More Details for SRKD

I. Message Recovery in SRKD

Message Recovery (Setup): Given a security parameter λ, output the $param = \langle F_q, E, o, P, P_{pub}, P_{sec}, H_1, H_2, F_1, F_2, k_1, k_2\rangle$ as system parameter, where F_q is a finite field, E is an elliptic curve over F_q, a prime o is the order of $G = E(F_q)$, $P \in G$, H_1 H_2 F_1 F_2 are four cryptographic hash functions $H_1 : \{0,1\}^* \times G \to Z_q$, $H_2 : \{0,1\}^* \times G \times \{0,1\}^* \to Z_q$, $F_1 : \{0,1\}^{k_2} \to \{0,1\}^{k_1}$, $F_2 : \{0,1\}^{k_1} \to \{0,1\}^{k_2}$, k_1 k_2 are positive integers and $|o| = k_1 + k_2$. Select a random $x \in Z_q^*$ as the master secret key, compute $P_{pub} = xG$, $P_{sec} = xP$.

Message Recovery (Setup): Given a security parameter λ, output the $param = \langle F_q, E, o, P, P_{pub}, P_{sec}, H_1, H_2, F_1, F_2, k_1, k_2\rangle$ as system parameter, where F_q is a finite field, E is an elliptic curve over F_q, a prime o is the order of $G = E(F_q)$, $P \in G$, H_1 H_2 F_1 F_2 are four cryptographic hash functions $H_1 : \{0,1\}^* \times G \to Z_q$, $H_2 : \{0,1\}^* \times G \times \{0,1\}^* \to Z_q$, $F_1 : \{0,1\}^{k_2} \to \{0,1\}^{k_1}$, $F_2 : \{0,1\}^{k_1} \to \{0,1\}^{k_2}$, k_1 k_2 are positive integers and $|o| = k_1 + k_2$. Select a random $x \in Z_q^*$ as the master secret key, compute $P_{pub} = xG$, $P_{sec} = xP$.

Message Recovery (Sign): In this step, node gets its ID_i, a private key $SK = (R, v)$ and a message $m \in \{0,1\}^{k_2}$. Pick a random value $t_{ran} \in Z_p^*$ and then calculate $m' = F_1(m) \parallel (F_2(F_1(m)) \oplus m)$, $y = (t_{ran}P)_Y \oplus m'$, $h = H_2(ID_i, R, y)$ and $z = t_{ran} + (hv) \, mod \, (o)$. $()_Y$ denotes the Y-coordinate of a point. $\sigma = (R, y, z)$ is the signature for ID_i on m. Then the node encrypts σ as $EN = E(K_{pubBS}, \sigma)$ and sends it to the base station.

Message Recovery (Verify): The base station receives EN and decrypts it as $\sigma = D(K_{priBS}, EN)$ to get the signature $\sigma = (R, y, z)$. To recover the message m, the base station calculates $c = H_1(ID_i, R)$, $h = H_2(ID_i, R, y)$, $m' = y \oplus (zP - hcP_{sec} - hR)_Y$, $m = [m']_{k_2} \oplus F_2({}_{k_1}[m'])$. To verify σ, the base station checks whether ${}_{k_1}[m'] = F_1(m)$. ${}_{k_1}[m']$ means the k_1 bits of m' on the left and $[m']_{k_2}$ means the k_2 bits of m' on the right. The correctness for $m' = y \oplus (zP - hcP_{sec} - hR)_Y$ is as follow:

$$\begin{aligned} & y \oplus (zP - hcP_{sec} - hR)_Y \\ & = [(t_{ran}P)_Y \oplus m'] \oplus \\ & [(t_{ran} + hr_{ran} + hcx)P - h(r_{ran}P) - hcP_{sec}]_Y \\ & = (t_{ran}P)_Y \oplus m' \oplus (t_{ran}P)_Y \\ & = m' \end{aligned} \tag{1}$$

II. Core Code of SPAN+AVISPA in SRKD

We use the "Security Protocol Animator for Automated Validation of Internet Security Protocols and Applications" (SPAN+AVISPA) and a role-oriented language, High Level Protocol Specification Language (HLPSL) to analyze the security of the key distribution phase of QSCP. SPAN+AVISPA has been designed for cryptographic scheme and can analyze their security properties in specified scenarios.

A role-oriented language, High Level Protocol Specification Language (HLPSL), is used to specify the protocol. And SPAN is designed to help scheme developers to writing HLPSL specifications. It can be used to find and build attacks interactively over scheme because of the active intruder of SPAN implementation.

In HLPSL, messages are composed of informations such as agent (means participants), symmetric_key (means symmetric keys), hash_func (means hash functions), nat (means numbers), protocol_id (means labels) and channel (means communication channels).

To simplify the communication process , we implement some roles as below: node_A, node_B, session, environment and goal. The core code of those roles are shown respectively in Listings 1-5.

In Listing 1 (Listing 2), the starting keys K_i and the identifiers of starting keys ID_{K_i} of node_A (node_B) are all stored in the set of KeySetA (KeySetB). The notation of M is the function of MAC and Snd (Rcv) is the channel. The transition shows the messages that node_A (node_B) transmits on the channel and the secrets should be protected.

The core code of session is displayed in Listing 3. We can describe the combination of those participants to establish a session.

Listing 4 presents the core code of environment, which defines the initial knowledge and sessions in communication.

Listing 5 defines the security goal in the scheme. Those data in role of goal cannot be eavesdropped.

Listing 1: Role specification for node_A

```
role node_A
(
    A,B:agent,
%% informations are communicated
%% between A and B
    KeySetA:(text)set,
    Kab:symmetric_key,
    M:hash_func,
    Snd, Rcv:channel(dy)
%% the channel is under
%% Dolev-Yao model
)
played_by A
def=
local  %%local varibles declaration
State:nat  %%natural number
Na,Nb,Ack:text  %%text
init State:=0  %%variables initialization
transition  %%transitions list

%% beginning, if A does not establish
%% the link with neighbor B
%% Rcv(data): receive data
```

```
%% Snd(data): send data
%% secret(data): data
%% requiring confidentiality
%% {data}_Kab: use Kab to encrypt the data
%% data and data': for the same identity
%% name it data for the first appearance
%% and data' for the next appearance
hello.
State=0
/\Rcv(start)
=|>State':=2
%% if receive(start signal)
%% Enter the State==2
/\Snd(ida.idkx.idky.idkw)
/\secret(kx,keyx,{A,B})
/\secret(ky,keyy,{A,B})
sendack.
State=2
/\Rcv(idb.idkx.idky.M(xor(kx,ky)))
=|>State':=4
%% if receive(data above)
%% Enter the State==4
/\Ack':=new() %% created by A
/\Na':=new()
/\Snd(xor(xor(kx,ky),Ack'.Na'.A))
/\secret(Ack',secack,{A,B})
/\secret(Na',secna,{A,B})
/\witness(A,B,b_a_ack,Ack')
recnb1.
State=4
/\Rcv(xor(xor(kx,ky),Ack'.Na').{Nb'}_Kab)
=|>State':=6
%% if receive(dara above)
%% Enter the State==6
/\Snd({Nb'}_Kab)
/\request(A,B,a_b_nb,N')
end role
```

Listing 2: Role specification for node_B

```
%% node_B is similar to node_A
role node_B
(
    A,B:agent,
    KeySetB:(text)set,
    Kab:symmetric_key,
    M:hash_func,
    Snd,Rcv:channel(dy)
)
played_by B
def=
local State:nat
Na,Nb,Ack:text
init State:=1
transition

recyes.
State=1
/\Rcv(ida.idkx.idky.idkw)
/\(in(idkx,KeySetB))
/\(in(idky,KeySetB))
%% in(X,Y) X in Y
=|>State':=3
/\Snd(idb.idkx.idky.M(xor(kx,ky)))
/\secret(kx,keyx,{A,B})
/\secret(ky,keyy,{A,B})
recack. State=3
/\Rcv(xor(xor(kx,ky),Ack'.Na'.A))
=|>State':=5
/\Nb':=new()
/\Snd(xor(xor(kx,ky),Ack'.Na').{Nb'}_Kab)
/\secret(Nb',secnb.{A,B})
/\witness(B,A,a_b_nb,Nb')
recnb2.
State=5
/\Rcv({Nb}_Kab)
=|>State':=7
/\request(B,A,b_a_ack,Ack)
end role
```

Listing 3: Role specification for session

```
%% sessions consist of A and B
role session
(
    A,B:agent,
    M:hash_func,
    Kab:symmetric_key,
    KeySetA,KeySetB:(text)set
    %% text set
)
def=
local
SA,RA,SB,RB:channel(dy)
%% SA the send channel of A
%% RA the receive channel of A
%% SB the send channel of B
%% RB the receive channel of B
composition
node_A(A,B,KeySetA,Kab,M,SA,RA)
/\node_B(A,B,KeySetB,Kab,M,SB,RB)
end role
```

Listing 4: Role specification for environment

```
role environment()
def=
local
KeySetA,KeySetB:(text)set
Snd,Rcv:channel(dy)
%% constant
%% i is the identity of intruder
const a,b,i:agent

%% m is a hash function
idkx,idky,idkw,idkt,ida,idb,kx,ky,kw,kt:text
m:hash_func
kab:symmetric_key

%% immutable value in
%% this environment
keyx,keyy,secack,secna,secnb,
a_b_nb,b_a_ack:protocol_id

%% initialise KeySetA and KeySetB
init
KeySetA:={idkx,idky,idkw,kx,ky,kw}
/\KeySetB:={idkx,idky,idkt,kx,ky,kt}

%% the information that the intruder has
intruder_knowledge={a,b,idkx,idky,idkw,idk}
composition
session(a,b,m,kab,KeySetA,KeySetB)
%% the session between node_A and intruder
/\session(a,i,m,kab,KeySetA,KeySetB)
%% the session between intruder and node_B
/\session(i,b,m,kab,KeySetA,KeySetB)
end role
```

Listing 5: Role specification for goal

```
goal
%% the safety target
%% those data cannot be eavesdropped
secrecy_of
keyx,keyy,secack,secna,secnb
end goal
```