2 OTR key-exchange Protocol

2.1 Protocol modelling

In the given paper, the key-exchange protocol is described as follows:

> In order to add authentication to the basic DH protocol, OTR uses public keys and digital signatures. Specifically, each party A in the OTR network has a pair of secret/public keys (sk_A , vk_A) for a digital signature scheme (implemented using either DSA or RSA signatures).
>
> In a signature scheme, the secret key sk_A is used to create valid signatures by the owner of the key such that no other party can create valid signatures. At the same time, anyone, using the public key vk_A, can verify the validity of any given signature. Note that the public key is associated to the identity of a specific party. OTR adopts a simple and non-hierarchical approach to the distribution of public keys, where each party stores the public keys of the users he communicates with. When first entered users are prompted to verify validity of the public key via fingerprint recognition, much like in SSH [26].
>
> The OTR authenticated key-exchange phase requires that each party signs its Diffie-Hellman value. The public key is sent with the first message:
>
> $$A \rightarrow B : Sign_{sk_A(g^x)}, vk_A$$
>
> $$A \leftarrow B : Sign_{sk_b(g^y)}, vk_B$$
>
> If the public key vk_A is already stored by B and it is associated with the identity of A, this assures B that g^x comes from A and vice versa (in the absence of pre-stored public keys the protocol could use PK certificates). After the verification of the signatures, both parties compute their shared secret value g^{xy} and erase the DH exponents.

In the protocol theory, we use the builtins “signing” and “diffie-hellman”, which include the necessary functions and equations for the protocol. We then use the Key-generation rule to generate key-pairs sk_x (signing key) and pk_x (verification key). Then we use Initialisation rules to create the agents A and B so each agent knows its partner’s pk (A know pk_B and B knows pk_A) from beginning on (as specified in the above description).

We then implement the protocol as follows:

$$1. \quad A \rightarrow B : g^x, sign\{g^x\}_{sk(A)}, pk(A)$$

$$2. \quad B \rightarrow A : g^y, sign\{g^y\}_{sk(B)}, pk(B)$$

Where g is a fixed constant to represent the generator, x and y are fresh values for the ephemeral keys and sk(x) and pk(x) are the key-pairs generated for each agent.

In order to model the executable lemma, we add *Finish* statements at the end of the protocol for each agent; we also make sure that a message received by A corresponds to a message sent by B, that only one agent A and one agent B are created and finally that agent A and agent B are different.

See OTR1.spthy

2.2 Attack on Authentication

The “man in the middle” attack presented in the paper looks as follows:

$$A \rightarrow E[B] : g^x, Sign_{sk_A}(g^x), vk_A$$
$$E \rightarrow B : g^x, Sign_{sk_E}(g^x), vk_E$$
$$E \leftarrow B : g^y, Sign_{sk_B}(g^y), vk_B$$
$$A \leftarrow E[B] : g^y, Sign_{sk_B}(g_y), vk_B$$

The adversary E impersonates B for A (hence the notation E[B] for the communication between A and E) and then forwards g^x to B but signed with E’s key. B then replies to E by sending g^y signed by B’s key and E forwards this message to A.

We were able to reproduce a similar attack in tamarin, but not quite the exact one. In this attack we created one agent A and two agents B (one representing the man in the middle E), but because the agent B is only able to send a g^y where y is fresh, E cannot forward g^x to B, but sends its own g^z to B; in the same manner E cannot forward B’s reply to A and so the attack in tamarin looks as follows:

$$A \rightarrow E[B] : g^x, Sign_{sk_A}(g^x), vk_A$$
$$E \rightarrow B : g^z, Sign_{sk_E}(g^z), vk_E$$
$$A \leftarrow B : g^y, Sign_{sk_B}(g^y), vk_B$$

So at the end A and B won’t be able to communicate because the resulting key is different for A and B (g^{xy} for A and g^{zy}) for B.

See OTR2.spthy