A Guide to our Tamarin Analysis of Exported Authenticators

Jonathan Hoyland

1 Introduction

`draft-sullivan` repurposes messages from the Transport Layer Security (TLS) 1.3 handshake to construct an authentication protocol to be used inside a TLS connection. We analysed `draft-sullivan` with the Tamarin analysis tool[Sch+12]. This document describes the Tamarin model we used to verify and explore the guarantees of `draft-sullivan`.

2 Model

The exported authenticators (EAs) model is written in a similar manner to the TLS model. The model is divided into two main pieces, the lemmas and the protocol model. Each of these pieces is further decomposed into a number of smaller pieces. As in the TLS model the sections are all composed into one Tamarin input file using `m4`. This decomposition serves two purposes. The first is to make it easier to construct the model from logical pieces, so that code is not unnecessarily repeated, this makes it easier to make changes to the model reliably. The second is to make clear the separation between the underlying TLS layer, whose security properties are assumed, and the EA layer, whose properties we are analysing. The protocol model is made of nine files which I will discuss in turn.

1. `client.m4i`
2. `server.m4i`
3. `adversary.m4i`
4. `channel.m4i`
5. `pki.m4i`
6. `msgs.m4i`
7. `crypto.m4i`
8. `header.m4i`
9. `model.m4i`

2.1 `client.m4i`

The `client.m4i` file contains the rules defining the client. The first section of these rules define a pseudo-TLS handshake. Rather than emulate a full TLS handshake with all its attendant complexity, we simplify our model by using an abstracted TLS handshake which is secure by definition, i.e. we do not provide the adversary with rules she can use to attack it, see 2.3.

In this way, rather than reanalyse TLS 1.3 as we did in our prior work[Cre+17], we assume TLS has the necessary security properties, and restrict ourselves to analysing their interaction with the properties we try to prove about EA.

The simplified handshake only differentiates between unilateral and bilateral handshakes, and does not model all the various other modes such as session resumption and zero round-trip time (0-RTT). The EA model has been designed with integration into the TLS 1.3 model in mind, and this is left to future work.

The remaining rules define the `draft-sullivan` protocol. Because the EA protocol is defined as a loop, and does not require certificate requests to be handled in the order they were received, the rules themselves can trigger in almost any order.

In the TLS 1.3 model the `State` fact was used to ensure that honest actors followed the steps of the protocol in the correct order. Because `draft-sullivan` does not have strong ordering requirements the `State` fact is used to bind together the actions of a single session. The ordering requirements that do exist, for example that a request must be sent before a response is received, are enforced by `PendingReq` facts. These facts represent the actors memory of either sending or receiving `CertificateRequests`. By having the request steps output consumable `PendingReq` facts the corresponding send/receive steps are unable to trigger until the `PendingReq` fact is available, enforcing the proper ordering.

2.2 `server.m4i`

The server file contains all the rules that define the server's behaviour. It mirrors the client rules almost exactly. The main difference is that the server is allowed to send a certificate spontaneously, whereas the client may only receive a certificate spontaneously.

2.3 `adversary.m4i`

The adversary file contains all the rules governing the behaviour of the adversary. These are, by design as general as possible to avoid limiting the adversaries actions. The adversary has the ability to compromise TLS sessions, and to send and receive messages over channels for which she knows the master secret.

The attacker also has the ability to compromise long term certificates. This ability parallels the threat model of the work done by Delignat-Lavaud[**dl**]. Delignat-Lavaud's work on channel bindings and compound authentication provides a generic protocol that achieves compound authentication. Proving that `draft-sullivan` refines the generic protocol allows us to rely on Delignat-Lavaud's proof that all such protocols achieve compound authentication.

2.4 `channel.m4i`

The channel file defines the basic rules for creating a TLS channel, and sending and receiving messages over a TLS channel. The `Create_TLS_Channel` rule is one of the few places where restrictions are used in the model. Restrictions will force Tamarin to ignore paths that do not match the restriction, rather than proving that they cannot happen. In this case the restriction `Check_is_none` is used to enforce a consistent model of bilateral authentication, i.e. that if a client does not send a certificate then the channel cannot be considered bilaterally authenticated, and conversely that if the client does send a certificate it is unilaterally authenticated. This is not entirely consistent with how TLS operates, because in TLS 1.3 the client can send a certificate to the server, and the server might either reject it, or accept it, and then later de-authenticate the client, see **??**. However we treat these cases as equivalent to the case where the client does not send the certificate.

The `Create_TLS_Channel` outputs two Exporter facts, one bound to the `tid` of the client and the other bound to the `tid` of the server. This is a design decision to simplify the rules necessary for calculating exporters without giving the client or server direct access to the master secret.

Because all rules are merged into a single file before analysis there is no practical difference between computing the rules in the channel definition as opposed to in the client and server definitions; however because `draft-sullivan` just defines calls to the TLS exporter interface, rather than accessing the `exporter_master_secret` itself, we have tried to emulate this and ensure a logical separation between the two layers.

2.5 pki.m4i

The Public Key Infrastructure (PKI) is built in the simplest way possible, and matches the construction in the Tamarin manual[**tamarin-manual**] almost exactly. The key difference is the outputting of the `DelegateLtk` fact. This fact allows a long-term key to be bound to an actor other than the owner. This is one of the intended purposes of `draft-sullivan`. An EA proves that an actor controls two certificates. If the first certificate is the one used to sign the TLS channel, then the purpose of sending an EA is to prove to the peer that you also have an identity with a different set of

permissions. Proving to the peer that you have an identity with the same permissions only makes sense in the scenario where the server de-authorises the client or doesn't accept the clients certificate, which we are modelling by treating that as the client not sending a certificate.

The public key infrastructure (PKI) infrastructure is modelled with a fresh value for the long-term key (LTK). The adversary is not restricted in any other way from generating a certificate. This means that the adversary can generate a delegated credential to any server, and therefore that the attacker can successfully man-in-the-middle (MITM) any connection. However, `draft-sullivan` does claim to protect against incorrectly issued certificates, so this "attack" does not actually defeat the security guarantees of `draft-sullivan`.

By writing our lemmas carefully we can make sure that the case where the adversary has been issued a delegated credential is accepted, whilst correctly rejecting cases where the adversary is *not* issued such a credential but still successfully performs a MITM attack.

2.6 msgs.m4i

The messages file defines the messages of the EA protocol. The rebalancing of `draft-sullivan` and `draft-bishop`[BST17] to include certificate requests in `draft-sullivan` allowed for a substantial simplification of the model, because it meant that `draft-sullivan` no longer just specified a number of messages, but actually defined a protocol for requesting and receiving certificates. This meant that the model no longer needed to capture both `draft-sullivan` and `draft-bishop` together, but could consider just `draft-sullivan`.

The message formats are defined in `draft-sullivan` as using the "message structures from section 4.4 of [TLS 1.3 [Res17, Section 4.4]], but different parameters."[Sul17, Section 2]. For this reason we re-used the exact message formats from our previous modelling of TLS 1.3 [Cre+17]. This served two purposes, first, that the message formats had already been formally verified, which gave a high level of confidence in their correctness, and second, that a future analysis merging both models and checking them as a single piece would be easier.

Although `draft-sullivan` uses the same message formats as TLS 1.3 , because they use different keys and labels for secure values they cannot be passed from a handshake to an EA.

The `certificate_extensions` are treated as a public value. This is a fault-preserving simplification. Hui and Lowe developed the idea of fault-preserving simplifying transformations[HL01], where a transformation could be applied to protocol that would simplify it, whilst also preserving any errors that might have been in it.

`certificate_extensions` are a set of values defining which extensions a client or server is willing to support, where the server picks a subset of those offered by the client and these are generally static and public. The `SpontaneousCertificate` message however, cannot use a subset of the client's `certificate_extensions`, because the server doesn't wait for the clients request before sending the `SpontaneousCertificate`. In this case, the specification suggests a default of using the `certificate_extensions` from the outer TLS handshake. In this case the values are not public, but include a fresh value, the Diffie-Hellman (DH) exponents. However, treating this value as public strictly increases the attackers power, and this is a fault-preserving simplification.

2.7 crypto.m4i

The cryptography primitives are encoded in `crypto.m4i`. These rules are taken from the definitions in the TLS 1.3 draft[Res17] and closely mirror the rules used in our previous TLS work[Cre+17].

These rules implement a symbolic version of an HMAC-based key derivation function (HKDF), with the key labels hard-coded into the macros. The model uses two values, `ms` and `TLS_transcript`, as fresh values. In practice these values are derived from the TLS handshake, but we rely on the property in the TLS 1.3 draft that guarantees that session keys are unique. Because we are analysing a post-handshake protocol, the `TLS_transcript` is fixed, and so we represent it as a static fresh value.

2.8 header.m4i

The header contains all the restrictions and function declarations. There are three restrictions that our model relies on.

The first two, `Eq_check_succeed` and `Neq_check_succeed` allow us to simulate an honest actor performing an equality or inequality check. We restrict Tamarin from looking down paths where an honest actor fails a check but continues running. The third restriction, `is_none`, checks that a TLS handshake where the client sends a certificate is bilateral. As discussed in Section 2.4 this restriction does not quite capture the full complexity of TLS authentication, but is a simplification.

2.9 model.m4i

The model file pulls together all the different parts of the model. Constructing the model in this way makes it very easy to exclude pieces of the model to make testing of other sections easier.

3 Lemmas

The lemmas in the Tamarin EA model look to prove three major properties,

1. that the master secret of the underlying TLS session is not leaked by the EA protocol,

2. that only someone who knows the private key of a certificate can prove ownership of it,

3. and that two EAs sent over the same channel were generated correctly.

To achieve this a number of helper lemmas need to be proven. Helper lemmas are lemmas that help Tamarin make pieces of later proofs easier. Some proofs, in particular those that need induction, can be unprovable without helper lemmas. Tamarin only supports induction at the topmost level of a proof. Thus subsections of a proof that need induction need to be separated out into helper lemmas.

Lemmas marked for re-use in Tamarin are used in later lemmas as proof steps without further verification. For example, the lemma `one_start_per_tid` claims that for a given thread identifier, there can only be one start action. If in a later proof Tamarin finds the start of thread it assumes that no other starts exist. This is different from a restriction in that for a proof of a later lemma to hold all prior helper lemmas need to be proven. A restriction is just assumed to be true in all cases, and does not need to proven.

When Tamarin loads a model its first step is to find all the sources of each Fact. A Fact's Sources are the rules or sets of rules from which it could have come. Tamarin splits these into two types of Source, refined and unrefined. Unrefined sources consist of every rule that outputs the fact in question.

Tamarin only partially unwinds the preconditions for sources. Once Tamarin has found all the raw sources for all Facts it then refines them. This means it tries to exclude some of the sources for a particular fact that have an impossible derivation, or to solve infinite chains of derivation inductively.

In the EA model the `PendingReqI` fact has 16 raw sources, considering all the various combinations of handshakes that could occur. However, a number of these handshakes are not actually possible. Because Tamarin doesn't expand the Source tree fully some traces can be impossible.

This can occur when one rule produces two facts with related values, for example if they both include a thread id, and later both those facts, or derivatives of them, are consumed by the same rule. This creates a dependency between two of the input facts which Tamarin might not detect. In this case, if one of those facts is unrolled back to the point where it splits, Tamarin might not detect that the other fact created does not match the fact it consumes in the latter rule.

By writing a lemma that claims some sources are impossible, or by enumerating all the possible sources and marking it as a source lemma, Tamarin will attempt to prove that lemma automatically as it loads. Assuming it succeeds, it will then use that to compute a set of refined sources. These sources are then used to determine the branches that must be explored in a backwards search.

4 Models

There are two model files in the project, `properties.m4` and `reachability.m4`. `properties.m4` constructs the model and appropriate lemmas. `reachability.m4` constructs a number of reachability tests that can be used to diagnose issues with the model. The `at_most_of.m4i` file constructs rules that limit the number of times a fact can occur. This is very useful in conjunction with the reachability tests by allowing for the exclusion of many paths that can make reachability tests hard to verify.

5 Results

The three lemmas mentioned at the start of Section 3 can all be proven with Tamarin. These proofs can be viewed in the Proofs folder. However, they do not prove under all threat models. The model includes as a final lemma a stronger compound authentication lemma. This lemma has a counterexample in the Proofs folder.

This threat model allows the adversary to be able to compromise master secrets and certificates. A stronger version of the draft is being designed.

References

[BST17] Mike Bishop, Nick Sullivan, and Martin Thomson. *Secondary Certificate Authentication in HTTP/2*. Internet-Draft draft-bishop-httpbis-http2-additional-certs-05. `http://www.ietf.org/internet-drafts/draft-bishop-httpbis-http2-additional-certs-05.txt`. IETF Secretariat, Oct. 2017.

[Cre+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A comprehensive symbolic analysis of TLS 1.3". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, USA*. 2017.

[HL01] Mei Lin Hui and Gavin Lowe. "Fault-preserving simplifying transformations for security protocols". In: *Journal of Computer Security* 9.1-2 (2001), pp. 3–46.

[Res17] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet-Draft draft-ietf-tls-tls13-22. `http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-22.txt`. IETF Secretariat, Nov. 2017.

[Sch+12] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. Ed. by Stephen Chong. IEEE, 2012, pp. 78–94.

[Sul17] Nick Sullivan. *Exported Authenticators in TLS*. Internet-Draft draft-ietf-tls-exported-authenticator-05. Work in Progress. Internet Engineering Task Force, Dec. 2017. 9 pp.