order to easier be able to compare this protocol to the SKA and AKA protocols. Our new ideal functionality is as mentioned described in Figure 2.5

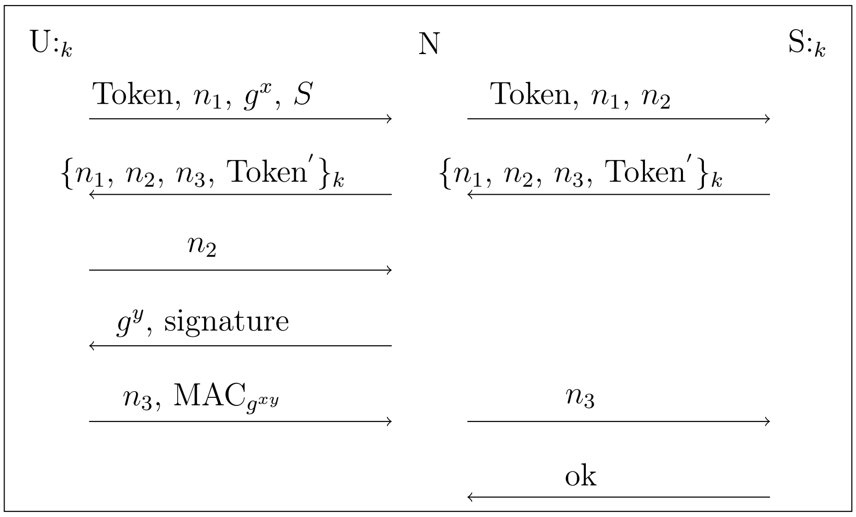

Figure 5.1: Summary of the anonymous key establishment protocol π_{KE}. Communication between users and MNOs is via $\mathcal{F}_{RL}$ and between MNOs and SPs via $\mathcal{F}_{sec}$.

5.2 Security Analysis

We have decided to try and perform a simple discussion of the security of this protocol for some of its classes. We will go through the classes in ascending order. We only perform the security analysis for the first four classes (excluding Class 0 as this is trivially secure). The remaining classes will only have 1 track each and the security analysis for these should be performed in the same manner as the rest. Due to a lack of time they are left to the reader.

In the security analysis we will use the Decisional Diffie-Hellman assumption. We will refer to it as DDH. The Diffie-Hellman assumption assumes the Decisional Diffie-Hellman problem to be hard. The Diffie-Hellman problem is briefly described *given g, g^x and g^y, find g^{xy}.* In the problem g is a generator of some group, usually a multiplicative group, a finite field or an elliptic curve group. x and y are random integers. The easiest known way to solve this problem is to solve the discrete logarithm problem (DLP), *find x given g^x.* There is no proof that this is the easiest way to solve the problem. However, there is also no proof that there is an easier way to solve it. Common belief is that the DHP is as hard as the DLP.

We further assume that any interested party has access to group $\mathbb{G}$, and thus, any interested party is able to create a $g^y \epsilon \mathbb{G}$ for a random $y \epsilon \mathbb{Z}$. We could chose to keep the group $\mathbb{G}$ secret, but information easily leaks, and there is not much point to this.

Before we proceed with the security analysis we will introduce some of the variables we use:

- $\mu = \text{MAC}_{g^{xy}}$.
- $c = \{n_1, n_2, n_3, T'\}_k$ where k is the secret key shared by U and S.
- $\sigma =$ A signature created by N containing all the variable known by N such that if σ is correct $Ver_{vk_N}((sid, n_1, n_2, g^x, g^y, T, c, pos, S), \sigma) = \text{True}$.
- $h \epsilon \mathbb{G}$.

We would also like to point out that in the newer version of [GPS12] *sid* is sent with every message of the protocol. We have not written this in the protocol summary of Figure 5.1 or in the security analysis that follows. However, it is assumed that all parties will know what *sid* a message belongs to.

5.2.1 Class 1

In Class 1 only *pos* is corrupt. There are two possible tracks that the protocol might follow in this class. The first track is shown in Figure 5.2.

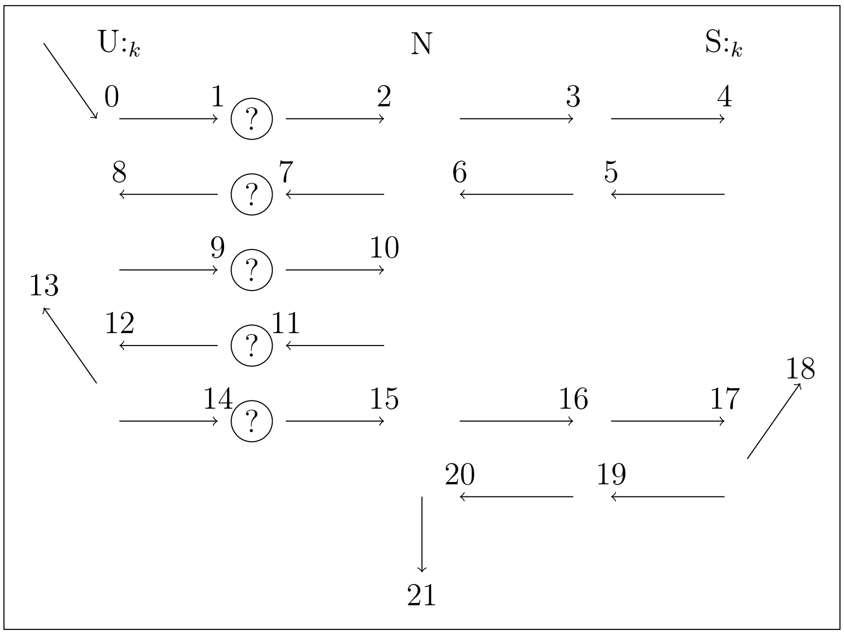

Figure 5.2: Track 1 of the anonymous key establishment protocol π_{TAP} for Class 1.

Let us start by looking at what happens when N accepts.

N accepts:

- N received (ok) from S for a record where he already sent (n_3) to S.
- Since S sent (ok) to N, S accepted.

S accepts:

- S received (n_3) from N for a record where he already sent $c = \{n_1, n_2, n_3, T^{'}\}_k$ to N.
- N received $(n_3,\ \mu)$ from $\mathcal{A}/pos$ for a record where he already sent $(g^y,\ \sigma)$ to $\mathcal{A}/pos$.
- n_3 is a part of c and only S and U know k and are able to decrypt c. Thus, due to RoR-CCA n_3 came from U.
- U sent $(n_3,\ \mu)$ to $\mathcal{A}/pos$.
- U accepted.

U accepts:

- U received $(g^y,\ \sigma)$ from $\mathcal{A}/pos$ for a record where he already sent (n_2) to $\mathcal{A}/pos$.

- Due to INT-CTXT σ must have been created by N and thus N sent (g^y, σ) to $\mathcal{A}/pos$. Since g^y is included in σ, $\mathcal{A}$ will not have been able to change g^y.
- Since U sent (n_2) to $\mathcal{A}/pos$, U must have received c from $\mathcal{A}/pos$.
- Due to INT-CTXT c must have been created by S.
- N received (c) from S and passed it on to $\mathcal{A}/pos$.
- Since S sent (c) to N, S must have received (T, n_1, n_2) from N.
- N received (T, n_1, g^x, S) from $\mathcal{A}/pos$.
- Since U accepted, the entry above received by N must have been sent to $\mathcal{A}/pos$ by U. If T was not the same S would not know who U was. If n_1 was not the same U would have aborted after receiving c in step 8 of Figure 5.2. If g^x was not the same N would have ignored the message he received in Step 15 of the same figure. If S was not the same S, he would not know U.
- Only U and N know k'.

If N accepts all of this must have happened in this exact way. Between N and S accepting the adversary is not involved and as long as S accepts everything will be the same. This execution clearly follows Track 1 shown in Figure 5.2. This track can be simulated by the simulator $\mathcal{S}_{KE}$ found in [GPS12].

Let us go on to have a look at what happens if the protocol aborts in Steps 8 or 13 of Figure 5.2. We will look for ways in which an adversary can gain an advantage from making the protocol abort on purpose.

The protocol outputs **Est**. **Failed Linkable** in Step 8 of Figure 5.2:

- U received a (c) from $\mathcal{A}$ that did not decrypt to $\{n_1$, -, -, -$\}$.
- Since U did not simply ignore (c) he must have sent (T, n_1, g^x, S) to $\mathcal{A}$.
- This is all we know when the protocol aborts in Step 8 of the protocol. Thus, there are two possible scenarios. Either $\mathcal{A}$ sent something to N before he sent the wrong c to U or he did not.
- If $\mathcal{A}$ sent something to N he must have sent (Token, nonce, h, S) or else the message would simply be ignored in step 4 of Figure 5.2.
- If he did not send anything to N, but simply proceeded by sending a bit string to U, only Steps 0, 1 and 8 would be parts of the execution.

Both of these scenarios are part of Track 1 and as such are consistent with the simulator $\mathcal{S}_{KE}$. Another possibility is for the protocol to start of in Track 1 only to have $\mathcal{A}$ send a message to U before the execution reaches Step 7. In such a case the execution will simply split up into two tracks, namely Track 1 and Track 2 which is shown in Figure 5.3. We will talk more about this track below.

Let us first look at what happens when the protocol aborts in Step 13 of Figure 5.2.

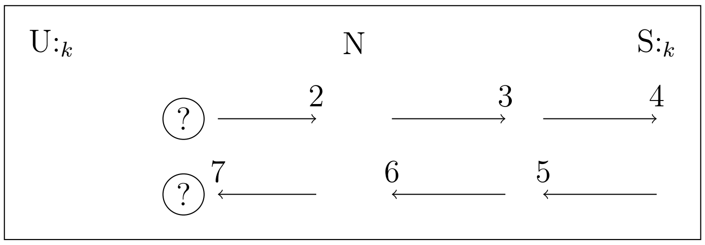

Figure 5.3: Track 2 of the anonymous key establishment protocol π_{TAP} for Class 1.

The protocol outputs **Est**. **Failed Unlinkable** in Step 13 of Figure 5.2:

- U received a (g^y, σ) where either g^y or σ is wrong.
- The protocol did not abort before this point so everything must have been okay until we reached Step 12.
- N received (n_2) from $\mathcal{A}$ where n_2 matched the one he sent to S in Step 3 of Figure 5.2.
- N sent (g^y, σ) to $\mathcal{A}$.
- Only U and S are able to decrypt c, and due to RoR-CCA $\mathcal{A}$ will not know n_2 unless U sends it to him.
- U must have received and decrypted c.
- Due to INT-CTXT c has been created by S.
- N must have sent (c) to $\mathcal{A}/pos$.
- N must have received (c) from S.
- S received (T, n_1, n_2) from N.
- N received (T, n_1, g^x, S) from $\mathcal{A}$.
- Since U accepted in Step 8, U sent (T, n_1, g^x, S) to $\mathcal{A}$. The only entry in this message $\mathcal{A}$ could have altered is g^x.

We can see that also this will be part of Track 1. Lastly, the way the protocol is written it is possible for $\mathcal{A}$ to start a session without having received anything from U. Let us look at how this will play out.

- $\mathcal{A}/pos$ sends (Token, nonce, h, S) to N.
- N sends (Token, nonce, n_2) to S.
- S sends (c) to N.
- N sends (c) to $\mathcal{A}/pos$.
- Any further messages sent to U by $\mathcal{A}/pos$ will simply be ignored by U since U does not have any record containing the correct *sid*.

This is what we earlier introduced as Track 2. It is easy to see that Track 2 never will lead anywhere. Further, it is consistent with the simulator $\mathcal{S}_{KE}$. As such we conclude that the protocol π_{KE} described in [GPS12] realizes the ideal functionality $\mathcal{F}_{KE}$.

5.2.2 Class 2

In Class 2 *pos* and N are corrupt. Class 2 also has 2 possible execution tracks. The first one is shown in Figure 5.4

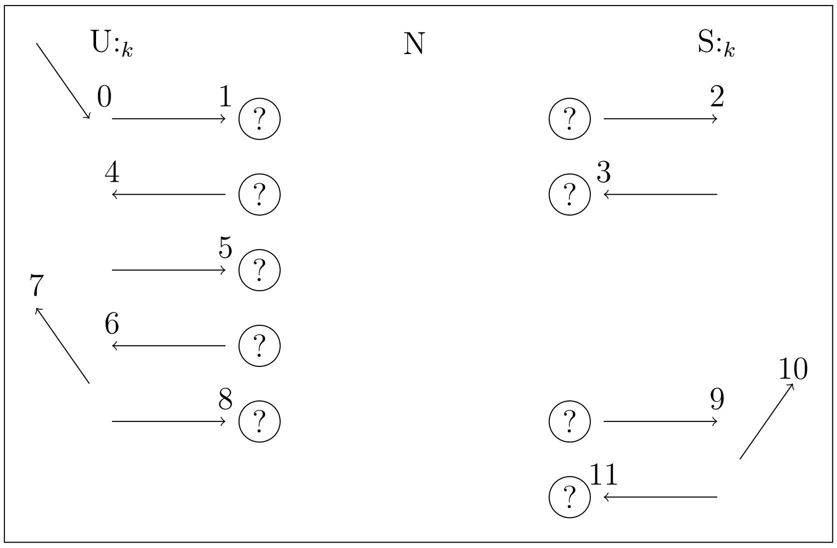

Figure 5.4: Track 1 of the anonymous key establishment protocol π_{KE} for Class 2.

Let us start by looking at what guarantees we have when S accepts.

S accepts:

- S received (n_3) from $\mathcal{A}$ for a record where he already sent (c) to $\mathcal{A}$.
- S will send (ok) to $\mathcal{A}$.
- Since S accepted n_3 he must have sent (c) to $\mathcal{A}$.
- S must have received (T, n_1, n_2) from $\mathcal{A}$ where *sid* was not on any record.
- Only S and U know k and are able to decrypt c.
- Due to RoR-CCA U must have decrypted c and sent (n_3, μ) to $\mathcal{A}$.
- U accepted.

U accepts:

- U must have received (g^y, σ) from $\mathcal{A}$ for a record where he already sent (n_2) to $\mathcal{A}$.

- Since U sent n_2 to $\mathcal{A}$ he must have received (c) from $\mathcal{A}$ containing the same n_1 as U sent to $\mathcal{A}$ in Step 1 of Figure 5.4.
- Due to INT-CTXT c must have been created by S.
- U must have sent (T, n_1, g^y, S) to $\mathcal{A}$.

Thus, as long as S accepts the execution follows Track 1. We can also see that as long as U accepts, but not S, the execution would still have had to follow Track 1 as S would have made c. All instances of Track 1 can be simulated by $\mathcal{S}_{KE}$. We will now go on to look at what happens if the protocol fails in Steps 4 and 7.

The protocol outputs **Est**. **Failed Linkable** in Step 4 of Figure 5.4:

- U received a (c) from $\mathcal{A}$ that did not decrypt to $\{n_1$, -, -, -$\}$.
- Since U did not ignore the message received in Step 4 he must have sent (T, n_1, h, S) to $\mathcal{A}$.
- This is all we know when the protocol aborts in Step 8 of the protocol.

However, as discussed in Section 3.3.1, this is still a part of Track 1.

The protocol outputs **Est**. **Failed Unlinkable** in Step 7 of Figure 5.4:

- U received a (g^y, σ) from $\mathcal{A}$ for a record where he already sent (n_2) to $\mathcal{A}$, where g^y or σ was wrong.
- Since U sent n_2 to $\mathcal{A}$, U must have received a (c) from $\mathcal{A}$ that he accepted.
- Due to INT-CTXT c must have been created by S.
- S must have received (T, n_1, n_2) from $\mathcal{A}$ and sent c to $\mathcal{A}$.
- Since U accepted c he must have sent (T, n_1, g^x, S) to $\mathcal{A}$.

Clearly this as well is a part of Track 1 and as such can be simulated.

We will now go on to look at a last possibility. It will be possible for $\mathcal{A}$ to start an execution without receiving anything from U first. If this happens we find ourselves in what we have named Track 2. Track 2 is portrayed in Figure 5.5.

When $\mathcal{A}$ sends (Token, nonce, nonce 2) to S, S will send back (c) as long as the token he receives is valid. However, this (c) will never be accepted by U as U does not have any record containing *sid*. Thus, the execution will stop in after Step 3 of Figure 5.5. This instance is consistent with the simulator $\mathcal{S}_{KE}$. Before concluding this section we wish to mention that an execution may start in Track 1 only to have $\mathcal{A}$ send something to U before the execution finishes Step 3. This again will simply lead to the execution splitting up so that one part follows Track 2 and one part follows Track 1 as in Class 1.

We thus conclude that the protocol π_{KE} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 2 as well.

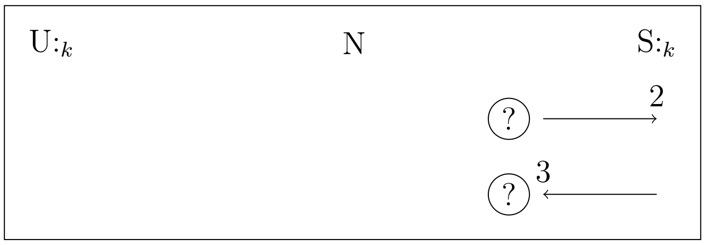

Figure 5.5: Track 2 of the anonymous key establishment protocol π_{KE} for Class 2.

5.2.3 Class 3

In Class 3 only S is corrupt. Thus, the executions in this class will only ever follow one track. We have named this track Track 1 and it can be found in Figure 5.6

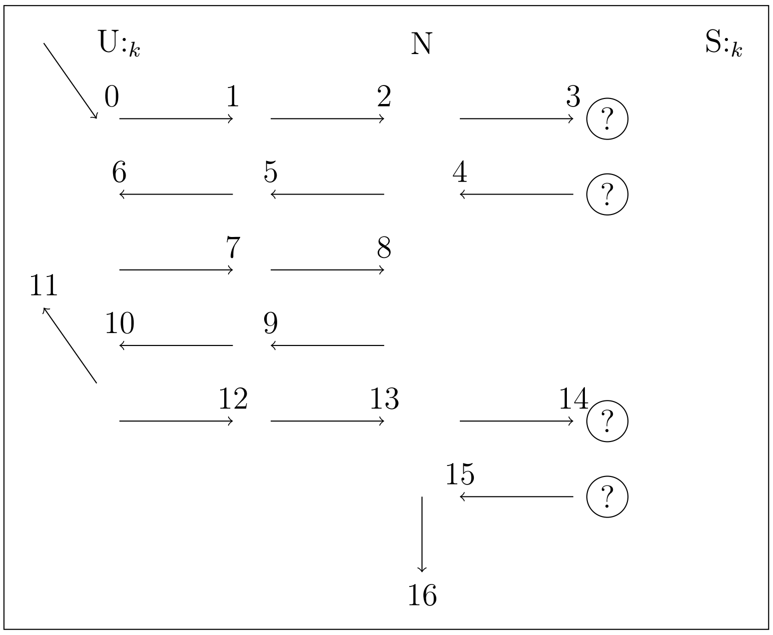

Figure 5.6: Track 1 of the anonymous key establishment protocol π_{KE} for Class 3.

We will again start by looking at what guarantees we have when N accepts:

N accepts:

- N received (ok) from $\mathcal{A}$ for a record where he sent (n_3) to $\mathcal{A}$.
- Since N sent (n_3) to $\mathcal{A}$ he must have received (n_3, μ) from U.
- Since U sent (n_3, μ) to N U must have accepted.

U accepts:

- U received (g^y, σ) from N for a record where he sent (n_2) to N.
- N received (n_2) from U where n_2 matched the one N sent to $\mathcal{A}$ in Step 3 of Figure 5.6.
- Since U sent n_2 to N, U must have received (c) from N where the n_1 contained in c matched the one sent to N by U in Step 1 of Figure 5.6.
- N received (c) from $\mathcal{A}$ for a record where he sent (T, n_1, n_2) to $\mathcal{A}$.
- U sent (T, n_1, g^x, S) to N.

Clearly this execution is equivalent to Track 1, and as such can be simulated. We will go on to look at what happens when the protocol aborts in Step 6 of Figure 5.6.

The protocol outputs **Est**. **Failed Linkable** in Step 6 of Figure 5.6:

- U received (c) from N, where c was wrong.
- N received this same (c) from $\mathcal{A}$ for a record where he already sent (T, n_1, n_2) to $\mathcal{A}$.
- Since N sent (T, n_1, n_2) to $\mathcal{A}$, N must have received (T, n_1, g^x, S) from U.

We are still operating in Track 1, and as such can simulate what happens.

The protocol will never abort in Step 11 for this class. If it does not abort in Step 6 that means that N received the correct n_2. Between Step 6 and Step 11 the adversary is not present, and since U, *pos* and N are all honest nothing will go wrong here.

These are all the possible executions of Class 3. $\mathcal{A}$ will never be able to send anything to N when N is not expecting it as N will not have stored an appropriate record. Thus, as described in the protocol π_{KE}, he will simply ignore attempts at this. We hence conclude that the protocol π_{KE} realizes the ideal functionality $\mathcal{F}_{KE}$ in Class 3.

5.2.4 Class 4

In Class 4 both S and *pos* are corrupt. This leaves us with two different tracks that an execution may follow. The first track is shown in Figure 5.7.

An adversary may start a session without U having attempted to start one. An attempt from $\mathcal{A}$ to start a session will follow what we have named Track 2 as shown in Figure 5.8.

When $\mathcal{A}$ tries to start a session he will send (T, n_1, g^x, S) to N. In such a case the following will happen:

- N receives (T, n_1, g^x, S) and as long as $g^x \epsilon \mathbb{G}$, N will send (T, n_1, n_2) to $\mathcal{A}$.

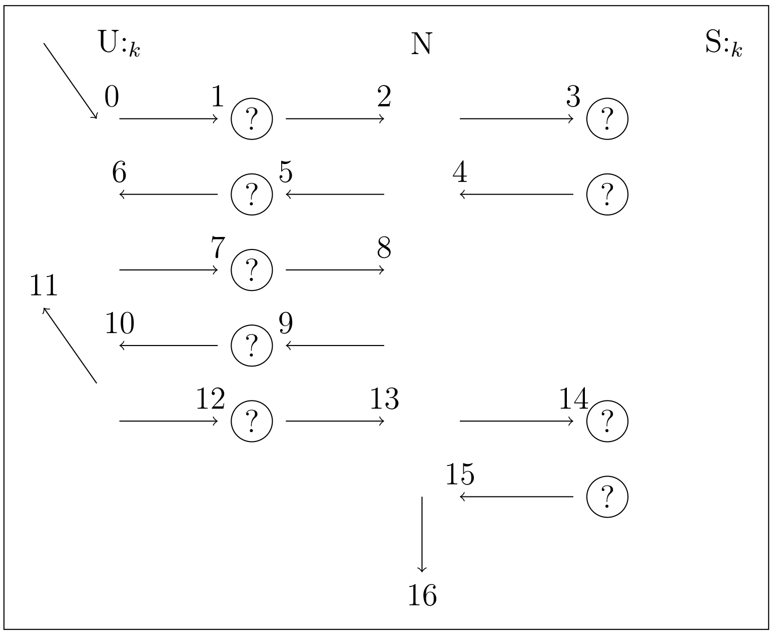

Figure 5.7: Track 1 of the anonymous key establishment protocol π_{KE} for Class 4.

- $\mathcal{A}$ could now choose to stop sending messages to N. However, if he does send a message it has to be on the format (c) where c is a string of ciphertext.
- N will then receive (c) and send (c) to $\mathcal{A}$.
- Again $\mathcal{A}$ could choose to stop sending messages to N. If he decides to send a message the only message N will accept is (n_2). n_2 needs to be the same one as N sent to $\mathcal{A}$ in Step 3.
- Upon receiving (n_2), N will create a g^y and a σ and send (g^y, σ) to $\mathcal{A}$.
- Also here $\mathcal{A}$ could choose to stop the execution by not replying. However, $\mathcal{A}$ knows x as he is the one who created g^x. Thus he is able to calculate g^{yx} and create μ. He could thus send (n_3, μ) to N.
- N would then send (n_3) to $\mathcal{A}$.
- $\mathcal{A}$ could then simply reply (ok) and N would accept.

This execution is depicted in Figure 5.8 and is called Track 2.

Let us go on to look at what guarantees we have when U accepts.

U accepts:

- U received (g^y, σ) from $\mathcal{A}$ for a record where he sent (n_2) to $\mathcal{A}$.
- Due to IND-CCA N created σ.

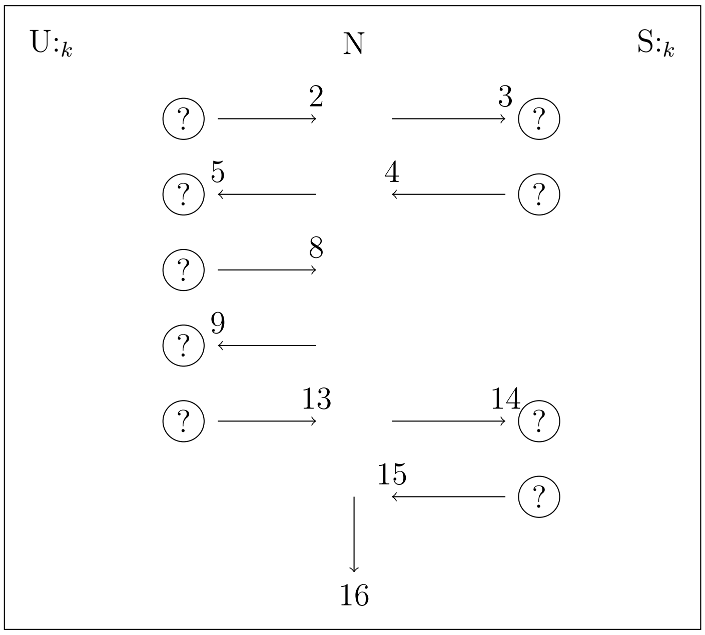

Figure 5.8: Track 2 of the anonymous key establishment protocol π_{KE} for Class 4.

- Since N created σ, N must have received (n_2) from $\mathcal{A}$ for a record where he already sent (c) to $\mathcal{A}$. The n_2 he received must have matched the one N sent to $\mathcal{A}$ in Step 3 of Figure 5.7.
- U received (c) from $\mathcal{A}$ for a record where he already sent (T, n_1, g^x, S) to $\mathcal{A}$. The n_1 contained in c must have matched the one he sent to $\mathcal{A}$ by U in Step 1 of Figure 5.7.
- Since N did not simply ignore (c) when he received it from $\mathcal{A}$, he must have sent (T, n_1, n_2) to $\mathcal{A}$ earlier.
- Since U accepted he must have sent (T, n_1, g^x, S) to $\mathcal{A}$.
- Since N sent (T, n_1, n_2) to $\mathcal{A}$ he must have received received (T, n_1, g^x, S) from $\mathcal{A}$.

We can see that this execution is equivalent to the top part of Track 1. As such it can be simulated. We will now go on to See what might happen after U accepts. When U accepts he will send (n_3, μ) to $\mathcal{A}$.

- If $\mathcal{A}$ wants to send a message to N, the only message that will be accepted is (n_3, μ).
- $\mathcal{A}$ knows g^x and g^y, but due to DDH will not be able to calculate g^{xy} and hence alter μ. He would be able to change n_3, but this lacks motivation.
- N would then send (n_3) to $\mathcal{A}$.

- Upon receiving (n_3) the adversary could either stop or reply (ok).
- On receiving (ok), N will accept.
- U and N will have output the same keys as $\mathcal{A}$ will not be able to change this (due to DDH). They will also be the only ones who know these keys.

We see that when N and U both accept for the same *sid* the execution necessarily follows Track 1. All of the steps of Track 1 need to occur for N to accept given that U has accepted. This can be simulated and we move on to what happens when the protocol aborts in Steps 6 and 11 of Figure 5.7.

The protocol outputs **Est**. **Failed Linkable** in Step 6 of Figure 5.7:

- U received a (c) from $\mathcal{A}$ that was not on the format $\{n_1, \text{-}, \text{-}, T'\}_k$.
- Since U did not ignore the above message he must have sent (T, n_1, g^x, S) to $\mathcal{A}$.
- This is all we know, and $\mathcal{A}$ needs not have involved N in this execution. We can see that this is simply a part of Track 1, and as such is covered by the simulator $\mathcal{S}_{KE}$.

The protocol outputs **Est**. **Failed Unlinkable** in Step 11 of Figure 5.7:

- U received a (g^y, σ) from $\mathcal{A}$ where either g^y or σ was wrong.
- Since U did not ignore (g^y, σ) he must have received (c) at an earlier stage, accepted c and sent (n_2) to $\mathcal{A}$.
- Since U did not ignore c he must have sent (T, n_1, g^x, S) to $\mathcal{A}$, and the c he received must have contained n_1.
- U must then have sent (n_2) to $\mathcal{A}$.
- N does not need to have been present at all during this execution.

All of this must have happened if the protocol outputs **Est**. **Failed Unlinkable**. It could also be that N has been involved in the execution, but we do not know this for sure. However, it does not matter whether N has been present or not. If N has not been present during the execution, the protocol will follow the path shown in Figure 5.9. This is simply a part of Track 1. If N has been present the execution will simply follow to top part of Track 1 from Figure 5.7.

We have now stated that it is possible to simulate all possible protocol executions in Class 4. Thus, we conclude that the protocol π_{KE} realizes the ideal functionality $\mathcal{F}_{KE}$ for Class 4.

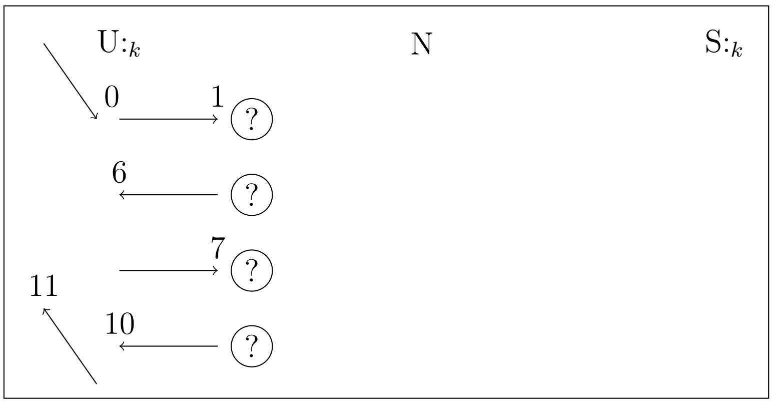

Figure 5.9: Part of Track 1 of the anonymous key establishment protocol π_{KE} for Class 4.

Chapter 6

Conclusion

We have in this paper introduced two protocols, hoping to find a quick way to obtain secure and anonymous communication in mobile connections. The two protocols are inspired by an already existing protocol we have named the TAP protocol. The TAP protocol is summarized in Chapter 5. This protocol boasts better security properties than the two protocols we introduce. However, it also has a running time that is higher than the symmetric key agreement protocol.

	TAP	SKA	AKA
U	1 private decryption 1 signature verification 1 MAC creation	1 private decryption 1 AEAD verification 1 AEAD creation	1 public encryption 1 private decryption 1 AEAD creation 1 AEAD verification
N	1 signature creation 1 MAC verification	1 AEAD verification 1 AEAD creation	1 AEAD verification 1 AEAD creation
S	1 private decryption 2 private encryptions	1 private decryption, 2 private encryptions	1 public decryption 1 private encryption

Figure 6.1: Required Operations for the different protocols.

Figure 6.1 shows the operations that the different protocols require from the three participants. It is easy to see that the symmetric key agreement protocol is the least expensive one. However, this is also the least secure out of the three protocols. As opposed to the TAP protocol it offers no security when operating within Class 4. Further, it is less secure than the asymmetric key agreement protocol as it in some instances reuses tokens, thus making it possible to link different attempts at starting a session to one another. This in turn makes it possible for an attacker to trace an anonymous user around the network.

The asymmetric key agreement protocol has a running time that equals that of the TAP protocol. Signature algorithms use public encryption, and as such the running times for the two are equivalent. Further, both message authentication codes and authenticated encryption with associated data use symmetric encryption and hence has running times equivalent to those of symmetric encryption. Thus, both protocols require two 'expensive' operations and six 'inexpensive' operations to have been performed upon completion.

The TAP protocol is the only protocol out of the three that offers any security when operating within Class 4. However, within the asymmetric key agreement protocol it is not possible to link different attempts at starting sessions to one another.

In conclusion, we consider the TAP protocol to be the better amongst the three. We consider it to be of high importance that the protocol be secure for Class 4. Another approach would be to construct a protocol with both of the desired properties. However, this might require as many as 4 asymmetric encryptions, and could be to expensive to be feasible.

Bibliography

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403. IEEE Computer Society, 1997.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *STOC*, pages 494–503. ACM, 2002.

[GPS11] Kristian Gjøsteen, George Petrides, and Asgeir Steine. A novel framework for protocol analysis. August 2011.

[GPS12] Kristian Gjøsteen, George Petrides, and Asgeir Steine. Secure and anonymous network connection in mobile communications. 2012.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 98–107, New York, NY, USA, 2002. ACM.