An extended abstract of this paper appears in the proceedings of CCS'19. This is the full version.

# Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures

Dennis Jackson[1], Cas Cremers[2], Katriel Cohn-Gordon[3], and Ralf Sasse[4]

[1]University of Oxford
[2]CISPA Helmholtz Center for Information Security
[3]Independent Scholar
[4]Department of Computer Science, ETH Zurich

4th July 2019

**Abstract**

The standard definition of security for digital signatures—existential unforgeability—does not ensure certain properties that protocol designers might expect. For example, in many modern signature schemes, one signature may verify against multiple distinct public keys. It is left to protocol designers to ensure that the absence of these properties does not lead to attacks.

Modern automated protocol analysis tools are able to provably exclude large classes of attacks on complex real-world protocols such as TLS 1.3 and 5G. However, their abstraction of signatures (implicitly) assumes much more than existential unforgeability, thereby missing several classes of practical attacks.

We give a hierarchy of new formal models for signature schemes that captures these subtleties, and thereby allows us to analyse (often unexpected) behaviours of real-world protocols that were previously out of reach of symbolic analysis. We implement our models in the Tamarin Prover, yielding the first way to perform these analyses automatically, and validate them on several case studies. In the process, we find new attacks on DRKey and SOAP's WS-Security, both protocols which were previously proven secure in traditional symbolic models.

## 1 Introduction

Digital signatures are a core cryptographic primitive, whose well-known definition has hardly changed for over 30 years. This definition, Existential Unforgeability under an Adaptive Chosen Message Attack (EUF-CMA), requires that no adversary can construct a valid signature for a new message without knowing the corresponding secret key. EUF-CMA is widely used and considered standard. However, it allows for some subtle and perhaps unexpected behaviours: EUF-CMA-secure signature schemes can and do permit adversaries to, for example,

1. given a signature, generate a new key pair that can also be used to verify the signature;
2. change some bits of a signature without affecting its validity;
3. given a signature but not its message, produce another signature on the message for an adversarial key pair; or
4. compute weak keypairs for which a single signature can verify against multiple messages.

Indeed, in Table 1 on the following page we give a number of widely-used concrete signature schemes (with columns corresponding in order to the properties above—we will expand on these later), and see for example that RSA-PSS allows generating new public keys against which existing signatures verify, and that ECDSA allows for signatures which verify for two different messages. These behaviours are not the result of implementation bugs or mistakes; rather, they are a consequence of how each signature scheme has been designed.

The existence of these subtle behaviours has enabled concrete attacks on protocols. For example, the Mt. Gox Bitcoin exchange famously lost millions of dollars because the malleability of the underlying signature scheme could be exploited [39], and an earlier draft of the Automatic Certificate Management Environment (ACME) certificate issuance protocol would have enabled adversaries to issue certificates for any Let's Encrypt domain on the Internet because of a signature key substitution vulnerability [4].

| Signature scheme | CEO/DEO | No-Mall. | No-ReSign | No-Coll. |
|---|---|---|---|---|
| RSA-PKCSv1.5 | • [64] | • [53] | • [50] | ▲ |
| RSA-PSS | • [64] | ▲ | • [58] | ▲ |
| DSA | • [64] | ✓[67] | ▲ | • [69] |
| ECDSA-FreeBP | • [26] | • [67] | ▲ | • [67] |
| ECDSA-FixedBP | ✓[59] | • [67] | ▲ | • [67] |
| Ed25519 | ✓[47] | • [19] | ✓[19] | • [19] |
| Ed25519-IETF | ✓[47] | ✓[52] | ✓[19] | • [19] |

Table 1: Subtle behaviours of concrete EUF-CMA-secure signature schemes. Columns refer to the security property, i.e., the *absence* of some unexpected behaviour: Conservative Exclusive Ownership (CEO)/Destructive Exclusive Ownership (DEO) (no DSKS), non-malleability, non-resignability and non-collidability. We will expand on all these properties later in the paper.
✓ means that the security property holds, and therefore the corresponding unexpected behaviour is not present. • means that the behaviour is present. For example, ECDSA-FreeBP signatures are malleable and allow for DSKS attacks. ▲ means that we conjecture that the behaviour is absent, so the security property holds, but this has not been proven.
FreeBP (resp. FixedBP) means the signature scheme's base point is considered a parameter of the signature (resp. fixed in advance).

At the protocol level, there have been many advances in automated analysis tools. Indeed, automated analysis of security protocols has made its way into mainstream security practice in recent years: notable success in the analysis of widely deployed standards such as TLS 1.3, 5G, and many more [18, 20, 21, 34, 35, 37, 38, 40, 56] have demonstrated its value in handling large scale, real world protocols with complex substructure. Tools such as ProVerif and Tamarin are given a protocol specification and its security requirements, and provide either a proof that no attack exists within their model or a concrete attack trace violating a security requirement, without requiring users to consider all potential edge cases or decide whether a primitive is being used correctly.

Symbolic verification tools do not directly operate on the cryptographic definition of digital signatures but on an approximation, which has also hardly changed for several decades: signing and verifying are both considered as abstract function symbols, and an equation is added to model that verification of a correctly-generated signature must succeed. Unfortunately, this approximation does not include the subtle behaviours we described above, and implicitly assumes they are not possible. This means that the tools can miss real attacks: for example, as we will show later, we find a number of new attacks on protocols which had previously been formally verified as secure with the traditional symbolic model of signatures.

In this paper, we remedy this shortcoming, by introducing new models for automated verification of protocols using signature schemes. Unlike previous work, our models capture these protocol attacks while allowing for automated attack finding and verification. We explore models capturing some specific attacks, generalise to a multi-purpose verification model, and then apply our techniques to a number of well-known protocols.

*Contributions.* Our main contributions include:

1. We develop a new hierarchy of tool-agnostic symbolic models for digital signatures, which captures attacks and behaviours omitted from traditional models. Our models include several *falsification* models, which capture subtle behaviours, and a *verification* model, which is close to the computational definition. These models make it possible to analyse the impact of concrete signature schemes on protocols, yielding more accurate and meaningful protocol analysis. Our models are tool-agnostic and can therefore be used to improve other symbolic approaches.
2. Using the Tamarin prover, we develop the first automated method for finding, or proving the absence of, attacks on security protocols that exploit subtle behaviours of provably-secure signature schemes. We evaluate the effectiveness of our approach on a range of case studies, which show that the approach is effective at both attack finding and verification.
3. We use our models to find known and new attacks on protocols which were previously proven secure in coarser models. Specifically, we break correlation and secrecy for WS-Security X.509 Mutual Auth (once widely used to secure SOAP services), and break authentication and collusion-resistance for the DRKey key exchange protocol (used for routing). We also automatically find, and verify the fix for, the known key substitution attack on ACME draft 4. These protocols were all previously verified using automated analyses under the traditional signature model, which does not capture our attacks.

*Outline.* We have three main sections after the background (§2). First, in §3 we give symbolic models for improved attack finding, identifying specific properties which are not captured by existing models, and show how this enables us to find attacks on protocols. However, there may exist further behaviours of signature schemes that are not instances of the properties we identified. Therefore, in §4 we move to verification instead of falsification, giving a general model for signature schemes that makes minimal assumptions on the signature scheme. In §5 we apply our techniques to further case studies. We discuss further related work in §6 and conclude in §7.

# 2 Background

## 2.1 Computational Model

We begin with the classical definition of signature schemes, stated informally. (The formal definitions can be found in [45, 54].)

**Definition 1.** A *digital signature scheme* is composed of three polynomial time algorithms:

1. $\mathsf{KGen}$, a probabilistic algorithm, takes in the security parameter and produces a pair $(sk, vk)$.
2. $\mathsf{Sig}$, a probabilistic algorithm, takes in a private key $sk$ and a message $m$ and produces a signature $s$.
3. $\mathsf{Vf}$, a deterministic algorithm, takes in a verification key $vk$, message $m$ and signature $s$ and outputs success or failure.

It is *correct* if $\mathsf{Vf}(vk, m, \mathsf{Sig}(m, sk))$ succeeds with high probability for all messages $m$ and any $(sk, vk)$ output by $\mathsf{KGen}$.

The essential security definition that nearly all signature schemes are expected to meet is existential unforgeability against an adaptive chosen message attack:

**Definition 2.** A signature scheme is *existentially unforgeable under an adaptive chosen message attack* or *EUF-CMA*-secure if no PPT adversary has a non-negligible advantage in this experiment:

1. The challenger generates a keypair and gives the public key to the adversary.
2. The adversary may adaptively query a signing oracle polynomially often which returns a signature on the chosen message.
3. The adversary wins if they can output a message and signature pair which is verified and the message was not previously given as input to the signing oracle.

This security definition captures *forgery resistance*: even an adversary that can adaptively query for signatures on messages of their choice cannot forge a signature for a different message.

## 2.2 Existing Symbolic Models

In the unbounded setting, automated tools such as Tamarin [65], ProVerif [27], Maude-NPA [43], and CPSA [42] have a long history and have seen many improvements over time. More recently, Tamarin and ProVerif have supported real world protocol development through analysis of protocols such as TLS 1.3, 5G, and Signal, as mentioned in the introduction.

Such tools accept a description of a protocol and its security properties and search for a trace demonstrating a security property violation, or proof that no violation occurs within the tool's framework. Each tool is required to model the behaviour of cryptographic functions, such as digital signatures.

These tools use a term algebra with an equational theory to model cryptographic messages. The term signature contains the function symbols with their arity, representing the applicable cryptographic algorithms, such as signing or verification. The equational theory can then be used to model the properties of the algorithms.

We continue by giving the equational theories for digital signatures. We declare the function symbols after the keyword `functions`, with their arity after a `/` and the equations after the keyword `equations`, where all non-declared symbols are interpreted as variables. For brevity and readability, we use Tamarin's notation.

Two typical models are given here:

```
functions: verify/3, sign/2, pk/1, true/0
equations: verify(sign(m, sk), m, pk(sk)) = true
```

Standard Signature Model used by Tamarin and ProVerif

```
functions: rvlSign/2, rvlVerify/3, getMsg/1, pk/1, true/0
equations: rvlVerify(rvlSign(m,sk),m,pk(sk)) = true
           getMsg(rvlSign(m,sk)) = m
```

Signature Model with Message Recovery used by both tools

The equation in the standard symbolic signature model allows the protocol and adversary to verify signatures by applying `verify` to a claimed signature, alongside the expected message and public key, and test if the resulting term is equal to `true`. In these models of signatures, a public key is considered to be a function of a secret key, rather than the typical notion that both are functions of a seed value. We follow the traditional symbolic model in this paper. Note that converting between the two representations is straightforward.

These models have become a standard over the past 20 years, see [68, Page 37] for Tamarin's version, and [29, Page 14] for ProVerif's. Indeed the message recovery model appears verbatim in [27], the original ProVerif paper. Tools such as CPSA [42] and Maude-NPA [43] use similar models.

The ProVerif manual [29] proposes an alternative signature model that is non-deterministic and has message and key recovery:

```
functions: spk/1, sign/3, getmess/1, checksign/2,getkey/1
equations: checksign(sign(m,k,r),spk(k)) = true
           getmess(sign(m,k,r)) = m
           getkey(sign(m,k,r))  = spk(k)
```

ProVerif's Probabilistic Model with Message and Key Recovery (Translated into Tamarin notation)

This removes the bijection between signatures and the messages they correspond to, allowing for more behaviour to be expressed. In this model it is possible to extract the message from a signature using the second equation, and to extract the public key from a signature using the third equation. We were unable to find a publication actually using it in practice.

## 2.3 Tamarin Model background

As explained previously, Tamarin uses a term algebra with an equational theory to model cryptographic primitives and their properties. The execution of a protocol in an environment with an adversary is then represented as a labeled transition system. The state consists of messages on the network, the adversary knowledge, and the internal states of the protocol participants. Protocol and adversary interact by exchanging messages on the (adversary-controlled) network. Both protocol rules and adversary capabilities are specified as labeled multiset rewrite rules. These are used to define a transition system that specifies a set of traces, which model all possible sequences of events. The security requirements are then specified in a (guarded) fragment of first-order logic, expressed over a trace. We will now detail the concepts that are required for our exposition in the remainder of the paper.

The state of the transition system is given by a multiset of *facts*. Facts are special symbols that take any (fixed) number of terms as their arguments. There is a special set of them that encodes messages on the network as well as adversary knowledge. All other facts represent the protocol state. We generally write `Factname(t1, t2, t3)` for a fact named `Factname` with three terms as its arguments.

A *labeled multiset rewriting rule* is then of the form

```
rule name:
  [ l ] --[ a ]-> [ r ]
```

where `rule` is a keyword, `name` is an identifier, and `l`, `a`, `r` are multisets of facts representing the *premises*, *actions*, and *conclusions* respectively. All of these may contain variables. Some rules may have no associated action, and in that case we omit the action `[ a ]` in the rule description.

The *labeled transition system* operates on ground terms, i.e., terms without variables. A rule is *applicable* in a given ground state when an instantiation of the premises of the rule is a subset of the

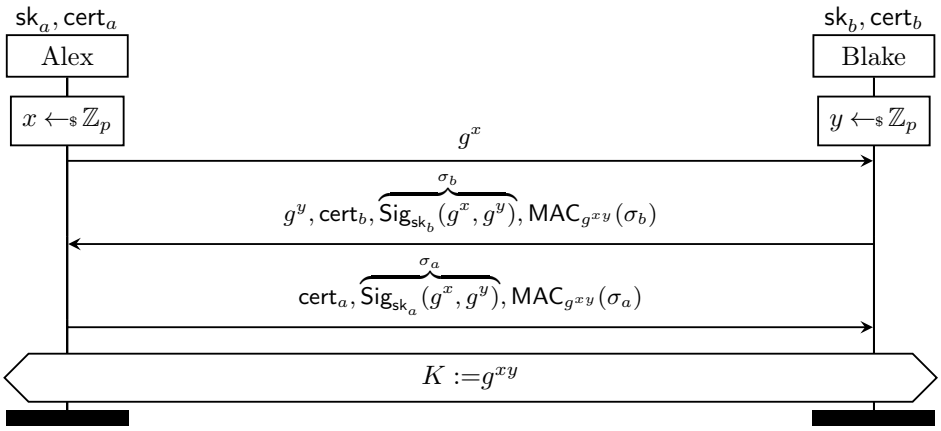



Figure 1: Sketch of the original STS-MAC protocol [41]

current state. Applying a rule changes the state by removing the premises, and adding appropriately instantiated conclusions. The instantiated action facts are the action associated with this rule instance.

An *execution* is a sequence of states starting from the empty set and using *rule instances* to transition from one state to the next. Then, a *trace* is the sequence of ground action facts appearing at the rule instances in a protocol execution.

Security properties are defined in a fragment of two-sorted first-order logic, with messages and timepoints, and quantification is possible over both. The atoms considered are then $\bot$, term (in-)equality `s = s'`, timepoint ordering $t_1 < t_2$, timepoint equality $t_1 = t_2$, or an action *Fact* at a timepoint $t_1$ written $Fact(terms)@t_1$, together with the usual logic connectives. Examples of security properties that can be modeled this way are secrecy, Perfect Forward Secrecy (PFS), Key Compromise Impersonation (KCI), and a large range of authentication properties, all with respect to potentially complex adversary models.

We additionally consider *restrictions* which limit the explored state space. Technically, restrictions are formulas just like security properties, and their semantics ensure that all traces must satisfy the restrictions. If a trace violates any restriction, it is immediately discarded. Restrictions are often used to model conditional rewriting rules, for example, to model inequality checks and that an action happens only once. More complex uses can enforce a passive (possibly message reordering) adversary by ensuring each received message by a protocol participant was previously sent by a different participant, and is only received one time. For full details on Tamarin see [65].

## 2.4 Running example: STS-MAC

We use Station-to-Station (STS) [41] as a running example, since it is a simple and well-understood authenticated key exchange protocol. It has the additional benefit of a long history of being exploited using surprising signature properties like those we consider, see [26]. In STS (specifically the STS-MAC variant), two parties exchange Diffie-Hellman (DH) ephemeral keys with a signature and a MAC in order to derive a shared symmetric key, as shown in Figure 1.

We write $x \leftarrow\!\!\$\ \mathbb{Z}_p$ to denote drawing a random number. $\mathsf{Sig}_{\mathsf{sk}_a}(t)$ means signing the term $t$ with the private key $\mathsf{sk}_a$ associated with the public key $\mathsf{pk}_a$. We assume the parties can authenticate each other's public key, for example through a certificate signed by a trusted third party, which we denote $\mathsf{cert}_a$. We assume that before signing the public key, the trusted third party verifies ownership of a corresponding private key.

In the usual symbolic notation signing becomes `sign(t,ska)` for a private key `ska` (representing some $\mathsf{sk}_a$) for which the verification key is `pk(ska)` (representing the corresponding $\mathsf{pk}_a$). Similarly, by $\mathsf{MAC}_{g^{xy}}(\sigma_a)$ we mean the MAC created with key $g^{xy}$ for term $\sigma_a$. The protocol's intended result is that both parties share the key $K$.

After a key agreement protocol like this completes, we expect some security properties to hold about the resulting key. In our verification in the following sections, we will consider the properties: *key secrecy*, *identity agreement*, and *strong session agreement*. We interpret key secrecy to mean that when two honest parties finish a protocol run with each other, the resulting key is secret from the adversary. Identity agreement means that, whenever two honest parties agree on a key, they also agree on each other's identity. Finally, strong session agreement means that if two honest parties finish a protocol run with each other, they agree on the transcript of that session. That means that every message sent by A (respectively

B) was received by B (respectively A) and every message that was accepted by A (respectively B), was transmitted by B (respectively A) without alteration.

# 3 Improved Attack Finding in the Symbolic Model

In this section we describe four properties of signature schemes that can lead to real-world attacks: *key substitution*, *malleability*, *re-signing* and *colliding signatures*. We relate each property to the existing computational and symbolic definitions, then develop new symbolic models that can capture those behaviours and attacks not already considered. We stress that all of these behaviours are permitted by the standard definition of signature security (EUF-CMA) and are not the result of implementation mistakes. For each symbolic model we develop, we demonstrate an attack on an example protocol missed by the traditional model.

## 3.1 Key Substitution

In a *key substitution attack*, the adversary is given an existing signature, message and public key, and is able to to construct a new public key (and possibly a new message as well) such that the honest signature will verify under the new public key (and new message).

This area of research has had at least three terminologies: Blake-Wilson and Menezes [26] called such attacks Duplicate Signature Key Selection (DSKS); Menezes and Smart [59] termed them Key Substitution attacks; and most recently Pornin and Stern [64] presented it as exclusive ownership. We follow this latest terminology.

### 3.1.1 CEO

The following property was first noted in [59], but we draw our definition from the later work [64].

**Definition 3.** [64, Def 1] A signature scheme fails to provide *Conservative Exclusive Ownership (CEO)* if there is an efficient algorithm $\texttt{fake}(pk, (sig, m)_i)$ that given a public key and a sequence of message and signature pairs under that key, outputs a key pair $(pk', sk')$ such that $pk \neq pk'$ and $\mathsf{Vf}(pk', m_j, sig_j) = \mathsf{true}$ for some $j$.

Some signature schemes, including ECDSA (where the signatures have a fixed generator) have been proven to satisfy CEO [59]. Other schemes, such as ECDSA (with signature specified generators) and RSA-PSS, do not satisfy CEO and a `fake` algorithm can be constructed for them [26]. Traditional symbolic models of signatures implicitly assume that CEO holds, because they do not include such a `fake` algorithm: each signature in the traditional model can only be verified by the (unique) public key that corresponds to the secret key used for signing.

To model this additional behaviour, we introduce a new abstract function `CEOgen` that models the existence of such an algorithm as formalised within the CEO definition. This function takes as argument a signature, and returns a private key $x$. We then add an equational theory that expresses that if $x$ is output by the `CEOgen` function, then the corresponding public key $pk(x)$ can also be used to successfully verify the corresponding signature. The CEO inequality $pk \neq pk'$ is given by construction, because the terms representing the two public keys are distinct.

```
functions: CEOgen/1
equations: verify(sign(m, sk), m, pk(CEOgen(sign(m, sk)))) = true
```

**No-CEO**: Model for signature schemes that do not satisfy CEO.
(I.e., they allow for DSKS/Strong Key Substitution attacks.)

If we add this equational theory, called no-CEO, to Tamarin and rerun our analysis of STS-MAC, Tamarin immediately discovers a Unknown Key Share (UKS) attack, violating identity agreement. In the attack, two parties A and B establish the same session key, but have different assumptions on whom they are talking to: A believes they share the key with B, but B believes they share the same key with a corrupted agent E. Thus, if A later receives a message from B encrypted with the shared key, they will incorrectly assume it was intended for them, although B believed they were sending it to E.

This attack was first reported by Blake-Wilson and Menezes [26] where they described it as a DSKS attack. It has also been referred to as "Strong Key Substitution" by [30, 59]. The adversary waits until A sends the final message to B, then using the no-CEO property produces a new public key, registering it with the Certificate Authority (CA), for which A's signature will verify. Note that the adversary cannot

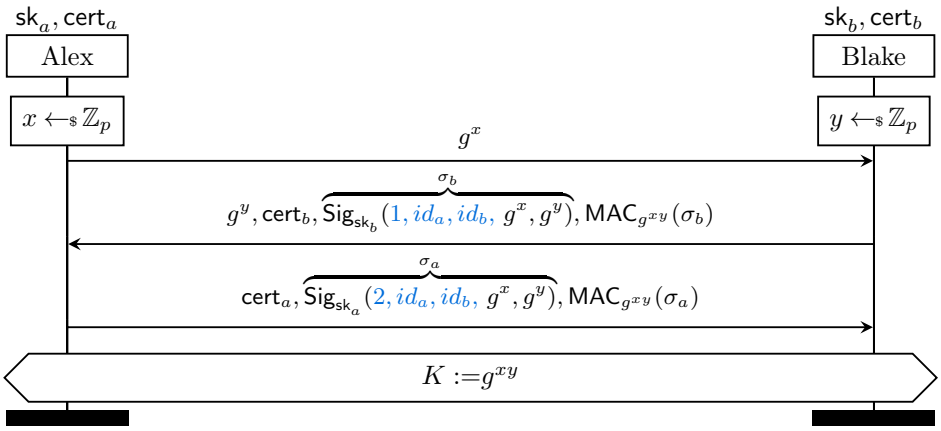



Figure 2: Sketch of STS-ID [26], which includes the identities of the communicating parties in the MAC.

just replace the signature directly, since it is protected by the MAC, to which the adversary does not know the key. The adversary then replaces the associated certificate from A with their own. Consequently, B concludes they are talking to the adversary, even though the key is shared with A and in fact secret from the adversary.

Blake-Wilson and Menezes [26] also suggested the fix of including the identities under the signature, depicted as STS-ID in Figure 2. If we model this patched protocol in Tamarin together with our CEO-falsifying equation, Tamarin successfully proves each property, i.e., secrecy, identity agreement, and strong session agreement, thus verifying the fix in this model of signatures.

### 3.1.2 DEO

It was later discovered [13] that the adversary could also change the *message* that the signature verifies for and, more troubling, that this appears to be possible in practice whenever the original attack was possible. We mark in red the differences between the following definition and that of CEO.

**Definition 4.** [64, Def 2] A signature scheme fails to provide *Destructive Exclusive Ownership (DEO)* if there is an efficient algorithm $\texttt{fake}(pk, (sig, m)_i)$ that given a public key and a sequence of message and signature pairs under that key, outputs $(m', pk', sk')$ such that $pk', sk'$ are a key pair, $pk' \neq pk$, $m' \neq m_j$, and $\mathsf{Vf}(pk', m', sig_j) = \mathsf{true}$ for some $j$.

This is equivalent to Message Key Substitution [59]. [64] also defines Universal Exclusive Ownership (UEO) as the combination of both CEO and DEO.

So, to model this symbolically, we adapt the no-CEO definition in the following way. We first note if there exists a `fake` function as in the DEO definition, then the adversary can choose a second, distinct message $m'$. Unlike in the $pk'$ case, there the distinctness does not follow from the construction. We cannot directly model this distinctness by using equational theories. We therefore model it through Tamarin's support for restrictions and rules, which enable a form of conditional rewriting. A standard restriction is `Neq(x,y)`, which only enables the transition if $x \neq y$. We add an equation as before, but mark the function that models `fake` as a *private function*. The adversary cannot apply private functions itself. Instead, we give the adversary access to the function through a rule, which enables us to enforce the restriction. This leads to the following model:

```
functions: DEOgen/2 [private]
equations: verify(sign(m1, sk), m2, pk(DEOgen(m2,sign(m1, sk)))) = true

rule make_DEO_sk:
[In(<m2,sign(m1,sk)>)]
--[Neq(m1,m2)]->
[Out(DEOgen(m2,sign(m1,sk)))]
```

**No-DEO**: Model for signature schemes that do not satisfy DEO.

After adding this equation and rule to Tamarin, we rerun our analysis on STS-ID and discover a UKS attack. This attack was reported in a short paper by Baek and Kim [13]. The attack proceeds in much the same way as the attack on the original STS-MAC protocol. The adversary waits until A sends the final message and then produces a public key for which A's signature will verify, but for a message altered to include the adversary's identity.

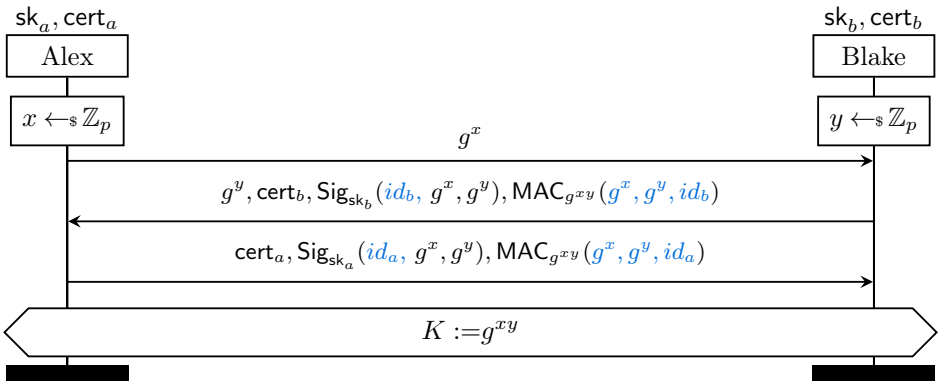



Figure 3: Sketch of STS-ISO [26], which includes the identities of the communicating parties.

Chevalier and Kourjieh [32] considered the decidability of protocol verification in the context of CEO and DEO, but neither implemented their algorithm nor considered any other properties. Interestingly, some years later, STS-MAC and STS-ID were proven to be secure against UKS attacks using the traditional symbolic model of signatures in [65]. This was possible since the traditional symbolic model did not take these signature behaviours into account, thereby missing the previously published UKS attacks [13, 59].

Baek and Kim [13] recommended adoption of an alternative variant of STS-MAC, dubbed 'key agreement mechanism 7' in ISO/IEC 11770-3 [48], which we show in Figure 3 as STS-ISO. Here, instead of including the identities of the communicating parties under the signature, they are directly MACed. We analyse this protocol in Tamarin with respect to these DEO and CEO equations and find the protocol is proven secure under this model. We return to the security of this STS variant in the next section.

## 3.2 Malleability

Signature schemes that are provably secure in the standard sense of being EUF-CMA may still be *malleable*. If a signature scheme is malleable, successful verification does not preclude that the signature was modified. In contrast, non-malleability implies that if a signature is verified under an honest public key, the signature is the same as one produced by the honest party.

Non-malleability is not implied by the standard forgery definition, which only describes the difficulty of producing a signature which verifies under a different message. For example, ECDSA [67] and EdDSA are malleable. Interestingly, whether or not this poses a problem is the subject of dispute between signature scheme designers and implementers. For example, Ed25519 was originally designed without regard for malleability [19], whereas the IETF standardisation body decided to explicitly require implementations to enforce non-malleability [52].

In practice, the design of security protocols may implicitly rely on the assumption of non-malleability, while their instantiation may only use a EUF-CMA-provably secure signature scheme.

**Symbolic model for malleability:** We provide an additional capability to the adversary allowing them to make a new signature out of an old one. This can be done in several straightforward ways; the main insight is to ensure to explicitly express the "malleable" part of the signature construction. Thus, the adversary cannot change arbitrary parts of the signature, since that would break the normal assumptions, but only the malleable part which otherwise does not affect unforgeability.

This can be modeled in Tamarin's framework by extending the term model for signatures with an additional argument, abstracting the malleable information. The signature convention then becomes

$$sign(m, rep, sk)$$

where $m$ represents the signed payload data, $sk$ the signing key, and $rep$ the malleable format. The corresponding verification remains similar to the existing one, in the sense that it ignores $rep$ and works as before on $m$ and $sk$. We also provide the adversary with an operation that allows them to change the malleable part:

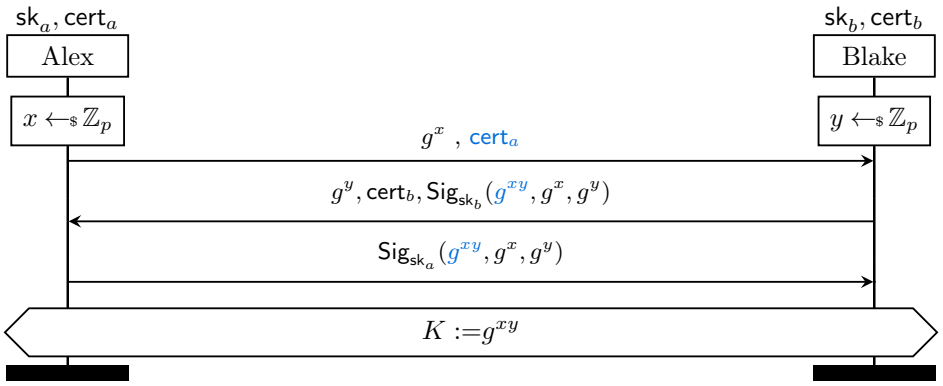



Figure 4: Sketch of STS-KSIG, a synthetic variant of STS-MAC which fixes the identities in the first message but drops the MAC in favour of signing the secret.

```
functions: mangle/2
equations: mangle(sign(m,r,sk),repnew)= sign(m,repnew,sk)
```

**Malleability**: Model for malleable signature schemes.

This additional capability enables the adversary, given a single signature, to produce an arbitrary number of different ones, that all verify under the same public key and message.

Using this model, we analyse STS-ISO. Now the property of strong session agreement fails, as the adversary can alter a signature whilst it is in flight, but both parties believe it is valid. Consequently one party accepts a message which was not transmitted by the other, breaking agreement.

Unlike the other properties in this section, malleability has long been accepted as problematic behaviour, leading to the introduction of a stronger definition of signature scheme security, Strong Existential Unforgeability under an Adaptive Chosen Message Attack (SEF-CMA) [5, 31]: instead of finding a new *message* with corresponding signature, the adversary just has to find a new valid *pair*, and may reuse a queried message. Signatures with SEF-CMA are not malleable, and implementing STS-ISO with such a scheme would provide strong session agreement. In §3.5 we analyse a protocol which achieves this without requiring a SEF-CMA scheme.

## 3.3 Re-signing

As we saw in §2.2, traditional symbolic models have considered the distinction between message-revealing and message-hiding signatures. However, in some signatures schemes a signature over a message reveals the hash of that message, preserving secrecy but allowing an adversary to re-sign the hashed messaged under their own key. We model this by providing the adversary with the explicit capability to re-sign signatures, even if the signature is message-hiding and the message is secret.

```
rule: ReSign
[ In( sign(m,r,sk1), sk2 ) ]
-->
[ Out( sign(m,r,sk2 )) ]
```

**Re-sign**: Model for re-signing an unknown message.

To illustrate the implications, we introduce the following synthetic variant of STS-MAC which we call STS-KSIG and present in Figure 4. Here, identities are fixed in the first message. However, the protocol has dropped the MAC value in favour of directly signing the secret value.

Analysing this protocol with the key substitution model finds no attacks, since the responder's view initiator's key is fixed before the initiator discloses a signature. However, when we add the re-signing equation we immediately discover an UKS attack, as the adversary can form a new signature on the secret key from Blake's response, consequently claiming Blake's DH public key as their own, despite not knowing the shared key. Much like the UKS attacks on STS-MAC and and STS-ISO, this attack violates identity agreement.

## 3.4 Colliding Signatures

Stern et al. [67] give an algorithm to produce a signature and public ECDSA key against which *two* messages of the adversary's choice both verify. Ed25519 allows this behaviour to a much greater degree: selecting signature and public key values from the order-eight subgroup leads to verification passing for *any message* with high probability. The design paper for Ed25519 [19, Page 7] explicitly notes this behaviour and argues it is not problematic, while implementations are split: LibSodium rejects low order elements [51], but Go's standard library currently accepts them [46].

**Definition 5.** We say a signature scheme has *non-colliding signatures* if it is computationally infeasible for an adversary to produce a public key and signature combination such that verification of more than one message succeeds with non-negligible probability. I.e., it is infeasible to select private and public keys and a signature, $sk, pk, s$, such that there exist messages $m_1, m_2$, for which $\mathsf{Vf}(s, m_1, pk) = \mathsf{true}$ and $\mathsf{Vf}(s, m_2, pk) = \mathsf{true}$.

Colliding signatures violate two implicit properties that protocol designers sometimes rely on:

(i) if a signature verifies against a particular public key and message, then the signer knew the message that was signed; and

(ii) for a given signature and public key, there exists a unique message which will verify under it.

```
functions: weak/1
equations: verify(sign(m1,r,weak(x)),m2,pk(weak(x))) = true
```

**Colliding Signatures**: Model for colliding signatures.

This model allows for the worst case behaviour, where a particular signature and public key will verify for any message without requiring the adversary to pick the messages they wish to collide in advance. We consider a (synthetic) variant of the previous protocol where the signatures are encrypted under the recipient's public key, dubbed STS-SCRYPT and shown in Figure 5, using the equational theory for colliding signatures. STS-SCRYPT patches STS-KSIG to prevent the adversary from re-signing a message; as well, the key substitution equations cannot be applied as there is no visible signature for the adversary to steal.

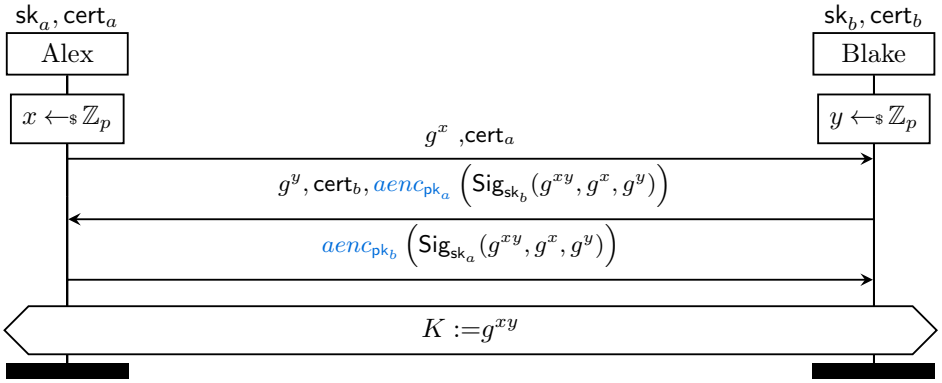



Figure 5: Sketch of STS-SCRYPT, which patches STS-KSIG by asymmetrically encrypting the signatures.

However, running Tamarin with our colliding signature equation reveals an attack. The adversary can simply register a colliding public key and asymmetrically encrypt their own colliding signature. The resulting value will verify with high probability, even though the adversary does not know the message being signed.

## 3.5 Fixing STS-MAC

An alternative to STS-ID was proposed in [26], which we refer to as STS-KDF and show in Figure 6. STS-KDF works just like STS-MAC but then uses a Key Derivation Function (KDF) to bind the shared key to the identities of the participants, $KDF(g^{xy}, id_a, id_b)$, instead of using only the shared Diffie-Hellman secret $g^{xy}$.

Noting that KDFs were poorly understood (at the time), the authors explicitly recommended using STS-ID over STS-KDF. It has been nearly two decades since their paper was published and we can now say that KDFs have stood the test of time. We analyse STS-KDF in Tamarin, allowing the adversary to

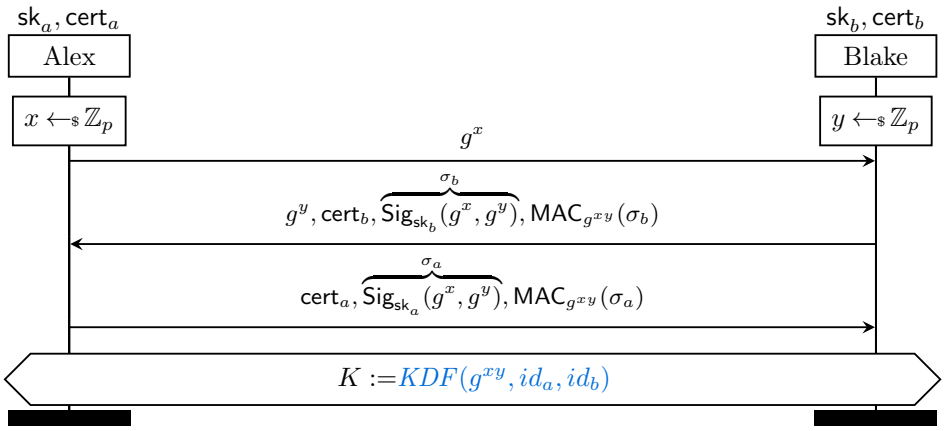



Figure 6: Sketch of STS-KDF [26], which patches STS-MAC by including the identities of the communicating parties in the KDF.

use all of the new properties we have discussed in the previous section and find that Tamarin proves that all three properties hold, making this protocol the only STS variant we have considered which satisfies all three security requirements (in this signature model) whilst only using EUF-CMA signatures. However, can we be sure no additional signature equations exist which will break this protocol? We return to this question in §4.

### 3.6 Summary

We summarise our analysis results in Table 2. We note that the traditional symbolic model fails to find any of the attacks we have discussed here. Our attack finding models only slightly increase running times, which implies the approach is tractable. This is better than we expected, considering the additional behaviours that Tamarin must consider, and that we did not introduce any new heuristics for this model.

We have seen several properties of signature schemes, none of which are implied by EUF-CMA, the traditional (and still most common) definition of signature scheme security, nor are they the result of implementation mistakes. In practice, many signature schemes in fact do not meet these properties: in Table 1 we gave a list of widely-used signature schemes and whether the subtle behaviours are present.

In contrast, the existing definitions of symbolic signatures in current tools implicitly assume all of these properties hold, which means that they cannot discover the corresponding attacks. To remedy this, we presented symbolic models for the absence of these properties, which enable finding those "invisible" attacks.

*Full sources to all of our models are available at [49].*

## 4 Improved Symbolic Model for Verification

In the previous section, we characterised a number of behaviours not captured by the traditional symbolic signature model and repaired the deficiency. However, each improvement is ad-hoc, designed to only capture a known behaviour, and provides no assurance that further, more subtle, behaviour has not been omitted. Thus, while the models in the previous section are extremely effective for attack finding, they raise the obvious question for verification: did we model enough, or do we miss more attacks?

In this section, we address this through the development of an entirely new symbolic model for the verification algorithm of digital signatures, directly inspired by the standard computational security definition for signatures and which we call *Symbolic Verification of Signatures* (SVS). It is a symbolic model for signature verification that makes minimal assumptions, relying only on the implications of existential unforgeability.

### 4.1 Specification

Revisiting the definitions of *correctness* and *forgery resistance* from §2.1, we note the behaviour of the verification function is specified only when the public key is honestly generated. To reiterate, the first

requirement is to accept correctly generated signatures for honest public keys:

$$\forall pk, sk, m \in M : (pk, sk) \leftarrow \mathtt{gen}() \Rightarrow \mathtt{verify}(\mathtt{sign}(sk, m), m, pk) = \mathsf{true}$$

The second implies that if a signature can be verified with an honestly generated public key for some message $m$, then $m$ and the matching signing key were previously used to sign:

$$\forall pk, sk, m \in M, s : (pk, sk) \leftarrow \mathtt{gen}() \wedge \mathtt{verify}(s, m, pk) = \mathsf{true} \Rightarrow \text{previously: } \mathtt{sign}(sk, m)$$

We note that whereas `sign` can be a probabilistic algorithm, `verify` is implicitly assumed to be a deterministic algorithm.

There are no further requirements on the output of `verify` given by the standard computational definition. The traditional definition of symbolic verification, shown in §2.2, agrees with the computational definition but further specifies that `verify` implicitly outputs false in any situation in which it is undefined, i.e., verification against a malicious key. Many of the equations we gave in §3 essentially remedied this in an ad-hoc fashion, by specifying additional cases where `verify` could return true.

We now build a symbolic definition of `verify` that agrees with the computational one: where its output is not otherwise constrained, we let the adversary choose whether it returns true or false. This definition encompasses our previous equations for maliciously generated public keys, as well as further unknown equations, so long as they are not ruled out by the computational definition.

In a symbolic setting, we consider traces made by a series of transitions of a labeled transition system, which (implicitly or explicitly) describes the state of the protocol at that point in the trace. When signature verification occurs in a trace, we now require the following constraints to be observed:

1. If the public key was honestly generated, the verification of a corresponding honest signature must succeed.
2. If the public key was honestly generated, the verification of a forged signature must fail.
3. If this particular message-signature-key triple has been verified before, the result is defined by the previous answer.
4. Otherwise, consider all possibilities—i.e., let the adversary decide.

We could model the first and second constraints purely in the term algebra. Similarly, the fourth constraint corresponds to allowing the adversary to send a value over a channel to the protocol. The third constraint is the interesting one because it requires storing (monotonic) state about previous queries. Succinctly, the verification output depends both on the "local query", the history of the trace, and the adversary's current choice. Consequently our symbolic model must record whether public keys have been created honestly, and what verification checks have been made previously. We now give an implementation of this abstract specification in Tamarin.

## 4.2 Tamarin Implementation: User-Visible

We use the function signature as defined in §3.2. We allow for public key and message extraction as the modeller wishes. We omit the verification function and its associated equation, and will replace it with a different mechanism that makes minimal assumptions on the properties of the scheme.

In previous approaches, the signature verification function was encoded into the term algebra, and explicitly stated under which conditions signature verification returns true. Here, we will instead only specify *restrictions* for the signature verification results, and consider all possibilities in other scenarios. To implement this, we specify signature verification as an annotation on a protocol rule, which guards the transition according to a series of first order logical statements we give below. So, we begin by defining the trace annotations that we use for restriction checks. Each of these annotations can be added to Tamarin as an *action* in a mechanical fashion for the rule using it:

**Honest Key generation** Where the *protocol* honestly generates a public key $pk$, we label the corresponding transition with the action $Honest(pk)$. This specifies that the public key has been output by the generation algorithm for signatures and consequently, the various restrictions on the signature verification algorithm will apply if a signature is tested against this public key.

**Signature Verification** Where a protocol will only make a certain transition conditional on the result of a signature verification result (be it true or false), we will provide an action label for this occurrence. Unlike the $Honest(pk)$ label, we will later use *restrictions*, first order formulae, to restrict situations under which this transition can occur. When a protocol wishes to verify a particular signature

term *sig* against a particular message and public key combination (labelled *tm* and *tpk*), we will write $\mathit{Verified}(\mathit{sig}, \mathit{tm}, \mathit{tpk}, \mathit{result})$ where *result* may be true or false depending on whether the transition should be allowed to occur.

**Manipulation of Honest Signatures** We also provide the equation for malleable signatures and the rule for re-signing we discussed in §3.2 and §3.3. Malleability allows an adversary to manipulate an honest signature and is therefore not part of our improvements to the signature verification algorithm. Re-signing has a very subtle usage for the adversary: if (i) the adversary compromises the private key of an honestly generated key pair, (ii) the signature theory is not message revealing, and (iii) the adversary is in possession of a signature for an unknown message, then the adversary can use the re-signing rule to generate a new signature, under the compromised honestly generated key, for the unknown message. As this refers to the generation of a new signature under an honestly generated public key, it is orthogonal to our changes to the specification of signature verification for malicious public keys.

## 4.3 Tamarin Implementation: Internal

**Syntactic Transformations** Behind the scenes, we will perform a mechanical transform of $\mathit{Verified}(\mathit{sig}, \mathit{tm}, \mathit{tpk}, \mathit{result})$ into an action fact $\mathit{Verified}(\mathit{sig}, \mathit{sm}, \mathit{spk}, \mathit{tm}, \mathit{tpk}, \mathit{result})$ using the extraction functions described in the listing below. We define $\mathit{sm} = e1(\mathit{sig})$ and $\mathit{spk} = \mathit{pk}(e3(\mathit{sig}))$. This transformation is needed for purely technical reasons: Tamarin requires reducible functions to be specified in the action fact annotation rather than in restriction formulae.

```
functions: e1/1, e2/1, e3/1 [private]

equations: e1(sign(x,y,z)) = x
           e3(sign(x,y,z)) = z
```

**Extraction Functions for Signatures**. These are not used by the protocol or the adversary, just by the implementation.

These functions allow us to easily refer to the message and public/private key that a signature corresponds to. Note that in the event the signature is not honestly generated, these functions are still well defined, but simply do not yield a result (technically, they will not reduce).

We now provide a series of *restrictions* which restrict the traces that can occur. All of our restrictions concern the behaviour of the signature verification function.

**Correctness** This requirement follows directly from the requirement that an honestly generated public key, an honestly generated signature, and the correct message must verify as true.

$$\texttt{Correctness} : \forall \mathit{sig}, \mathit{tm}, \mathit{tpk}, t_1, t_2.\ \mathit{Honest}(\mathit{tpk})@t_1 \wedge \mathit{Verified}(\mathit{sig}, \mathit{tm}, \mathit{tpk}, \mathit{tm}, \mathit{tpk}, \mathsf{false})@t_2 \implies \bot$$

**NoForgery** Here we state that if a signature verification does succeed against an honest public key, then the signature must have been honestly produced.

$$\texttt{NoForgery} : \forall \mathit{sig}, \mathit{tm}, \mathit{tpk}, \mathit{sm}, \mathit{spk}, t_1, t_2.\ \mathit{Honest}(\mathit{tpk})@t_1 \wedge \mathit{Verified}(\mathit{sig}, \mathit{sm}, \mathit{spk}, \mathit{tm}, \mathit{tpk}, \mathsf{true})@t_2 \implies \mathit{sm} = \mathit{tm} \wedge \mathit{spk} = \mathit{tpk}$$

**Consistency** Verification is typically defined as a deterministic function, here we specify that repeated calls to verify will always return a consistent answer.

$$\texttt{Consistency} : \forall \mathit{sig}, \mathit{sm}, \mathit{spk}, \mathit{tm}, \mathit{tpk}, r_1, r_2, t_1, t_2.\ \mathit{Verified}(\mathit{sig}, \mathit{sm}, \mathit{spk}, \mathit{tm}, \mathit{tpk}, r_1)@t_1 \wedge \mathit{Verified}(\mathit{sig}, \mathit{sm}, \mathit{spk}, \mathit{tm}, \mathit{tpk}, r_2)@t_2 \implies r_1 = r_2$$

The result of this model is that if a particular transition is labeled with a verification annotation it will be allowed to occur unless it violates one of these three restrictions.

If we compare these restrictions to our earlier specification, we note the following: in the event that the signature is being verified against an honest public key, *Correctness* ensures honest signatures will be accepted and *NoForgery* ensures forged signatures will be rejected. Otherwise, the only rule

that will apply is *Consistency* which simply ensures signature verification calls are deterministic. It is straightforward to see how this presentation matches our earlier specification, as this expresses our required properties to Tamarin directly. Consequently, this *Symbolic Verification of Signatures* (SVS) model allows the adversary to perform key substitution attacks, craft colliding signatures and many other known or unknown behaviours concerning maliciously chosen public keys.

### 4.4 Results

| STS-MAC variant | Signature Model | Security property Sec | IA | SSA | Time in seconds |
|---|---|---|---|---|---|
| MAC | Traditional | ✓ | ✓ | ✓ | 14 |
| | no-CEO | ✓ | • | • | 35 ∗ |
| | SVS | ✓ | • | • | 23 ∗ |
| ID | Traditional | ✓ | ✓ | ✓ | 14 |
| | no-DEO | ✓ | • | • | 68 ∗ |
| | SVS | ✓ | • | • | 16 ∗ |
| KSIG | Traditional | ✓ | ✓ | ✓ | 13 |
| | Re-sign | ✓ | • | • | 46 ∗ † |
| | SVS | ✓ | • | • | 30 ∗ |
| SCRYPT | Traditional | ✓ | ✓ | ✓ | 16 |
| | Coll. | ✓ | • | • | 25 ∗ |
| | SVS | ✓ | • | • | 23 |
| ISO | Traditional | ✓ | ✓ | ✓ | 17 |
| | Mall. | ✓ | ✓ | • | 34 |
| | SVS | ✓ | ✓ | • | 25 |
| KDF | Traditional | ✓ | ✓ | ✓ | 3 |
| | All in §3 | ✓ | ✓ | ✓ | 19 |
| | SVS | ✓ | ✓ | ✓ | 9 |

Table 2: Verification results when applying our various Tamarin models to a number of distinct STS-MAC variants.
Sec, IA and SSA are respectively the security properties of key secrecy, identity agreement and strong session agreement.
✓ indicates that Tamarin successfully verified the property
• indicates that Tamarin found an attack
∗ indicates that attack finding was done in the bounded model
† indicates that the non-default `i` heuristic was used for the proof

We now show that the above model is tractable in practice and we present results and running times in Table 2.

This model, as close as it is to the computational definition, forces us to consider issues not normally raised in a symbolic analysis. Traditional symbolic tools often produce an attack trace that is practical in reality, as it consists of a series of explicit capabilities provided to the adversary. In contrast, our SVS model is closer to the computational model in the sense that the attack trace will consist of the adversary specifying certain signatures pass or fail verification, but providing no intuition on how an adversary may arrange for this to happen.

Consequently, both our SVS model and our earlier models for attack finding are independently of interest to protocol modellers. First, the SVS model (§4) should be used and if Tamarin returns a proof, it is within the strongest model of signature security we have described. However, should it return an attack trace, the modeller can use our *attack finding models* (§3) to effectively recover practical attacks that could be used in reality. By using the attack finding model for each property separately, it is possible to isolate the signature behaviour which is leading to the attack and thus consider possible mitigations. We demonstrate this functionality on our case studies in the next section.

It is possible that our SVS model returns that an attack is found, but none of our falsification models yields an attack. In this case, we suggest it is best to think of the result as "Not Proven". There may well

be an attack as the protocol requires behaviour of signatures not provided by the standard definition, yet Tamarin is not aware of a method of crafting public keys or signatures to enable the attack in practice. This means one does not have the desired symbolic proof, but still gets the proofs for the falsification models, which is a much stronger guarantee than previously.

# 5 Further Case Studies

We now demonstrate the utility of our approach on more complex case studies. We automatically find three attacks on different real world protocols, two of which are novel and previously unreported. Furthermore, all three protocols have undergone previous formal analysis using the traditional model of digital signatures and reported to be secure:

(i) a known attack on an earlier version of the Let's Encrypt certificate issuance protocol, arising from a key substitution property;

(ii) a previously unreported attack on a WS-Security Handshake, arising from a key substitution property. This attack was missed despite having been analysed using ProVerif in two separate papers.

(iii) a previously unreported attack on DRKey, a key exchange protocol, arising from a weak signature property. This last attack was missed in a previous formal analysis, and allows us to violate the security claims of OPT, an origin and path tracing protocol which uses DRKey.

## 5.1 Let's Encrypt

Let's Encrypt (LE) is the world's most popular Certificate Authority. To issue certificates automatically, it uses the ACME protocol for issuance, renewal and revocation, as standardised by the Internet Engineering Task Force (IETF) [10]. ACME allows a website owner to prove ownership of a domain and request a certificate from a CA via a choice of signature-based challenge-response protocols. If the protocol succeeds, LE issues a certificate to the owner.

ACME went through a number of drafts prior to (and after) release. Draft Barnes 01 [1] was the first incarnation in which DNS challenges are completed by placing a nonce in a DNS record. The DNS challenge mechanism was then updated in Draft Barnes 03 [2] to be a signature over the nonce by the account holder's public key.

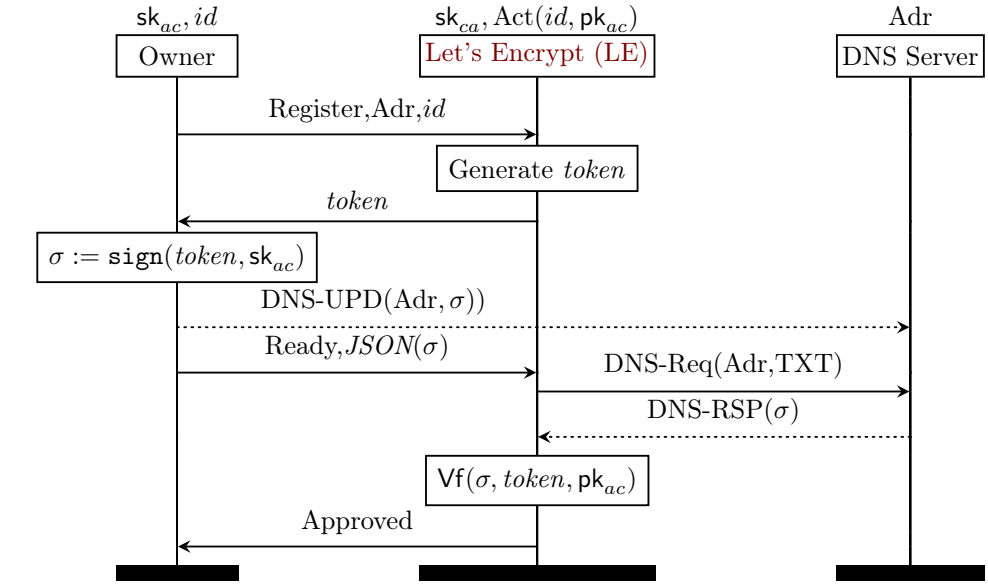
Draft Barnes 04 [3], only a minor refinement of 03, was then adopted by the ACME working group as IETF Draft 00 [8], at which point—only six weeks before LE was loaded into major browsers' certificate stores—a signature key substitution attack was discovered and reported [4] to the IETF ACME mailing list. (IETF ACME Draft 00 is also known as Barnes Draft 04.) The attack allowed an active attacker to pass the ACME challenge and receive a valid TLS certificate for any website using LE DNS Challenges, and thus intercept and modify any such website's TLS traffic. This prompted the DNS Challenge to be updated to a hash of the account public key and the nonce (known as a key thumbprint) in IETF Draft 02 [9]. This mechanism remains in use today [10].

The attack stems from Draft 00's use of a DNS-based signature challenge, shown in Figure 7a: the website owner requests a random nonce from LE, signs the nonce with a key to be used in the new certificate, and places the signature in a DNS record for the domain. LE then extracts and verifies the signature from the website's DNS records, concluding that the owner controls (i) the claimed private key and (ii) the DNS records for that website. Based on that conclusion, it issues a certificate for the corresponding public key.

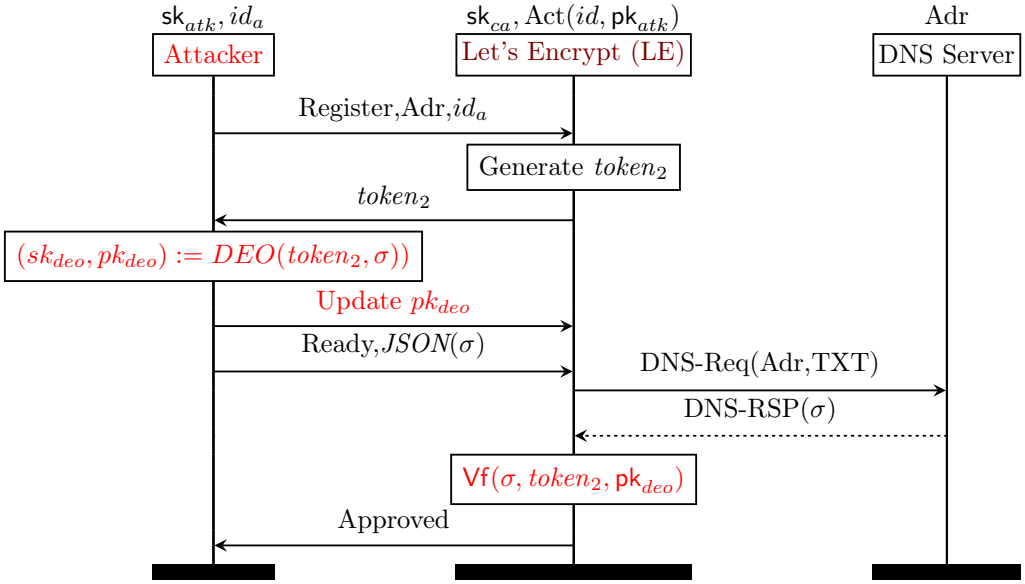
In the attack, depicted in Figure 7b, suppose Alex has completed a LE ACME challenge as normal, and has placed the signature in their DNS records. The adversary can begin a new instance of the challenge response protocol with the CA, claiming ownership of Alex's website and receive a token to sign and display in Alex's DNS records. The adversary then performs a key substitution (no-DEO) attack on Alex's signature and the new token value (and updates their account key accordingly). Afterwards, they trigger the second phase of the protocol by sending the Ready message. LE retrieves Alex's signature and verifies it against the adversary's malicious public key. This succeeds and LE will now issue the adversary a certificate for Alex's website and the adversary's public key.

### 5.1.1 Analysis of ACME

We developed a Tamarin model of the vulnerable draft of the ACME certificate issuance protocol. Using our model from §4, we automatically find the reported attack. We check that using the traditional symbolic model of signatures Tamarin successfully verifies ACME, confirming that it misses this attack




(a) Normal Operation of the ACME Draft 00 Protocol




(b) Attack on the ACME Draft 00 Protocol

Figure 7: ACME Draft 00 Let's Encrypt DNS Challenge Response Protocol. The dotted arrows indicate that the channel is assumed to be authentic.

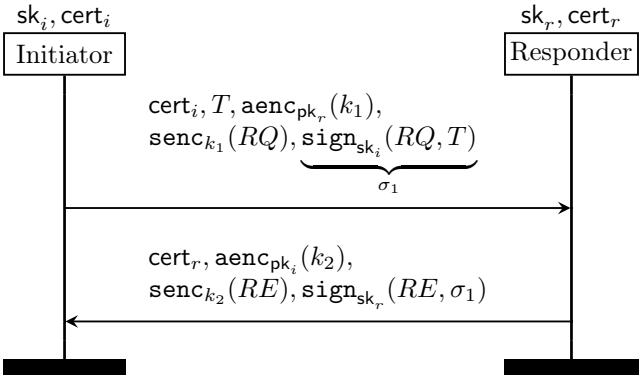
without our improvements. We also provide a Tamarin model corresponding to IETF ACME Draft 02 [9], the patched version of ACME. Although the IETF could have elected to use a signature scheme which provides DEO, they it felt it safer to forgo the use of signatures entirely, instead replacing the signed value with a hash of the account public key and the token. Using our SVS model, Tamarin verifies the attack is no longer possible. We collect these results in Table 4.

This example also illustrates the complementary uses of SVS and our attack finding models, such as 'no-DEO'. Whilst SVS reports an attack, the attack trace does not correspond exactly to the attack reported on the mailing list [4]—rather, the trace simply allows an adversary to successfully pass the verification directly, since this possibility is not excluded by the EUF-CMA definition. If we then use our 'no-DEO' equation from §3, Tamarin recovers the exact attack trace from the initial report. This demonstrates the utility of our two-pronged approach.

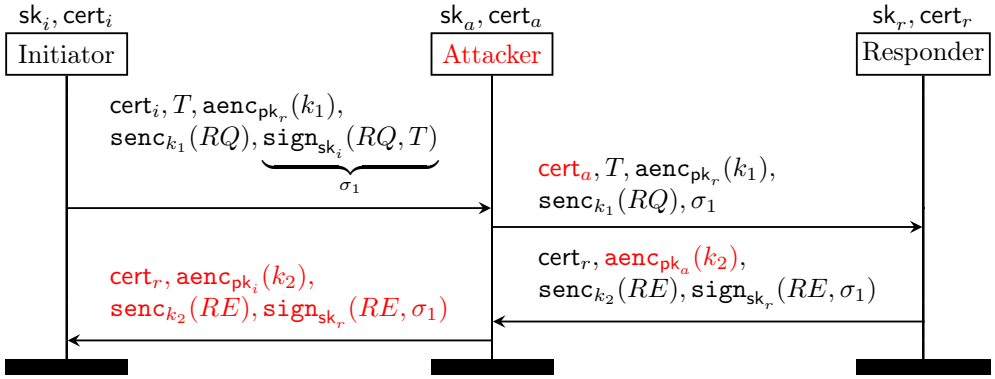
Previously, Bhargavan, Delignat-Lavaud and Kobeissi [21] presented a symbolic model of draft Barnes 01 [1] (which they refer to as ACME Draft 1) and draft IETF 00 [8] (referred to as ACME Draft

4) using ProVerif. Due to the traditional symbolic signature model, their analysis missed this attack. In fact, their analysis concluded the (vulnerable) draft IETF 00 satisfied stronger security properties than the earlier (secure) draft Barnes 01, which contradicts our findings.

## 5.2 WS-Security




(a) Normal Protocol Flow. $T$ is a timestamp used to prevent replay attacks, $RQ$ is the request payload and $RE$ is the response payload.




(b) This attack violates request correlation and response secrecy. The Attacker passes off the Initiator's request as their own by replacing the certificate, can then learn the response and can even pass it back to the Initiator. Note that the responder does check the match of signature $\sigma_1$ and certificate $\mathsf{cert}_a$, but is fooled due to no-CEO.

Figure 8: The WSS1.1-MA-X509-SE protocol from [73] and the attack we automatically discovered.

In 2004, the OASIS Consortium published the Web Services Security Standard [61], which defines a suite of protocols for securing XML web requests and responses without requiring the use of TLS (which was not yet widely deployed). This standard enjoyed considerable popularity until it was overtaken by SAML and later TLS based solutions. Nonetheless, it is still in use and supported by many enterprise frameworks such as gSOAP [44], Apache CXF [6], IBM Websphere [71], and Microsoft's WCF [60].

As well as suffering from a number of implementation flaws, primarily due to the complexities of XML parsing and canonicalisation [72], the complexity and popularity of the standard made it of considerable interest to the automated verification community [12, 22, 23, 24, 25], leading to the creation of verified cross compilers which could accept a protocol specification from the standard and produce both an automated proof of security using ProVerif and an executable implementation in F# [25].

The 1.0 standard published in 2004 was later superseded by the 1.1 standard released in 2006 [61]. One of the motivations for the updated standard was the introduction of *Signature Confirmation*, a mechanism for correlating requests and responses to prevent adversarial manipulation [23]. The principal idea of Signature Confirmation is that after receiving a signed request, the responder's signature should also cover the signature from the request.

Although the standard only directly defined a method for specifying particular message formats and how to parse them, a number of example handshakes and 'scenarios' were also provided. One such scenario which saw widespread adoption was WSS1.1-MA-X509-SE [73] which is depicted in Figure 8a. It supports a request response framework where each party holds a X.509 certificate and corresponding

private key and claims to provide mutual authentication of the communicating parties, as well as binding requests and response together securely using signature confirmation. In addition to being the default setting in IBM's Websphere Platform [71], documentation of its use as a default can still be found for the Spring Framework [66], the Windows Communication Foundation [70], Oracle's Fusion Middleware [62] and Apache CXF [7].

In 2006, a team at Microsoft Research verified the design of this protocol and the benefits of signature confirmation using ProVerif [22]. In addition to proving the secrecy of requests and responses made in the protocol, they also proved 'request correlation', that every accepted response matched the intended request. This analysis was also followed up on in [25], where it was dubbed 'WS Request-Response' and the authors presented a tool for extracting ProVerif models of the protocol from F# implementations.

Using our new model for signature verification, we revisit this protocol in Tamarin. We automatically discover a number of attacks, the most devastating of which makes use of the no-CEO property to 'steal' a client's request and is depicted in Figure 8b. Not only does this violate the request correlation property that signature confirmation was introduced to ensure, but furthermore the attacker can learn the contents of the response to the honest request, violating the secrecy requirement.

There are many scenarios in which this would be damaging, notably if the request contained login credentials and the response a cookie or other secret authentication response. The previous analyses in ProVerif could not have discovered this attack, as they used the traditional symbolic model of signatures, which does not consider these types of attacks.

We stress that whilst we demonstrate our attack on this particular protocol, it is the very mechanism of signature confirmation which is flawed. Signing a signature does not (necessarily) create a unique binding to the contents or public key of the signed signature. Instead, it is much better practice to directly sign the original message and original public key. Using our SVS model, we verify that this proposal fixes the security issues in the original protocol.

### 5.3 DRKey and OPT

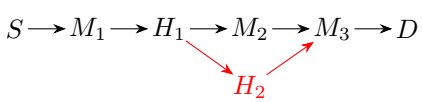
The "Dynamically Recreatable Key" Protocol (DRKey) was first published in 2014 [55] and was supported by a mechanised proof performed using Coq [74]. It is a lightweight key exchange protocol for routers on a packet-switched network to agree on symmetric keys, used as part of a secure routing architecture.

At a high level, DRKey participants generate directional symmetric secret keys, one for use with each other participant. They send both a public-key encryption and signature of the key to the recipient, thereby securely transporting and authenticating the keys to other participants. These keys are then used as part of a higher-level protocol called "Origin and Path Trace" (OPT) [55]. OPT aims to prevent malicious routers from altering the paths of packets through a wider network, using the keys generated by DRKey to authenticate each link in turn. One of OPT's security goals is that malicious routers should only be able to affect routes to their immediate neighbours:

> "When there are multiple **adjacent** malicious nodes on the intended path, a wormhole is present: an honest node down the path can only conclude that the packet has entered the hole via the first malicious node and exited from the last malicious node."

Zhang et al. [74, Section 6.2] presents a formal analysis and claims that this non-collusion property holds. We automatically find a previously unreported attack on this property with Tamarin. We also show that using the traditional model of digital signatures leads to a successful Tamarin verification which misses our attack.

We describe the attack using an example topology, in which $S$ and $D$ are an honest source and destination, $H_1, H_2$ are honest routers, and $M_1, M_2, M_3$ are malicious routers. $S$ wishes to send a packet to $D$ along the intended upper path shown in black. $H_2$ is an honest router, not on the intended path; the malicious routers collude to route the packet through $H_2$ on the lower path (in red) while $S$ and $D$ believe that it took its intended route via $M_2$. This violates the security requirement we quoted earlier, which requires that the packets travel the edge $H_1 \to M_2$, whereas due to our attack they will instead transit $H_1 \to H_2$.



The attack arises because of the ability to re-sign *secret* messages under a new key. The message used to pass keys in DRKey is:

$$\mathtt{aenc}_{\mathtt{pk}_{S,t}}(K_{H_2,S}), \mathtt{sign}_{\mathtt{sk}_{H_2}}(K_{H_2,S}, \mathtt{pk}_{S,t}, S)$$

Here we show the message produced by $H_2$, carrying keys intended for $S$ in the session using the temporary public encryption key $\mathtt{pk}_{S,t}$. During the DRKey protocol, the adversary as $M_1$ can forge a message to $H_2$, claiming that $S$ wishes to agree on keys for the lower path. When $M_3$ receives the DRKey message from $H_2$, containing a signature and an encrypted key, the adversary (as $M_3$) can re-sign the packet as if it came from $M_2$ and pass it on to $D$. This is possible even though the adversary does not know the message, and is a behaviour not captured by traditional symbolic models. Note that even if the DRKey implementation uses a signature scheme where re-signing is not possible, e.g., Ed25519, the colliding signature property can also be used to craft a signature for an unknown message. The rest of DRKey proceeds as normal, at the end of which $S$ and $D$ each hold a key they believe they share with $M_2$, but in fact they share this key with $H_2$. This constitutes a UKS attack on the DRKey Protocol.

The OPT protocol then prescribes a series of chained MACs such that honest routers can detect maliciously-routed packets, and that the destination can verify that the correct path was followed. However, in the above context, $S$ intends to route a packet via $M_2$ but $M_1$ can maliciously alter the route to the lower path. Because $S$ and $D$ share a key with $H_2$, that they *believe* they share with $M_2$, neither of them can detect this malicious routing[1]. As a consequence of this attack, $M_2$ could bill $S$ for routing packets, despite in fact offloading all of the transmission work to the unsuspecting $H_2$.

We stress that whilst we demonstrate this attack on the example topology described above, it applies generally to any topology where an adversary can control two (or more) adjacent malicious routers and at least one router earlier in the chain. Besides the double billing attack we mentioned earlier, this attack could also be used to perform a denial of service attack on an honest router, by forcing additional packets to pass through it, despite the fact both source and destination believe their packets are travelling a different route.

As DRKey is intended for use in the SCION [63] internet architecture it is still under active development. The DRKey authors agree that the attack we found is serious and have modified their protocol according to our proposed fix. The prototype is already updated and this will be reflected in an extension of their work, which is currently under submission for publication.

Our proposed fix for this protocol follows the intuition behind STS-KDF. We do not need to change any of the messages on the wire, instead, we apply a key derivation function which binds each key to the identity of the party who is using it, and the party they believe they share it with. This suffices to prevent any unknown key share attacks on DRKey, as honest parties will only agree on keys if they also agree on identities. Using our SVS model of digital signatures, Tamarin verifies the fix in only 7 seconds.

## 5.4 Summary

| | **Previous traditional verification** | | | **Attacks found in this work by adding our new signature models** | | | | |
|---|---|---|---|---|---|---|---|---|
| **Protocol** | **Ref** | **Year** | **Methodology** | **Properties violated** | **Model** | **Time (s)** | **Sect.** | **First reported** |
| X.509 Mutual Auth | [22] | 2006 | ProVerif | Correlation & Secrecy | no-CEO | 5 | §5.2 | *This paper* |
| WS Request-Response | [25] | 2008 | $F\# \rightarrow$ ProVerif | | | | | |
| STS-MAC-fix1 | [65] | 2012 | Tamarin | Authentication | no-CEO | 35 | §3.1.1 | [26] |
| STS-MAC-fix2 | [65] | 2012 | Tamarin | Authentication | no-DEO | 68 | §3.1.2 | [13] |
| DRKey & OPT | [74] | 2014 | Coq | Authentication & Collusion Resistance | Coll. | 2640 | §5.3 | *This paper* |
| ACME Draft 4 | [21] | 2017 | ProVerif | DNS Validation | no-DEO | 53 | §5.1 | [4] |

Table 3: Summary of new findings using our signature models, compared to previous analyses that used the traditional symbolic model for signatures. The previous analyses did not discover any attack on these protocol properties. In contrast, our new approach efficiently finds attacks, including previously unreported ones.

In Table 3 we relate our attacks to previously published academic papers. Notably, we have uncovered previously unknown attacks on real world protocols that have previously undergone formal analysis. Each

[1] Perhaps a simpler form of misbehaviour would be rewriting the route $S \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow D$ to $S \rightarrow M_1 \rightarrow \underline{H} \rightarrow M_3 \rightarrow D$. However, this does not technically contradict OPT's claimed security goals, while our example violates their non-collusion property. The malicious router $M_1$ is necessary in order to change the route of the packet in the first place.

| Protocol | Signature Model | Analysis results | Time in seconds |
|---|---|---|---|
| WS-Security | Traditional | ✓ | 3 |
| | no-CEO | • | 5 |
| | SVS | • | 12 |
| WS-Security (fixed) | Traditional | ✓ | 2 |
| | All in §3 | ✓ | 12 |
| | SVS | ✓ | 13 |
| LE-00 | Traditional | ✓ | 1 |
| | no-DEO | • | 53 |
| | SVS | • | 98 |
| LE-02 | Traditional | ✓ | 1 |
| | All in §3 | ✓ | 2 |
| | SVS | ✓ | 1 |
| DRKey | Traditional | ✓ | 240 |
| | Coll. | • | 2640 |
| | SVS | • | Manual |
| DRKey (fixed) | Traditional | ✓ | 4 |
| | All in §3 | ✓ | 32 |
| | SVS | ✓ | 7 |

Table 4: Verification results on our further case studies.
✓ indicates that Tamarin successfully verified the property
• indicates that Tamarin found an attack.
Manual indicates that Tamarin's interactive mode was used to reconstruct the attack trace as Tamarin's built in heuristics did not terminate in a reasonable timeframe.

attack relies on a subtle signature scheme property, which previous analysis tools could not take into account. We have responsibly disclosed our attacks.

We give a brief summary of the performance of our case studies and their proposed fixes in Table 4, showing the overall tractability of our approach. Our combined approach (verification with SVS, attack finding with the equational model) demonstrates its utility here: SVS is both more efficient and finer-grained where the protocol verifies. In contrast, when there is an attack, our attack finding models are quickest.

In conjunction with a companion work, building symbolic models of non-prime order groups [36], we investigated some common cryptographic libraries' handling of Ed25519 signatures. We discovered that whilst LibSodium [57], Golang's NaCl Module [46], Project Everest's formally verified HACL [75] and Cloudflare's CIRCL [33] advertise the same API and 'drop in' compatibility, they are not consistent in their handling of Ed25519 signatures. Notably, LibSodium checks for and rejects 'low order points' which are used to construct colliding Ed25519 signatures. However, the other three libraries accept these points, allowing colliding signatures to be crafted. We reached out to the maintainers of each library and they have fixed (or agreed to fix) this issue, ensuring that protocol developers are not caught unaware.

*Full sources to all of our models are available at [49].*

# 6 Other Related Work

In the preceding, we discussed in detail the existing literature on symbolic models of digital signatures as we presented various aspects. In this section we briefly mention some alternative strands of research aiming to tackle similar problems.

**Automated Computational Verification** Computational proofs do not rely on an abstraction of signature schemes, instead reducing security of a protocol directly to (among other things) EUF-CMA. A few tools aim to construct these proofs either automatically [28] or with human assistance [16, 17]; they have the great advantage that all behaviours of the signature scheme are by definition captured, since

the reduction is directly to its security definition. This also means that proofs at this level are generally much more challenging to produce, and harder to scale to more complex protocols.

**Computational Soundness** Backes, Hofheinz and Unruh [11] consider the *computational soundness* of existing symbolic models for digital signatures. A symbolic model is computationally sound for a particular class of protocols and properties if the existence of a symbolic proof of a property implies the existence of a computational one. Most computational soundness approaches in the literature require that primitives be used in a carefully controlled fashion; for example,[11] required that signatures verify under unique public keys and for unique messages. While it seems that such requirements can be enforced by mechanisms such as appropriate tagging, real-world protocols typically do not (or cannot) meet these requirements, thereby limiting the applicability of these approaches.

Bana et al. propose an alternative model for protocol verification which they call the *Computationally Complete Symbolic Attacker*, see, e.g., [14, 15]. Their approach is the first attempt to base a symbolic model on adversary restrictions, rather than explicit capabilities. They have not yet shown their approach can be automated in practice. In their model, correctness is specified and then the adversary is permitted to act freely as long as it does not violate the axioms. Their computational soundness results apply to the proof, so that the resulting protocol is proved secure computationally.

This approach has the potential to be very powerful, and may provide an alternative to our verification approach. However, their approach is in its early stages: it works only for a bounded number of sessions and does not have tool support yet, unlike ours which is unbounded and has full tool support. Additionally, their approach is focused on proof finding, without support for establishing attacks.

# 7 Conclusions

In this work, we revisited many subtle behaviours of digital signature schemes, such as the possibility of key substitution and malleability, and showed how they fall between the cracks: their absence is not guaranteed by the classical EUF-CMA security definition for signatures, but at the same time their absence is assumed by modern automated protocol analyses. Yet the presence of such behaviours can lead, and has led, to critical attacks.

We developed a range of alternative signature models for use in modern tools. Our models capture a wide range of these behaviours and give a general theory for verification of their absence. We thereby provide the first automated procedure to show the absence or presence of attacks exploiting these subtle behaviours.

As a side effect of evaluating the effectiveness of our work, we found two new attacks on protocols, which is remarkable for multiple reasons: the WS-Security protocols served as the basis of globally used technologies and were therefore under close scrutiny, and both WS-Security and DRKey were previously proven secure.

In the wider sense, our work increases the scope of attacks considered by automated analysis tools: future protocol analysis models that include our more accurate equations will be able to find more attacks, or show the absence of more attack types.

A more long-term question is whether it is possible to "close the gap" between falsification and verification, showing that any attack found in our general theory corresponds to a real attack on the underlying signature scheme itself.

# References


[1] *ACME Draft Barnes*. 2015. URL: https://datatracker.ietf.org/doc/draft-barnes-acme/01/.

[2] *ACME Draft Barnes*. 2015. URL: https://datatracker.ietf.org/doc/draft-barnes-acme/03/.

[3] *ACME Draft Barnes*. 2015. URL: https://datatracker.ietf.org/doc/draft-barnes-acme/04/.

[4] *ACME signature misuse vulnerability in draft-barnes-acme-04*. 2015. URL: https://www.ietf.org/mail-archive/web/acme/current/msg00484.html.

[5] Jee Hea An, Yevgeniy Dodis and Tal Rabin. 'On the Security of Joint Signature and Encryption'. In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. Vol. 2332. Lecture Notes in Computer Science. Springer, 2002, pp. 83–107. ISBN: 3-540-43553-0.

[6] *Apache CXF: An Open-Source Services Framework*. 2019.

[7] *Apache CXF: WS-Security*. 2018. URL: https://cxf.apache.org/docs/ws-security.html.

[8] *Automatic Certificate Management Environment (ACME)*. 2015. URL: https://tools.ietf.org/html/draft-ietf-acme-acme-00.

[9] *Automatic Certificate Management Environment (ACME)*. 2016. URL: https://tools.ietf.org/html/draft-ietf-acme-acme-02.

[10] *Automatic Certificate Management Environment (ACME)*. 2018. URL: https://tools.ietf.org/html/draft-ietf-acme-acme-16.

[11] Michael Backes, Dennis Hofheinz and Dominique Unruh. 'CoSP: a general framework for computational soundness proofs'. In: *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*. ACM, 2009, pp. 66–78.

[12] Michael Backes et al. 'Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario'. In: *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*. 2006, pp. 428–445.

[13] Joonsang Baek and Kwangjo Kim. 'Remarks on the unknown key share attacks'. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 83.12 (2000), pp. 2766–2769.

[14] Gergei Bana and Rohit Chadha. 'Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability'. In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 69.

[15] Gergei Bana and Hubert Comon-Lundh. 'Towards Unconditional Soundness: Computationally Complete Symbolic Attacker'. In: *Principles of Security and Trust*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 189–208.

[16] Gilles Barthe et al. 'Computer-Aided Security Proofs for the Working Cryptographer'. In: *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 71–90.

[17] David A. Basin, Andreas Lochbihler and S. Reza Sefidgar. 'CryptHOL: Game-based Proofs in Higher-order Logic'. In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 753.

[18] David A. Basin et al. 'A Formal Analysis of 5G Authentication'. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 1383–1396.

[19] Daniel J. Bernstein et al. 'High-speed high-security signatures'. In: *J. Cryptographic Engineering* 2.2 (2012), pp. 77–89.

[20] Karthikeyan Bhargavan, Bruno Blanchet and Nadim Kobeissi. 'Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate'. In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 483–502.

[21] Karthikeyan Bhargavan, Antoine Delignat-Lavaud and Nadim Kobeissi. 'Formal Modeling and Verification for Domain Validation and ACME'. In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*. Vol. 10322. Lecture Notes in Computer Science. Springer, 2017, pp. 561–578.

[22] Karthikeyan Bhargavan, Cédric Fournet and Andrew D. Gordon. 'Verified Reference Implementations of WS-Security Protocols'. In: *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings*. 2006, pp. 88–106.

[23] Karthikeyan Bhargavan et al. 'Secure sessions for web services'. In: *Proceedings of the 1st ACM Workshop On Secure Web Services, SWS 2004, Fairfax, VA, USA, October 29, 2004*. 2004, pp. 56–66.

[24] Karthikeyan Bhargavan et al. 'TulaFale: A Security Tool for Web Services'. In: *Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4-7, 2003, Revised Lectures*. 2003, pp. 197–222.

[25] Karthikeyan Bhargavan et al. 'Verified interoperable implementations of security protocols'. In: *ACM Trans. Program. Lang. Syst.* 31.1 (2008), 5:1–5:61.

[26] Simon Blake-Wilson and Alfred Menezes. 'Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol'. In: *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*. Vol. 1560. Lecture Notes in Computer Science. Springer, 1999, pp. 154–170.

[27] Bruno Blanchet. 'An Efficient Cryptographic Protocol Verifier Based on Prolog Rules'. In: *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 2001, pp. 82–96.

[28] Bruno Blanchet and David Pointcheval. 'Automated Security Proofs with Sequences of Games'. In: *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*. Vol. 4117. Lecture Notes in Computer Science. Springer, 2006, pp. 537–554.

[29] Bruno Blanchet et al. 'ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial'. In: (2018). Version from 2018-05-16.

[30] Jens-Matthias Bohli, Stefan Röhrich and Rainer Steinwandt. 'Key substitution attacks revisited: Taking into account malicious signers'. In: *Int. J. Inf. Sec.* 5.1 (2006), pp. 30–36.

[31] Dan Boneh, Emily Shen and Brent Waters. 'Strongly Unforgeable Signatures Based on Computational Diffie-Hellman'. In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 229–240.

[32] Yannick Chevalier and Mounira Kourjieh. 'Key Substitution in the Symbolic Analysis of Cryptographic Protocols'. In: *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*. Vol. 4855. Lecture Notes in Computer Science. Springer, 2007, pp. 121–132.

[33] *CIRCL (Cloudflare Interoperable, Reusable Cryptographic Library)*. 2019. URL: https://github.com/cloudflare/circl (visited on 02/02/2019).

[34] Cas Cremers and Martin Dehnel-Wild. 'Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion'. In: *Network and Distributed Systems Symposium (NDSS 2019)*. (To appear). 2019.

[35] Cas Cremers, Martin Dehnel-Wild and Kevin Milner. 'Secure Authentication in the Grid: A Formal Analysis of DNP3: SAv5'. In: *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*. Vol. 10492. Lecture Notes in Computer Science. Springer, 2017, pp. 389–407.

[36] Cas Cremers and Dennis Jackson. 'Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman'. In: *IEEE CSF* 19 (2019).

[37] Cas Cremers et al. 'A Comprehensive Symbolic Analysis of TLS 1.3'. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1773–1788.

[38] Cas Cremers et al. 'Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication'. In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 470–485.

[39] Christian Decker and Roger Wattenhofer. 'Bitcoin Transaction Malleability and MtGox'. In: *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*. Vol. 8713. Lecture Notes in Computer Science. Springer, 2014, pp. 313–326.

[40] Antoine Delignat-Lavaud et al. 'Implementing and Proving the TLS 1.3 Record Layer'. In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 463–482.

[41] Whitfield Diffie, Paul C. van Oorschot and Michael J. Wiener. 'Authentication and Authenticated Key Exchanges'. In: *Des. Codes Cryptography* 2.2 (1992), pp. 107–125.

[42] Shaddin F Doghmi, Joshua D Guttman and F Javier Thayer. 'Searching for shapes in cryptographic protocols'. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2007, pp. 523–537.

[43] Santiago Escobar, Catherine Meadows and José Meseguer. 'Maude-NPA: Cryptographic protocol analysis modulo equational properties'. In: *Foundations of Security Analysis and Design V*. Springer, 2009, pp. 1–50.

[44] *Getting Started with gSOAP*. 2019. URL: https://www.genivia.com/dev.html.

[45] Shafi Goldwasser, Silvio Micali and Ronald L. Rivest. 'A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks'. In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308.

[46] *go/x/crypto/nacl*. 2019.

[47] Felix Günther and Bertram Poettering. 'Linkable Message Tagging: Solving the Key Distribution Problem of Signature Schemes'. In: *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*. Vol. 9144. Lecture Notes in Computer Science. Springer, 2015, pp. 195–212.

[48] ISO Central Secretary. *Information technology security techniques - key management - Part 3: Mechanisms using asymmetric techniques*. en. Standard ISO/IEC 11770-3. Version Final Text for Publication. Geneva, CH: International Organization for Standardization, 1999.

[49] Dennis Jackson et al. *Supplementary Materials and Models*. 2019. URL: https://people.cispa.io/cas.cremers/downloads/archives/Tamarin_better_signatures.zip.

[50] Tibor Jager, Saqib A. Kakvi and Alexander May. 'On the Security of the PKCS#1 v1.5 Signature Scheme'. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 1195–1208.

[51] jedisct1. *It's possible to forge messages that cryptosignopen verifies if the public key is zero*. Issue 112 jedisct1/libsodium. 2016. URL: https://github.com/jedisct1/libsodium/issues/112 (visited on 11/2018).

[52] Simon Josefsson and Ilari Liusvaara. 'Edwards-Curve Digital Signature Algorithm (EdDSA)'. In: *RFC* 8032 (2017), pp. 1–60.

[53] B. Kaliski, J. Jonsson and A. Rusch. *PKCS# 1: RSA Cryptography Specifications Version 2.2, Section 9.2, Note 2*. 2016.

[54] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 9781466570269.

[55] Tiffany Hyun-Jin Kim et al. 'Lightweight source authentication and path validation'. In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. ACM. 2014, pp. 271–282.

[56] Nadim Kobeissi, Karthikeyan Bhargavan and Bruno Blanchet. 'Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach'. In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 435–450.

[57] *LibSodium: a modern, easy-to-use cryptographic library*. 2019. URL: https://download.libsodium.org/doc/ (visited on 02/02/2019).

[58] Christina Lindenberg, Kai Wirt and Johannes A. Buchmann. 'Formal Proof for the Correctness of RSA-PSS'. In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 11.

[59] Alfred Menezes and Nigel P. Smart. 'Security of Signature Schemes in a Multi-User Setting'. In: *Des. Codes Cryptography* 33.3 (2004), pp. 261–274.

[60] *Message Security in WCF*. 2019.

[61] *OASIS Web Services Security (WSS) TC*. 2006.

[62] *Oracle Fusion Middleware: WS-Policy Reference*. 2014. URL: https://docs.oracle.com/cd/E55956_01/doc.11123/user_guide/content/ws_policies.html.

[63] Adrian Perrig et al. *SCION: A Secure Internet Architecture*. Springer International Publishing AG, 2017. ISBN: 978-3-319-67079-9.

[64] Thomas Pornin and Julien P. Stern. 'Digital Signatures Do Not Guarantee Exclusive Ownership'. In: *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*. Vol. 3531. Lecture Notes in Computer Science. 2005, pp. 138–150.

[65] Benedikt Schmidt et al. '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*. IEEE Computer Society, 2012, pp. 78–94.

[66] *Securing your Web services with Spring-WS*. 2018. URL: https://docs.spring.io/spring-ws/docs/3.0.4.RELEASE/reference/#security.

[67] Jacques Stern et al. 'Flaws in Applying Proof Methodologies to Signature Schemes'. In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 93–110.

[68] Tamarin Team. *Tamarin-Prover Manual – Security Protocol Analysis in the Symbolic Model*. Tech. rep. Version of 2018-11-27. https://tamarin-prover.github.io/manual/index.html: tamarin-prover.github.io, 2016.

[69] Serge Vaudenay. 'The Security of DSA and ECDSA'. In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. 2003, pp. 309–323.

[70] *WCF: Message Security with Mutual Certificates*. 2017. URL: https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/message-security-with-mutual-certificates.

[71] *WebSphere Application Server Liberty base*. 2019.

[72] *WS-Attacks*. 2019.

[73] *(WSS1.1) Mutual Authentication with X.509 Certificates, Sign, Encrypt*. 2019.

[74] Fuyuan Zhang et al. 'Mechanized network origin and path authenticity proofs'. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 346–357.

[75] Jean-Karim Zinzindohoué et al. 'HACL*: A verified modern cryptographic library'. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1789–1806.

# Acronyms

**ACME** Automatic Certificate Management Environment
**CA** Certificate Authority
**CEO** Conservative Exclusive Ownership
**DEO** Destructive Exclusive Ownership
**DH** Diffie-Hellman
**DSKS** Duplicate Signature Key Selection
**EUF-CMA** Existential Unforgeability under an Adaptive Chosen Message Attack
**IETF** Internet Engineering Task Force
**KCI** Key Compromise Impersonation

**KDF** Key Derivation Function
**LE** Let's Encrypt
**PFS** Perfect Forward Secrecy
**SEF-CMA** Strong Existential Unforgeability under an Adaptive Chosen Message Attack
**STS** Station-to-Station
**UEO** Universal Exclusive Ownership
**UKS** Unknown Key Share