Formal Methods for Information Security Project Report

Freya Murphy Sophie Selgrad

June 3, 2022

1 PACE Protocol

1.1 A simple challenge-response protocol

In `P1.spthy` we formalise the protocol given in Alice & Bob notation as seen in the assignment brief:

$$\begin{aligned} A &\to B : x \\ B &\to A : [x]_{\mathsf{k}(B,A)} \end{aligned}$$

where x is a nonce generated by A, $[M]_K$ denotes the MAC of message M with key K, and $\mathsf{k}(A, B)$ is a symmetric long-term key shared by B and A.

To ensure the key used is unidirectional, we added a restriction to the theory. However, this is not particularly relevant to this iteration of the protocol as only one key is used.

In this protocol A (the initiator) injectively agrees with B (the responder) on the nonce x because only B knows the key $\mathsf{k}(B, A)$. However, B does not injectively agree with A because B has no way of knowing who the nonce x came from.

1.2 Mutual authentication

(a)

`P2a.spthy` models the protocol P2 as shown below:

$$\begin{aligned} A &\to B : x \\ B &\to A : y \\ A &\to B : [y]_{\mathsf{k}(A,B)} \\ B &\to A : [x]_{\mathsf{k}(B,A)} \end{aligned}$$

Here, the unidirectional key restriction is important to ensure that a different key is used in each direction.

The desired security property is for each role to agree with the other on both nonces. In other words, A and B should injectively agree on x and y. The above protocol does not have this property (in either direction), and we found attacks demonstrating such.

(b)

To disprove the injective agreement from the initiator's perspective, we found the following attack:

$$\begin{aligned} A_1 \to A_2 : &\quad x \\ A_2 \to E[A_1] : &\quad y \\ E[A_2] \to A_1 : &\quad z \\ A_1 \to E[A_2] : &\quad [z]_{\mathsf{k}(A_1,A_2)} \\ A_2 \to A_1 : &\quad [x]_{\mathsf{k}(A_2,A_1)} \end{aligned}$$

where $E[A]$ means that the adversary E is impersonating agent A. In this attack, both agents play the role of an initiator and the attack is against injective agreement from A_1's point of view. A_1's nonce x is sent to A_2 and the MAC is received back without any interference from the adversary E. However A_2's nonce y is blocked by the adversary and replaced with an adversary-generated nonce z. A_1, assuming the nonce is from A_2, sends back the MAC of z but this too is intercepted and blocked by E. Hence A_1 believes the nonces to be x and z while A_2 believes them to be y and x.

The following similar attack disproves the injective agreement from the responder's perspective:

$$\begin{aligned} A \to E[B]: &\quad x \\ E[A] \to B: &\quad z \\ B \to A: &\quad y \\ A \to B: &\quad [y]_{\mathsf{k}(A,B)} \\ B \to E[A]: &\quad [z]_{\mathsf{k}(B,A)} \end{aligned}$$

Here A is the initiator and B is the responder while E is the adversary. A's nonce x is intercepted by the adversary and replaced with the adversary's own nonce z, which B returns the MAC of. B's nonce y is sent and the MAC is received back from A without any interference from the adversary. Hence the initiator A believes the nonces to be x and y, while the responder B believes them to be z and y.

The key problem in both attacks is that the MAC of only one of the nonces is being checked: in the first attack E blocks the MAC of z from reaching A_2 and in the second attack E blocks the MAC of z from reaching A. One way to fix this is to send the MAC of both nonces together in the second stage, to ensure that both agents agree on which nonces have been sent and received. We model this improvement in `P2b.spthy`:

$$\begin{aligned} A \to B: &\quad x \\ B \to A: &\quad y \\ A \to B: &\quad [x, y]_{\mathsf{k}(A,B)} \\ B \to A: &\quad [x, y]_{\mathsf{k}(B,A)} \end{aligned}$$

In this new version of the protocol, the injective agreement of A and B on x and y can be verified both from the perspective of the initiator and the responder.

1.3 Introducing a session key

(a)

We model the following protocol as `P3a.spthy`:

$$\begin{aligned} A \to B: &\quad x \\ B \to A: &\quad y \\ A \to B: &\quad [y]_{Kab} \\ B \to A: &\quad [x]_{Kab} \end{aligned}$$

where the session $Kab = \mathsf{kdf}(\mathsf{k}(A, B), x, y)$ depends on the long-term secret key and both of the nonces. We verified injective agreement of A and B on both x, y and Kab (from initiator and responder point of view) and the secrecy of the session key Kab.

(b)

Although only one nonce is included in the message that is MAC'd, both nonces are included in the key derivation. This means that the MAC depends on both nonces and so both can be included in the agreement. In `P2a.spthy` only one nonce was included in the MAC and the MAC depended on neither nonce, so agreement on both nonces could not be reached.

1.4 Replace the password by a nonce

In `P4.spthy` we model the following protocol:

$$\begin{aligned} A \to B: &\quad x, [s]_{\mathsf{h}(\mathsf{k}(A,B))} \\ B \to A: &\quad y \\ A \to B: &\quad [y]_{Kab} \\ B \to A: &\quad [x]_{Kab} \end{aligned}$$

where s is a nonce generated by A and $Kab = \mathsf{kdf}(s, x, y)$. Hence the long-term key $\mathsf{k}(A, B)$ is now only used to encrypt and decrypt the nonce s.

Just as in protocol P3a, we were able to verify that A and B have mutual injective agreement on x, y and Kab and that Kab remains secret.

1.5 Introducing Diffie-Hellman: The PACE protocol

(a)

We model the following protocol in `P5ab.spthy`:

$$\begin{aligned} A \to B: &\quad g^x, [s]_{\mathsf{h}(\mathsf{k}(A,B))}, p \\ B \to A: &\quad g^y \\ A \to B: &\quad [g^y]_{Kab} \\ B \to A: &\quad [g^x]_{Kab} \end{aligned}$$

where $g = \mathsf{map}(s, p)$ and $Kab = \mathsf{h}(g^{xy})$. Again, we were able to verify that A and B have mutual injective agreement on g^x, g^y and Kab and that Kab remains secret.

(b)

We also verified that the above protocol has perfect forward secrecy (PFS) for Kab, without any additional changes being required.

(c)

The secrecy of g is relied upon for the authentication of both parties. If the base is public, an adversary can perform a Man-in-the-middle (MITM) attack by setting up a session key with any of the two parties. Below, we describe one side of the MITM attack with Alice and Eve:

$$\begin{aligned} A \to E: &\quad g^x, [s]_{\mathsf{h}(\mathsf{k}(A,B))}, p \\ E \to A: &\quad g \\ A \to E: &\quad [g^x]_{Kab} = [g^x]_{h(g^x)} \end{aligned}$$

Bob never receives the first message because it is intercepted by Eve. Eve then sends the publicly-known g back to Alice (effectively using nonce $y = 1$). Deriving the shared key Kab is then trivial for Eve, as it is simply $\mathsf{h}(g^x)$. By repeating these steps with Bob, Eve can mount a successful MITM attack.

(d)

In `P5ab.spthy` we used tags inside the MACs to specify the role of the sender (not shown in the Alice & Bob notation above). Removing these tags leads to a reflection attack on injective agreement from the initiator's perspective:

$$\begin{aligned} A \to E[B]: &\quad g^x, [s]_{\mathsf{h}(\mathsf{k}(A,B))}, p \\ E[B] \to A: &\quad g^x \\ A \to E[B]: &\quad [g^x]_{Kab} \\ E[B] \to A: &\quad [g^x]_{Kab} \end{aligned}$$

Since $A's$ half-key g^x is sent in the clear, the adversary E can simply separate it from the rest of A's first message and send it back to A as g^y. Since g^y is also sent in the clear, A has no way to check who this half-key has come from. A responds with the MAC of g^x which is simply reflected back to become the final message of the protocol.

The problem here is that A accepts the same half-key that it sends out (g^x). If x and y are truly nonces chosen at random then it is extremely unlikely that B would choose the same nonce as A. Therefore adding a restriction to discard any traces where $g^x = g^y$ prevents this attack. This is shown in `P5d.spthy`, which has the same Alice & Bob notation as `P5ab.spthy` above (although note that no tags are sent).

2 The Off-the-Record Messaging Protocol

2.1 Modelling the original OTR Key Exchange

In `OTR1.spthy` we model the OTR Key Exchange as given in [1]:

$$\begin{aligned} A \to B: &\quad g^x, \{g^x\}_{\mathsf{sk}_A}, \mathsf{vk}_A \\ B \to A: &\quad g^y, \{g^y\}_{\mathsf{sk}_B}, \mathsf{vk}_B \end{aligned}$$

where $\{M\}_{\mathsf{sk}}$ denotes the signature on message M using signing key sk and vk_X is the public verification key associated with the secret signing key sk_X. We additionally send the half-key in plaintext alongside the signature to more easily model extracting the message from the signature in Tamarin.

A and B already know the verification keys of potential communication partners, so sending vk_X serves to notify the receiver who the sender is, rather than just providing an arbitrary verification key. Thus B already knows that only A has the matching secret key sk_A when they receives the verification key vk_A. This allows B to verify that the signature on the half-key g^x was indeed generated by A. The same is true when A is verifying B's signature.

We modelled checking the validity of signatures in Tamarin using an equality restriction.

2.2 Authentication Failure

In `OTR2.spthy` we model the same protocol as above but add a lemma for injective agreement on the derived key $K = \mathsf{h}(g^{xy})$ from the initiator's perspective, which fails to verify. We further restricted this lemma to identify the attack described in [1], where A and B derive the same key K but do not agree on each other's identities:

$$\begin{aligned} A \to E[B]: &\quad g^x, \{g^x\}_{\mathsf{sk}_A}, \mathsf{vk}_A \\ E \to B: &\quad g^x, \{g^x\}_{\mathsf{sk}_E}, \mathsf{vk}_E \\ B \to E: &\quad g^y, \{g^y\}_{\mathsf{sk}_B}, \mathsf{vk}_B \\ E[B] \to A: &\quad g^y, \{g^y\}_{\mathsf{sk}_B}, \mathsf{vk}_B \end{aligned}$$

In this attack, $(\mathsf{sk}_E, \mathsf{vk}_E)$ is the key pair belonging to a compromised agent E, who is controlled by the adversary. E replaces A's signature on g^x with their own, while keeping the half-key the same. Thus A and B end up with the same final value g^{xy} (which E cannot calculate, as they only know g^x and g^y), but disagree on who they are communicating with: A thinks they are communicating with B and B thinks they are communicating with E.

2.3 Improvement

In `OTR3.spthy` we model the following protocol:

$$\begin{aligned} A \to B: &\quad g^x, \{g^x, B\}_{\mathsf{sk}_A}, \mathsf{vk}_A \\ B \to A: &\quad g^y, \{g^y, A\}_{\mathsf{sk}_B}, \mathsf{vk}_B \end{aligned}$$

This is the same as the original protocol but with the identity of the intended recipient also included in the signature, so that the recipient can verify that they are who the sender expected to be communicating with.

A modified injective agreement lemma shows that the previous attack is now prevented, but full injective agreement from the initiator's perspective is still not achieved when multiple instances of the initiator role are permitted. Modifying OTR3 to include tags in the signature still does not prevent attacks.

We only analysed injective agreement from the initiator's perspective. Analysis from the responder's perspective would require that B make a Commit claim at the end of its execution, after it has sent the second message, and that A make a corresponding Running claim after A has learned g^{xy} but before B's Commit claim. This is not possible since A only learns g^{xy} after B has sent g^y and made its Commit claim.

Both the secrecy and PFS of the key K can be verified.

2.4 SIGMA

We further improved the OTR Key Exchange by modelling the SIGMA protocol in `OTR4.spthy`:

$$\begin{aligned} A \to B: &\quad g^x \\ B \to A: &\quad g^y \\ A \to B: &\quad A, \{g^y, g^x\}_{\mathsf{sk}_A}, [\text{"0"}, A]_{K_m}, \mathsf{vk}_A \\ B \to A: &\quad B, \{g^x, g^y\}_{\mathsf{sk}_A}, [\text{"1"}, B]_{K_m}, \mathsf{vk}_B \end{aligned}$$

In the SIGMA protocol both agents first send their half-keys in the clear and then respond with their own identity, a signature on both half-keys, a MAC of a constant and their own identity using a shared session key K_m, and their public verification key. The session key $K_m = \mathsf{h}(g^{xy})$.

In this improved version of the protocol PFS of K_m still holds, and we also achieve injective agreement from the initiator's perspective. However, injective agreement from the responder's perspective (which is now possible to analyse since there are 4 messages) is falsified.

References

[1] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure Off-the-Record Messaging. In WPES, pages 81–89, 2005.