Cryptographic protocols. Functionality for secure channels.

Ilja Kuzovkin

January 28, 2011

1 Introduction

We have an predefined functionality F, which provides secure channels for n parties. Our new functionality must not allow reordering of the messages, an adversary A, who can decide when the message will reach it's destination will be obliged to transmit the messages in the correct order, otherwise the protocol will fail. Also we have to forbid duplication of the messages. In other words M_i and P_i will accept same message only once and reject duplicates.

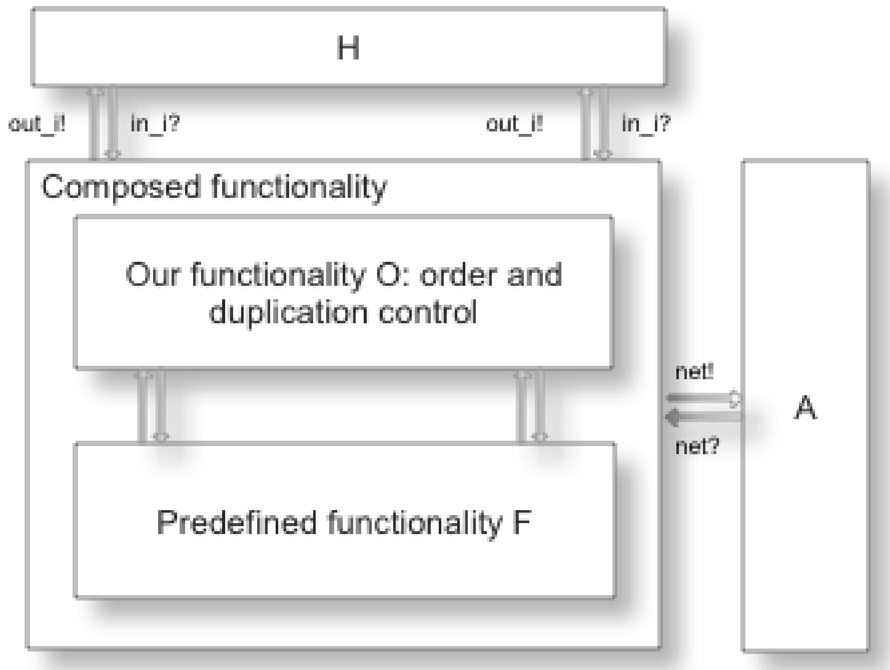

Figure 1. Composed functionality we want to build.

2 Ordering and duplication control

Functionality F includes objects $M_1^F \ldots M_n^F$ which provide functions needed for secure message exchange. M_i^F is controlled by P_i, P_i send a task to the M_i^F through in_i and after this task is complete M_i^F responds through out_i back to the P_i. Our goal is to preserve the order of those messages and somehow make them unique, to avoid duplication. Our functionality O also includes object $M_1^O \ldots M_n^O$ which add the following function to the system: each M_i has two sets of counters $counter_1^{in_i} \ldots counter_n^{in_i}$ and $counter_1^{out_i} \ldots counter_n^{out_i}$, they count how many messages were received from M_j and how many were sent to M_j. When P_i want to send message m to P_j then P_i gives the task to M_i^O, M_i^O appends $counter_j^{out_i}$ to the message, increments

$counter_j^{out_i}$ and forwards it to the M_i^F it, which securely sends it to the M_j^F, it forwards the message to the M_j^O and here M_j^O checks if appended $counter_j^{out_i}$ equals $counter_i^{in_j}$. If equals, then everything is OK and we can forward message to the M_j and increment $counter_i^{in_j}$. The whole scheme looks as follows:

1. P_i *send* (m, j) *to* M_i^O
2. M_i^O *send* $(m||counter_j^{out_i}, j)$ *to* M_i^F, increment $counter_j^{out_i}$
3. M_i^F *send* $(m||counter_j^{out_i}, j)$ *to* M_j^F
4. M_j^F *send* $(m||counter_j^{out_i}, j)$ *to* M_j^O
5. M_j^O check if $counter_j^{out_i} = counter_i^{in_j}$
    - **equal**: M_j^O *send* (m) *to* P_j, increment $counter_i^{in_j}$
    - **not equal**: do nothing

Such approach does not allow to break the order of the messages, because the receiver will wait until he gets correct message and drop all other messages. And if we try to send one message twice, it will be rejected, because counter, which counts incoming messages is already incremented.