Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses

Vincent Cheval, *Inria Paris;* Cas Cremers and Alexander Dax, *CISPA Helmholtz Center for Information Security;* Lucca Hirschi, *Inria & LORIA;* Charlie Jacomme, *Inria Paris;* Steve Kremer, *Université de Lorraine, LORIA, Inria Nancy Grand-Est*

https://www.usenix.org/conference/usenixsecurity23/presentation/cheval

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the
32nd USENIX Security Symposium
is sponsored by USENIX.

Hash Gone Bad:
Automated discovery of protocol attacks that exploit hash function weaknesses

Vincent Cheval[¶] *Cas Cremers*[‡] *Alexander Dax*[‡]
Lucca Hirschi[†] *Charlie Jacomme*[¶] *Steve Kremer*[*]

[¶] *Inria Paris, France*
[‡] *CISPA Helmholtz Center for Information Security, Germany*
[†] *Inria & LORIA, France*
[*] *Université de Lorraine, LORIA, Inria Nancy Grand-Est, France*

Abstract

Most cryptographic protocols use cryptographic hash functions as a building block. The security analyses of these protocols typically assume that the hash functions are perfect (such as in the random oracle model). However, in practice, most widely deployed hash functions are far from perfect – and as a result, the analysis may miss attacks that exploit the gap between the model and the actual hash function used.

We develop the first methodology to systematically discover attacks on security protocols that exploit weaknesses in widely deployed hash functions. We achieve this by revisiting the gap between theoretical properties of hash functions and the weaknesses of real-world hash functions, from which we develop a lattice of threat models. For all of these threat models, we develop fine-grained symbolic models.

Our methodology's fine-grained models cannot be directly encoded in existing state-of-the-art analysis tools by just using their equational reasoning. We therefore develop extensions for the two leading tools, TAMARIN and PROVERIF. In extensive case studies using our methodology, the extended tools rediscover all attacks that were previously reported for these protocols and discover several new variants.

1 Introduction

Cryptographic hash functions are a fundamental and highly efficient building block in nearly all cryptographic protocols. They are traditionally required to meet several security properties, such as collision resistance and first and second preimage resistance. Ideally, they are "perfect" and do not suffer from phenomena like length-extension attacks. Modern hash functions like SHA3 (Keccak) are believed to satisfy all these properties and behave like a perfect hash function.

In many modern protocol security analyses, both in the computational and symbolic setting, hash functions are assumed to be "perfect" in the following sense: the modeled hash function meets all desired cryptographic properties and every input/output combination is completely independent of all others. Such a hash function corresponds to the "Random Oracle Model (ROM)" often used in cryptographic proofs.

In practice though, real hash functions are unfortunately far from perfect. In Table 1 we show some widely deployed hash functions and their currently known imperfections. Several of these hash functions, such as SHA1, are considered to be weak or broken with respect to collision or preimage resistance; and nearly all widely deployed hash functions allow so-called "length extension attacks", which can enable someone to compute $hash(x\|y)$ even if they do not know x – which is not possible with a perfect hash function.

There are several reasons for the gap between reality and the perfect hash function. First, the security of hash functions is often based on a heuristic argument, since we cannot reduce them to a known hard problem, and history has shown that many hash functions that initially seemed secure turned out to be broken some years later [2, 52]. Second, it was long believed that potential hash weaknesses would not weaken protocols that use (second) preimage resistant [9, 48] hash functions. Third, even if a hash function satisfies all standard requirements for cryptographic hash functions (resistance to collisions and preimages), it may still not be perfect. For example, many popular hash function designs follow the Merkle-Damgård (MD) construction, which in its default setup, allows for length extension attacks. We thus have to face the reality: protocols use hash function that are already imperfect, and history has shown that over time, hash functions that appear secure now will become easier to attack in the future.

This raises the natural question: how can we check if a protocol using a hash function with a particular weakness meets its security guarantees? History has shown such attacks are rare but can be very subtle, *e.g.,* [9, 48, 49], and thus difficult to detect manually. From a cryptographic perspective, the answer would be: provide a computational proof of the security of the entire protocol, and if this proof relies on assumptions not met by the hash function, this may indicate an attack. However, for most protocols, this task ranges from daunting to infeasible; and most existing protocol proofs simply assume that the hash function is perfect, by using the

Hash function	Year	Examples of currently deployed applications	Collision resistance	(2nd) Preimage resistance	No Length-extension
MD4	1990	NTLM key derivation for Microsoft Windows	✗ 2^{1}	⊗* 2^{95}	✗
MD5	1992	File checksums (md5sum)	✗ 2^{18}	⊗* 2^{123}	✗
SHA1	1995	Europay Mastercard Visa (EMV), File checksums, Telegram	✗ 2^{61}	✓ 2^{160}	✗
RIPEMD-160	1996	Bitcoin	⊗** 2^{80}	✓ 2^{160}	✗
SHA2-256	2001	Bitcoin, TLS, SSL, SSH, S/MIME, IPSec, DNSSEC, Linux/Unix password hashing, Telegram	✓ 2^{128}	✓ 2^{256}	✗
SHA2-512	2001	TLS, SSL, SSH, S/MIME, IPSec, DNSSEC, Linux/Unix password hashing	✓ 2^{256}	✓ 2^{512}	✗
SHA3-256	2012	Ethereum	✓ 2^{128}	✓ 2^{256}	✓

✓= currently still secure ⊗=weak, but no full attack yet ✗= known attack

* = Theoretical attacks on (second) preimage resistance were found [54] [44], but they are still not feasible.

** = The small bit size allows to find collisions in practice, but doing so is not necessarily feasible.

Table 1: Examples of widely used hash functions that are currently deployed in security protocols and do not offer perfect (random-oracle like) guarantees. The numbers indicate the complexity of the currently best known attack on the property [32,44,45,53,54]. For the hash functions currently deemed secure the best known attacks would be a brute-force approach; *e.g.,* the complexity to break collision resistance on SHA2-256 is 2^{128}. Crucially, this situation is not constant, but expected to get worse: history suggests that the numbers for the best attacks are likely to decrease over time for all hashes, see *e.g.,* [2,48].

ROM.

In contrast, automated protocol analysis tools have shown to be effective for analyzing real-world protocols [5, 6, 8, 17, 19, 28]. However, they model hash functions as being perfect (traditionally as an operator in a free term algebra). Thus, like computational proofs that use the ROM, such analyses miss any attacks that exploit the use of a non-perfect hash function.

In this work, we revisit cryptographic hash function definitions, common weaknesses, and the potential attacker capabilities that arise from them. Based on this, we develop a methodology to systematically discover attacks on protocols that exploit their use of “less-than-perfect” hash functions, and show how this can be implemented in the two leading protocol-analysis tools. To realize this, we both exploit advanced features of these tools (such as equational theories, event-based modeling, and restrictions) but we in fact also extend them (partial support for associative operators, recursive computation functions). Our methodology can be used in the design phase to avoid the use of hash functions that are too weak, or to find and fix problems in deployed protocols.

Contributions.

1. We develop the first systematic, automated methodology to find protocol attacks that exploit weaknesses of real-world hash functions. At a technical level, we achieve this by symbolically modeling cryptographic weaknesses (*i.e.,* the lack of desirable cryptographic properties) as well as real-world attack classes that are not captured by classical security definitions for cryptographic hash functions.
2. We automate our methodology in the two leading automated protocol analysis tools, TAMARIN and PROVERIF. To achieve this, we (a) propose dedicated modeling techniques, and (b) extend both tools with new required features that are of independent interest beyond this work.
3. We apply our methodology to over 20 protocols, automatically rediscovering all previously reported attacks on those protocols that exploit weak hash functions, as well as finding several new variants.

Source code and reproducibility. We provide the sources of the modified PROVERIF and TAMARIN tools as well as our all case-studies at [15]. This includes a docker image with an environment that directly allows the reproduction of the case-studies, and notably all the tables of the detailed case studies available in the long version [16], by following the documentation in the READMEs. Installation instructions from *dockerhub* for the docker image are included in [15].

Outline. We first provide some background about hash functions, their security properties and how weaknesses of deployed hash functions may break protocols in practice (Section 2). Then we present a novel hierarchy of threat models related to hash functions, detailing adversarial capabilities that we are going to use for protocol analysis (Section 3) and show how this analysis can be automated in (improved versions of) TAMARIN and PROVERIF (Section 4). Finally, we demonstrate the effectiveness and methodology of our approach on a number of case studies discovering both known and novel vulnerabilities (Section 5). We provide additional related work in Section 6 and conclude in Section 7.

2 Background

Many of the problems around hash functions arise from the gap between the properties described in the theory and the property that real-world hash functions satisfy; but in this particular case, there is already sufficient tension in how hash functions are handled in the theory of protocol proofs. We first describe the state of the theory, before returning to the situation in practice. Afterwards we give background on the symbolic model that we will be using in the next sections.

2.1 Hash functions in theory

The main three desirable properties that a cryptographic hash function should satisfy are well-established: first and second preimage resistance, and collision resistance [38], stating that there is no better algorithm than brute force to

- **(Preimage resistance)** find x given h, such that $h = \mathsf{H}(x)$;
- **(Second preimage resistance)** find y given x, such that $\mathsf{H}(y) = \mathsf{H}(x)$;
- **(Collision resistance)** find x, y such that $\mathsf{H}(x) = \mathsf{H}(y)$

Collision-resistance implies second preimage resistance, as finding a second preimage effectively results in a collision.

An undesirable property is so-called **length-extension**: Many deployed hash functions are based on the MD construction: $\mathsf{H}(m\|m') = f(\mathsf{H}(m), m')$ where f is the underlying compression function and $\|$ expresses concatenation of blocks. The origin of this design choice can be traced back to an implicit design goal of many hash functions: it should be possible to compute a hash incrementally, *i.e.,* to compute $H(m\|m')$ without having to store both m and m' in memory, for example by computing a compact intermediate product based on m to later compute the full result once m' is available. By default, MD constructions satisfy the *length-extension* property: Given $H(m)$ and m', one can compute $H(m\|m')$As we will see below, this property can be problematic in certain protocol contexts because it enables so-called *length-extension attacks*. In theory, this possibility has been known in the academic literature at least as early as 1992 [51].

However, when developing proofs of security protocols that use hash functions, it turns out using the three resistance properties as assumptions is very complex and error-prone. Because of this, the **Random Oracle Model (ROM)** was introduced in 1993 by Bellare and Rogaway [7] as a proof methodology to simplify proofs of protocols that use hash functions. We will go into more detail in Section 2.2.1, but intuitively speaking, the ROM models perfect hash functions: as functions whose outputs are chosen uniformly at random, independent of the input. We will give a definition of (a symbolic version of the) ROM in Section 3.

The ROM satisfies first and second preimage resistance and collision resistance, and does *not* have the length-extension property. The ROM thus has the most desirable cryptographic hash properties, effectively over-approximating the security of real-world hash functions. Therefore, proving that a protocol is secure using the ROM does not guarantee that it is secure when instantiated with a real hash function. However, the vast majority of protocol proofs use the ROM due to its simplicity.

2.2 Hash functions in practice

In Table 1 we review a selection of widely used hash functions, the complexity of the best known collision and preimage attacks against them, and whether length-extension is possible. The conclusion is clear: high-profile cryptographic protocols still use hash functions that suffer from weaknesses that contradict the usual idealized security assumption (ROM) and even (second) preimage and collision resistance. We also see that hash functions get weaker over time as the attacks get more and more efficient (see also the survey [48] and [2]). Moreover, it is likely that hash functions that are deemed secure today will be weakened in the future.

The length extension property of many hash functions can theoretically be used to break a protocol's security, because it enables the following behaviors: (i) collisions can be extended since $\mathsf{H}(x) = \mathsf{H}(y)$ implies $\mathsf{H}(x\|s) = \mathsf{H}(y\|s)$ for any s, and (ii) the adversary can extend the payload under known hash outputs: given $\mathsf{H}(x)$, it can compute $\mathsf{H}(x\|s)$ for any known s. As an example of the latter, if the prefix x contains a shared secret, and the protocol relies on this to authenticate hash values, then the adversary can "forge" hashes by extending any hash values it observes. An early example of a widely deployed protocol that was vulnerable due to such an attack was Flickr in 2009 [21]; we will revisit this attack in Section 5.3.2.

Despite this example, it was thought for a long time that cryptographic protocols are likely to remain secure even though they rely on weakened hash functions as long as the hash functions are (second) preimage resistant [9, 48]. For instance, even if the adversary can compute *some* c, c' such that $\mathsf{H}(c) = \mathsf{H}(c')$, which breaks collision-resistance, it seems unlikely that it can impact honest agents in a protocol who will compute hashes for inputs that are unrelated to c, c'. Unfortunately, this is a false sense of security: it has been shown that cryptographic protocols can be entirely broken when using hash functions that merely suffer from some restricted classes of collisions [48, 49]; see an example in Section 2.2.1.

Unfortunately, it is difficult and error-prone to manually assess if a cryptographic protocol can be broken if its hash function is vulnerable to some restricted class of collisions.

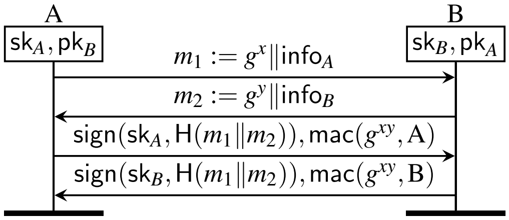
A
$\mathsf{sk}_A, \mathsf{pk}_B$
B
$\mathsf{sk}_B, \mathsf{pk}_A$
$m_1 := g^x\|\mathsf{info}_A$
$m_2 := g^y\|\mathsf{info}_B$
$\mathsf{sign}(\mathsf{sk}_A, \mathsf{H}(m_1\|m_2)), \mathsf{mac}(g^{xy}, \mathrm{A})$
$\mathsf{sign}(\mathsf{sk}_B, \mathsf{H}(m_1\|m_2)), \mathsf{mac}(g^{xy}, \mathrm{B})$

Figure 1: The Sigma' protocol [9]

2.2.1 Example: Hash Transcript Collisions

Using *hash transcript collisions*, we exemplify how certain collisions can be weaponized against protocols. They have been shown to affect various authentication protocols such as TLS 1.2, SSH, or IKEv2 [9]. As a running example, we use a variant of the sign-and-mac protocol [29]: the Sigma' authentication protocol introduced in [9] and depicted in Figure 1. It is essentially a signed Diffie-Hellman (DH) protocol with MAC-based key confirmation where additional information info_A and info_B (*e.g.,* for later negotiation) is appended to the exchanged DH shares. $\mathsf{info}_A, \mathsf{info}_B$ are length-varying and therefore prefixed with their lengths. Sigma' aims at guaranteeing *matching conversations*: after a successful execution both parties share the same view of the transcript, even in the presence of an active attacker. As noted in [9], the parties do not directly agree on the transcript but rather on the hash of the transcript. If the hash function were perfect, this would not matter – but it makes a difference for real-world hash functions.

Machine-in-the-Middle (MIM) scenario. First, one should note that the protocol is not immediately broken, even if the hash function H is not preimage resistant. Assume a MIM attacker: the attacker can replace messages m_1 and m_2 by messages $m_1' := g^{x'} \| \mathsf{info}_A'$ and $m_2' := g^{y'} \| \mathsf{info}_B'$ of its choice. This results in the message $m_3 = \mathsf{sign}(\mathsf{sk}_A, \mathsf{H}(m_1 \| m_2')), \mathsf{mac}(g^{xy'}, \mathrm{A})$. If the attacker does not know sk_A, it cannot modify m_3. Hence, B will check whether $\mathsf{H}(m_1 \| m_2') = \mathsf{H}(m_1' \| m_2)$. Clearly, (second) preimage attacks do not directly allow such a MIM attack to succeed, as the input for the target hash output $\mathsf{H}(m_1 \| m_2')$ must be of the form $m_1' \| m_2$, where m_2 is fixed and not adversary-chosen. Similarly, the mere existence of collisions, say $c \neq c'$ such that $\mathsf{H}(c) = \mathsf{H}(c')$, cannot be used to break this protocol.

Hash transcript collisions attacks. Chosen-Prefix Collision (CPC) [48, 49] are among the least costly collisions to compute and yet can be weaponized against protocols. Given two prefixes, p_1 and p_2, a CPC attack computes two suffixes $s_1 \neq s_2$ such that $\mathsf{H}(p_1 \| s_1) = \mathsf{H}(p_2 \| s_2)$. When additionally $p_1 = p_2$, such a collision is called Identical-Prefix Collision (IPC) and is even less costly to compute.

As we shall see, Sigma' is entirely broken as soon as *(i)* the used hash function suffers from CPC attacks, *(ii)* obeys the length-extension property, and *(iii)* the length of m_2 is predictable. Indeed, given m_1 sent by A, the adversary can choose arbitrary x', y' and compute a CPC for prefixes $m_1 \| g^{y'}$ and $g^{x'}$, resulting in suffixes $\mathsf{infoPartial}_B'$ and info_A' such that:

$$\mathsf{H}(m_1 \| g^{y'} \| \mathsf{infoPartial}_B') = \mathsf{H}(g^{x'} \| \mathsf{info}_A'). \qquad (\dagger)$$

Moreover, the claimed length field of $\mathsf{infoPartial}_B'$ can be chosen to be $|\mathsf{infoPartial}_B'| + |m_2|$ The MIM adversary then uses $m_1' := g^{x'} \| \mathsf{info}_A'$ and $m_2' = g^{y'} \| \mathsf{info}_B'$ where $\mathsf{info}_B' = \mathsf{infoPartial}_B' \| m_2$. By the length-extension property of H, we obtain by appending m_2 to the above collision ($\dagger$):

$$\mathsf{H}(m_1 \| m_2') = \mathsf{h}(m_1 \| g^{y'} \| \mathsf{info}_B') = \mathsf{H}(m_1' \| m_2).$$

Therefore, the MIM adversary successfully impersonated A and B and hijacked the session key, *i.e.,* $g^{xy'}$ with A and $g^{x'y}$ with B. To give a sense of the attack cost, finding such a collision costs about 2^{39} for MD5 and $2^{63.4}$ for SHA1 [32, 48]. As we shall see in Section 5, other kinds of CPC but no IPC affect Sigma', findings we formally establish with our automated formal analysis framework (Section 4).

2.3 Symbolic Model: Term Algebra

In the next section we will present our formal analysis framework, which is based on the so-called *symbolic model*, going back to the seminal work of Dolev and Yao [20]. In this model, messages are described by terms from a term algebra. For example, $\mathsf{senc}(m,k)$ represents the message m encrypted using the key k. The algebraic properties of cryptographic functions are specified by equations over terms. For example, $\mathsf{sdec}(\mathsf{senc}(m,k),k) = m$ specifies the expected semantics for symmetric encryption: decryption using the encryption key yields the plaintext. As is common in the symbolic model, cryptographic messages only satisfy those properties explicitly specified algebraically. This yields the now standard *black-box cryptography assumption*: one cannot exploit potential weaknesses in cryptographic primitives beyond those explicitly specified. Still, a wide range of attacks, including *logical attacks* and attacks based on an explicit algebraic model, are covered.

Formally, we assume a set of *operators* with their arities as signature Σ and a countably infinite set of variables $\mathcal{V}$. Operators model computations over messages such as symmetric encryption. We similarly treat *atoms* (usually called *names* in symbolic models), *i.e.,* atomic data such as nonces or keys, with a countable set $\mathcal{A}$. The set of *terms* given by the closure of using operators from the signature Σ containing variables in $\mathcal{V}$ and atoms in $\mathcal{A}$ is denoted $T := T_\Sigma(\mathcal{V}, \mathcal{A})$. A *message* is a term without variable. A *substitution* σ is a function from variables to messages. We homomorphically lift substitutions to terms.

Algebraic properties over operators, such as decrypting a ciphertext with the right key yields the plaintext, are expressed through an equational theory. Given a signature Σ, an *equation* is an unordered pair of terms s and t, written $s = t$, for $s, t \in T_\Sigma(\mathcal{V})$. To a set of equations E, we associate an *equational theory* that is the smallest congruence relation over terms $=_E$ that contains E and is closed under substitution of terms for variables and atoms. Two messages s and t are equal modulo E if and only if $s =_E t$.

Example 1. *For a basic model of digital signatures, let* Σ *contain the operators* $\mathsf{sign}(\cdot,\cdot)$, $\mathsf{checksign}(\cdot,\cdot)$, $\mathsf{pk}(\cdot)$ *and a constant (i.e., operator of arity 0)* $\mathsf{true} \in \Sigma$ *together with the equation* $\mathsf{checksign}(\mathsf{sign}(x,y), \mathsf{pk}(y)) = \mathsf{true}$. *This does*

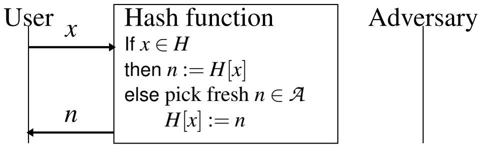

Figure 2: Abstract hash function in the ROM. Initially, H is the empty mapping: $H := \emptyset$. Note that the adversary can also act as a user of the hash function, but it cannot *influence* the oracle, unlike in some threat models we will define later.

indeed model signature verification as the public key used for verification $\mathsf{pk}(y)$ *must match the signature key y, which can be instantiated with any message.*

Example 2. *We assume a concatenation operator* $\cdot\|\cdot \in \Sigma$ *equipped with an equation* $(x\|y)\|z = x\|(y\|z)$*, that is the concatenation is associative. We shall use* $\|$ *to concatenate (suffix as second argument) different messages prior to hashing.*

To simplify presentation, we tacitly identify a message t with its equivalence class modulo $=_E$, *i.e.,* the set $\{t' \mid t' =_E t\}$. In particular, we interpret all relations to be closed under the equational theory. For example, we simply write $t \in S$ for $\exists t' :\ t =_E t' \wedge t' \in S'$ and assume that for any binary relation $\sim$, $t_1 \sim t_2$ implies $t'_1 \sim t'_2$ whenever $t'_1 =_E t_1$ and $t'_2 =_E t_2$.

3 Threat Models for Hash Functions

We now develop our hierarchy of hash function models. We start from the ROM model, which represents an ideal hash function, *i.e.,* in which the adversary has the least possible capabilities to manipulate or learn information from it. We then strengthen the adversary's capabilities in various dimensions, corresponding to possible weaknesses of hash functions.

A core observation is that from the different types of possible weaknesses – here framed as adversarial capabilities – some are independent of others, and some are related. For example, Length Extension attacks and CPC are independent: there exist hash functions that have one of these two weaknesses, but not the other. In contrast, CPC and IPC are related: if a hash function is vulnerable to CPC attacks, then the attacker can also choose two identical prefixes: thus, any hash function that is vulnerable to CPC is also vulnerable to IPC.

We identify four main independent dimensions of hash function weaknesses, and thus corresponding adversary capabilities: collision-related weaknesses, length-extension style weaknesses, output-control weaknesses that can model, *e.g.,* backdoored hash functions, and weaknesses that leak information about the inputs from the output.

Random Oracle Model (ROM). Virtually all prior symbolic analyses model hash functions in the ROM, which corresponds to the weakest possible adversarial capabilities. At the technical level, this means the hash function is symbolically modeled as a free operator, *i.e.,* an operator $\mathsf{H}(\cdot) \in \Sigma$ that does not occur in any equation (in E). Since H does not occur in E, $\mathsf{H} : T \to T$ has the same algebraic property as a random oracle: it associates to an input t, an output value $\mathsf{H}(t)$ that has no other algebraic property than being the hash output of t, modeled as a fresh atom n_t. We informally describe this modeling choice using an abstract hash functionality that is interfaced with a user and an adversary. Here, the adversary has no control over the hash function, as shown in Figure 2.

Modeling dimensions of Hash Weaknesses. We identified four main *dimensions* of adversarial capabilities that together can form various *threat models*, *i.e.,* any two capabilities from different dimensions can always be combined.

The overall structured lattice is depicted with its dimensions in Figure 3. Capabilities higher up represent stronger capabilities; the capability at the bottom in each dimension is the weakest one and implies that the attacker does not have a meaningful capability in this dimension. For example, the combination of the capabilities on the bottom row effectively corresponds to the ROM model of an ideal hash function. We use a list notation to represent a specific threat model, by listing the adversarial capabilities in each dimension. For example, we denote the weakest threat model across the bottom row by $\emptyset$. Conversely, {allCol,allExt,anyTarget,inLeak} is the strongest threat model in which the adversary has all capabilities (and corresponds to modeling the weakest hash function). We now introduce the details of each dimension in turn, including various types of collisions, how hash outputs relate to other messages, and modeling hashes that leak their inputs.

Allowed Collisions (COL). Without ROM-like constraints, all kinds of worst-case collisions might be considered, provided that the resulting hash function is indeed a function. For instance, this includes the constant function that maps all inputs to a single value. Such a strong adversarial capability corresponds to an extremely weak hash function requirement, and can be of interest to establish strong security guarantees when possible. In fact, such a constant function is not even the worst-case scenario: if the protocol has authentication properties or inequality checks, modeling a hash function as a constant function might miss attacks. On the other hand, a protocol can be deemed insecure with this strong adversarial capability due to unrealistic attacks. To refine this, we restrict the allowed choices of hash outputs to the relevant classes of collisions one may want to consider.

This is done using a **collision-relation** $\sim_c$: it captures that for all x and y such that $x \sim_c y$, the hash function H allows collisions $H[x] =_E H[y]$. We cover a large spectrum of types of collisions that can be combined as defined in Table 2 (the last two rows are explained next). To define those relations, we introduce a number of abstract operators $\mathsf{cp}_1(\cdot,\cdot)$, $\mathsf{cp}_2(\cdot,\cdot)$, $\mathsf{sp}_1(\cdot)$, $\mathsf{sp}_2(\cdot)$, $\mathsf{pi}^1(\cdot)$, $\mathsf{pi}^2(\cdot)$, $\mathsf{c}()$, $\mathsf{c}'() \in \Sigma$ that do not occur

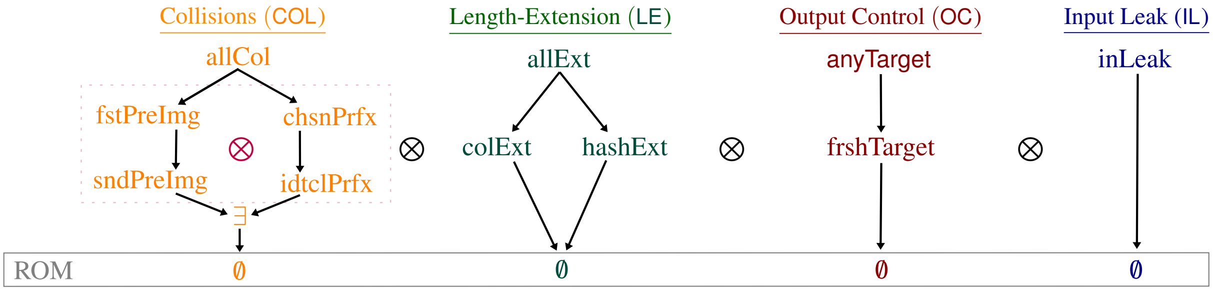

Figure 3: Lattice of adversarial capabilities. An edge $x \to y$ expresses that x is a stronger capability than y. Each column is a dimension described at the top. Capabilities from different dimensions can be combined into threat models, *e.g.,* {idtclPrfx,allExt}. The minimal threat model is the empty one, $\emptyset$, which corresponds to the ROM.

in protocols. Those operators correspond to computations performed by the adversary to find a certain collision or preimage, and are motivated by real-world attack strategies. For instance, given two messages p_1, p_2, the messages $\mathsf{cp}_1(p_1, p_2)$ and $\mathsf{cp}_2(p_1, p_2)$ correspond to the message the adversary obtains when computing a CPC for the prefixes p_1 and p_2: the hash of $p_1 \| \mathsf{cp}_1(p_1, p_2)$ equals the hash of $p_2 \| \mathsf{cp}_2(p_1, p_2)$, as expressed by $\sim_{\mathsf{CP}}$. (See an example of CPC in Section 2.2.1.)

Given a relation $\sim_c$ corresponding to the chosen kinds of collisions, the allowed hash outputs determined by the output control dimension OC (explained below) are filtered out. The resulting hash function is depicted in Figure 4.

Length-Extension (LE). The two last types of collisions defined in Table 2 are specific to the length-extension property and weaknesses of hash functions built with MD.

The first, hashExt, captures the adversarial capability to extend the payload that is under a hash with some adversary-chosen suffix, and is captured by the collision-relation $\sim_{\mathsf{LEa}}$. Namely, given a hash output $H[x]$, for any suffix s of its choice the adversary can compute $H[x\|s]$ without knowing x. Indeed, the hash output of $x\|s$, which the attacker cannot compute, is allowed to collide with the one of $H[x]\|s$, which the attacker can compute. This is the most classical weakness of length-extension. The second, colExt, corresponds to the fact that collisions may be extended, *i.e.,* as soon as $H[x] = H[y]$, we will also have that $H[x\|s] = H[y\|s]$ for any s. Interestingly, HMAC-SHA2 and HMAC-MD5 constructions have colExt but not hashExt for a given key.

Output Control (OC). In a worst-case scenario, we consider a hash function where the attacker may control the output of the hash function to some extent, provided that the resulting hash is indeed a function. Such scenarios could occur if, *e.g.,* the hash function was badly designed or has a backdoor. It also mirrors attacks similar to the nostradamus attack over MD5 [26], for which the attacker can inject some bytes inside the input to make the hash output go to a previously chosen target. The hash outputs can be taken from (i) a set of atoms $\mathcal{A}_H$, that model fresh values unknown to the attacker (unless revealed), as in the ROM, (ii) fresh atoms chosen by the attacker (frshTarget), or (iii) arbitrary messages provided that the adversary knows them (anyTarget). The default capability models one of the behaviours of the ROM, where each hash output is taken from a set of fresh atoms (but still allowing for collisions based on COL). The second capability frshTarget models the case where the attacker can partially control the outputs of the hash function that still need to be taken from a set of atoms. This captures some type-flaw attacks but the attacker cannot control the actual shape of the hash values which will appear as junk bytes. The third one anyTarget additionally captures arbitrary type-flaws attacks, where the attacker can fully control the hash output to an arbitrary value.

Input Leak (IL). Finally, hash functions are sometimes implicitly assumed to guarantee confidentiality of their inputs, and sometimes ill-used for this purpose in practice. In reality, some badly designed hash functions might leak information about their inputs. Previous symbolic analyses did not capture this capability, which can yield practical attacks. We model the worst case scenario in which the adversary can obtain the complete input. We therefore introduce an adversarial capability inLeak that allows the adversary to learn the hash input of a given hash output, which is directly obtained by forwarding every hash inputs x in Figure 4 to the attacker.

3.1 Lattice of threat models

As explained previously, we use the notation $\{\cdot\}$ to describe a specific threat model made of the explicit hash weakness in each dimension, and omit it when there is none for this dimension. The weakest capabilities occur as $\emptyset$ corresponding to ROM and Figure 2. Conversely, in the strongest threat model, {allCol,allExt,anyTarget,inLeak} all collisions are possible (and adversary-chosen), all types of type-flaw attacks are considered, and hash outputs leak their inputs. The purpose of the lattice structure is to structure this spectrum of threat models spanning those two extremes. In the next section we show how we effectively explore the lattice.

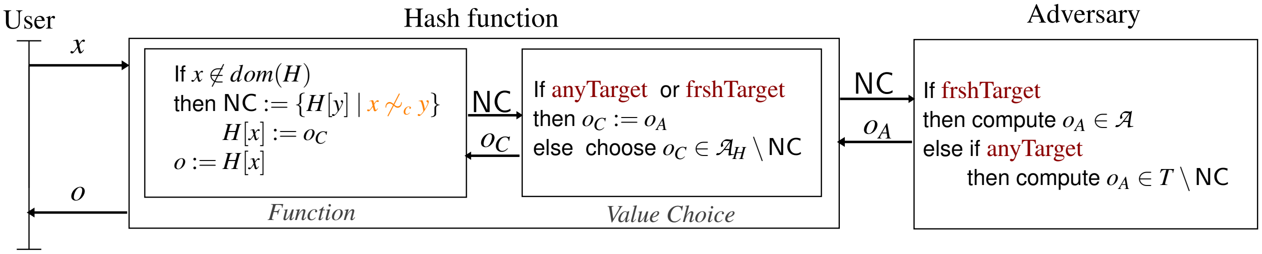
User
x
o
Hash function
If $x \notin dom(H)$
then NC $:= \{H[y] \mid x \not\sim_c y\}$
$H[x] := o_C$
$o := H[x]$
Function
NC
o_C
If anyTarget or frshTarget
then $o_C := o_A$
else choose $o_C \in \mathcal{A}_H \setminus$ NC
Value Choice
NC
o_A
Adversary
If frshTarget
then compute $o_A \in \mathcal{A}$
else if anyTarget
then compute $o_A \in T \setminus$ NC

Figure 4: Generic hash function model that generalizes the model from Figure 2, and can be instantiated with different adversarial capabilities. Initially, $H := \emptyset$. Collisions and Length-Extensions do not appear explicitly since they are captured in $\sim_c$ (see Table 2). If the input leak capability is present, the model additionally includes that every hash input is forwarded to the attacker.

Capability	$\sim_c$	Intuitions behind the types of allowed collisions	
$\emptyset$	$\sim_\bot$	Ideal model in which hash outputs never collide;	$\forall t \neq t' : t \not\sim_\bot t'$
$\exists$	$\sim_\exists$	There exist two constants c and c' whose hashes collide $H[c] = H[c']$;	$c \sim_\exists c'$
fstPreImg	$\sim_1$	Given $o = H[t]$, the adversary can compute a preimage $t' = \mathsf{pi}^1(o)$ such that $H[t'] = o$;	$t \sim_1 \mathsf{pi}^1(H[t])$
sndPreImg	$\sim_2$	Given t, the adversary can compute a second preimage $t' = \mathsf{pi}^2(t)$ such that $H[t'] = H[t]$;	$t \sim_2 \mathsf{pi}^2(t)$
chsnPrfx	$\sim_{\mathsf{CP}}$	Given t, t', the adversary can compute $u = \mathsf{cp}_1(t,t')$ and $u' = \mathsf{cp}_2(t,t')$ such that $H[t\|u] = H[t'\|u']$;	$t\|u \sim_{\mathsf{CP}} t'\|u'$
idtclPrfx	$\sim_{\mathsf{IP}}$	Given t, the adversary can compute $u = \mathsf{sp}_1(t)$ and $u' = \mathsf{sp}_2(t)$ such that $H[t\|u] = H[t\|u']$;	$t\|u \sim_{\mathsf{IP}} t\|u'$
allCol	$\sim_\top$	All hash outputs can collide, which models the worst possible collision case;	$\forall t,t' : t \sim_\top t'$
hashExt	$\sim_{\mathsf{LEa}}$	Length-extension collision. Given $H[x]$ and s, the adversary can compute $H[x\|s]$;	$x\|s \sim_{\mathsf{LEa}} H[x]\|s$
colExt	$\mathsf{LEc}(\sim_c)$	Length-extension closure. $H[x\|s]$ collides with $H[y\|s]$, if $H[x] = H[y]$ (based on any of the previous capabilities).	

Table 2: Intuition behind the basic collision-relations $\sim_c$ depending on the chosen adversarial capabilities. Many of these relations define that there exist collisions that can be computed for very specific, but not all, input patterns. Collision-relations in different dimensions can be combined by taking their union. We give the formal definitions in Table 6.

Example 3. *The weakest threat model capturing the CPC attack from Section 2.2.1 is {chsnPrfx,colExt}. Indeed, the attack is possible provided that CPC exist (chsnPrfx) and can be extended due to the length-extension property (colExt).*

Remark 1. *Second preimage resistance (sndPreImg) implies preimage resistance (fstPreImg) in our modeling, while in the computational models with the same name [38] this is not the case. This is an* intended *consequence of our modeling choices with an orthogonal inLeak capability, which causes us to use slightly different definitions in this case despite using the same names. inLeak expresses the ability for the adversary to compute* the *preimage of a given hash output. fstPreImg expresses the ability to compute* some *input* $\mathsf{pi}^1(H[t])$ *that when hashed yield a given, target output* $H[t]$ *but that is* always different *from the original input t and similarly for sndPreImg. For this modeling choice in our symbolic model, the above implication indeed holds.*

Cross-dimension implications. The previous lattice contains some redundant capabilities that are not captured by the joint partial order. For instance, we have that *allCol* $\Rightarrow$ *hashExt*, *sndPreImg* $\wedge$ *inLeak* $\Rightarrow$ *fstPreImg* and *inLeak* $\Rightarrow$ *hashExt*. Further, if on a dimension the protocol is secure at some given level, it is secure for all weaker threat models (*i.e.,* levels below), and conversely for attacks.

4 Automation methodology

We now present different ways to automate verification for the previously defined threat models by casting them in both TAMARIN and PROVERIF, two of the most widely used tools with distinct active user communities. In general, they offer incomparable features, reasoning engines, and input languages and tend to perform better on different examples. Developing our methodology for both greatly increases the potential targets and users, and demonstrate the generality of the framework, which is not tailored to a given tool.

We first explore a direct way of modeling these capabilities as equational theories (Section 4.1), which is the classical solution in symbolic tools but limited here to a few capabilities due to a lack of tool support for more advanced equations.

We then explain how we overcome this by extending TAMARIN (Section 4.2) yielding a fully automated tool capable of exploring all of our threat model lattice. We achieve this by extending TAMARIN with an associative concatenation operator in order to provide a more precise model for collisions and length extensions. We build on the latter and model hash function computations by a call to an oracle process, in the flavor of Figure 4. This process allows fine-grained control over the output values and allows for a logical specification of the set of possible collisions and can be fully automated in TAMARIN, which we then use as back-end to automatically explore our lattice of threat models and obtain the weakest threat models under which attacks were found (if any).

Finally, we discuss how we extended PROVERIF to allow modeling any threat model of our lattice (Section 4.3). We developed for this a new feature in PROVERIF that allows to define powerful recursive predicates that we use to approximate an equational modeling of the $\|$ and MD construction, enabling reasoning on length extensions, IPC, and CPC. We explain how to produce those PROVERIF models but automating this task further is left as future work.

4.1 Equational theory based modeling

As explained in Section 2.3, the classical way to model cryptographic primitives in a symbolic model is to use an equational theory, that specifies which operations yield equal values.

4.1.1 Capabilities modeled as operators

Using an equational theory allows us to naturally and efficiently explore the set of scenarios corresponding to the capabilities fstPreImg, sndPreImg, ∃, as well as inLeak. Each threat scenario for a given dimension can be encoded by the following corresponding equation that we explain below.

$$\begin{array}{rlrl} \textit{fstPreImg} & \mathsf{H}(\mathsf{pi}^1(z)) = z & \exists & \mathsf{H}(c) = \mathsf{H}(c') \\ \textit{sndPreImg} & \mathsf{H}(\mathsf{pi}^2(z)) = \mathsf{H}(z) & \textit{inLeak} & \mathsf{i}(\mathsf{H}(z)) = z \end{array}$$

We note that the absence of a weakness for each dimension is the default in symbolic models and does not require any particular modeling. For each threat model we simply add the corresponding equations for the collision and input leak dimension.

To express that a hash function is not *preimage resistant* (fstPreImg) we provide the attacker with an explicit function pi^1 that allows the attacker to compute a preimage of a hash z, *i.e.,* a value $\mathsf{pi}^1(z)$ that hashes to z. The absence of *second preimage resistance* (sndPreImg) is similarly expressed by the function pi^2.[1] The absence of *collision resistance* (∃) relies on two distinguished constants c and c' on which the hash function collides. These do not have any particular structure and are not attacker chosen, modeling that a collision merely *exists*. Finally, we model that a hash function may leak (part of) its input (inLeak) by giving the attacker the capability to inverse the function using the symbol i, and no leak being the default does not require an equation.

We encoded this part of the hierarchy in both TAMARIN and PROVERIF, and used it on multiple case-studies as presented in Section 5.1. As we will see, while such an equation based model is easy to deploy using symbolic tools, it is also rather weak (and not a very effective way of finding attacks): the equations are basically obtained by negating the security assumption and model the existence of a collision, or (second) preimage without giving the attacker any additional control. (We exemplified this gap with Sigma’ in Section 2.2.1.) On the other hand, when finding an attack with this model generally translates directly to a missing assumption on the security of the used hash function.

4.1.2 Challenges with modeling associative $\|$ and MD

To go beyond the above *existential* modeling of weaknesses in hash function we will give a more detailed model of the associative $\|$ operator and of the MD construction whose properties can be exploited by an adversary. This will allows us to explore IPC, CPC, and length extension attacks, which are not covered using equations from the previous section.

The presence of an associative concatenation operator is required if we want to capture IPC and CPC. Indeed, recall that given prefixes, p_1 and p_2, a CPC attack computes suffixes s_1, s_2 such that $\mathsf{H}(p_1\|s_1) = \mathsf{H}(p_2\|s_2)$. If, for example, a protocol participant computes the transcript $\mathsf{H}(m_1\|m_2\|m_3)$ and the attacker controlled parts are m_2 and m_3, then the suffix in the previous equation needs to be $m_2\|m_3$. With a non-associative concatenation operator (that we denote $\langle\cdot,\cdot\rangle$) the CPC attack would fail as $\mathsf{H}(\langle\langle m_1, m_2\rangle, m_3\rangle)$ would not be identified with $\mathsf{H}(\langle m_1, \langle m_2, m_3\rangle\rangle)$. This raises a challenge for the existing tools, as neither TAMARIN nor PROVERIF allow to model such an *associative operator*. The core difficulty is that both tools rely on unification. Given two messages one needs to be able to compute a finite and complete set of most general unifiers, *i.e.,* a set of substitutions that represents all possible ways of instantiating the messages that make them equal. For instance, $\langle x, 0\rangle =^? \langle 1, y\rangle$ has a unique unifier $\{x \mapsto 1, y \mapsto 0\}$. For associative operators, the issue is that such a set is not always finite. For instance, $0\|x =^? x\|0$ has an infinite set of unifiers of the form $\{x \mapsto 0^n \mid n \in \mathbb{N}\}$. A first approach is to model the associativity under a hash function operator for a bounded depth only, for instance specifying that $h(\langle x, \langle y, z\rangle\rangle) = h(\langle\langle x, y\rangle, z\rangle)$, but this does not imply that $h(\langle x, \langle y, \langle z, z'\rangle\rangle\rangle) = h(\langle\langle\langle x, y\rangle, z\rangle, z'\rangle)$. We use such a bounded modeling successfully on some examples in Section 5. However, this modeling may miss attacks that require associativity at a deeper depth than modeled. We explain how we overcame this problem in both tools in the next sections.

Furthermore, a naive way to encode the MD construction would be to directly consider the equation $\mathsf{H}(\langle x, y\rangle) = f(\mathsf{H}(x), y)$. However, such an equation is out of scope of both TAMARIN and PROVERIF as it cannot be completed into a convergent rewrite system, which seems to be an inherent difficulty for all tools based on equational reasoning.

[1] The equation for pi^2 does not work out of the box for a technical reason that we describe in Appendix A.

4.2 TAMARIN extension for the full lattice

Extension for the $\|$ operator. We extended TAMARIN to obtain partial support for associative operators such as $\|$. The key observation is that we do not need the unification problem for an associative operator to yield a finite set in general: it is sufficient that all *particular* unification problems that actually appear in a protocol's verification have a finite set of unifiers.

TAMARIN relies on the MAUDE tool as a backend to perform equational unification. Although unification for an associative operator is infinitary, support has recently been added in MAUDE [22]: it either returns the complete set of unifiers, or only a subset but with a warning. We integrated this new feature in TAMARIN, which now has a built-in associative concatenation operator denoted by $\|$. To ensure correctness, TAMARIN stops the analysis as soon as MAUDE raises a warning. In particular, as we use $\|$ under a hash function only, our case studies (Section 5) illustrate that TAMARIN encountered this MAUDE warning in rare occasions only[2]. The soundness proof of this extension is provided in Appendix C.

Event based modeling. The equational based models presented above have several drawbacks:

- The equations for computing collisions, and (second) preimages are *existential* and do not give the adversary any control over the computed messages, missing most of our threat model (*e.g.,* CPC) and practical attacks.
- The associative operator added to TAMARIN increases the complexity of the equational theories often leading to non-termination or extensive verification times when modeling CPC and length extension attacks.
- The previous models do not cover the OC dimension.

To overcome these limitations, instead of using an operator to model a hash computation, we define, in parallel to the protocol processes, a dedicated process for computing the hash function, in the spirit of Figure 4. Notably this approach allows to either sample the hash value from a fresh set of values ($\emptyset$) or query the output values to the attacker (frshTarget or anyTarget), *i.e.,* let the attacker choose the value. By default, this process is free to create *any* collisions. To restrict this, we will give a *logical specification* when precisely collisions are allowed. More precisely, *(i)* whenever a hash output value o is returned for some input i, we raise an event $\mathsf{Hash}(i,o)$, and *(ii)* we specify that when a trace contains two events $\mathsf{Hash}(i,o)$ and $\mathsf{Hash}(i',o)$, *i.e.,* two inputs i and i' are mapped to the same output o, then we must have $i \sim i'$ for the desired $\sim$ relation. Otherwise the trace is discarded. For example, if $\sim$ is the identity relation we do not allow any collision; if $\sim$ relates all inputs then arbitrary collisions are allowed.

Discarding traces is achieved using TAMARIN's *restrictions*, which are logical formulae considered as part of the specification and where any execution that does not satisfy it is discarded. We can represent a simplified threat model of {chsnPrfx,colExt} with a restriction that requires that if a collision occurs for i and i', then it must be a length-extended CPC: the $\mathsf{cp}_i(p_1,p_2)$ represent the attacker chosen suffix for prefixes p_1 and p_2 (chsnPrfx) and l is a length extension on both inputs (colExt), which is formally written as:

$$\begin{aligned} &\forall i,i',o.\ \mathsf{Hash}(i,o)\ \&\ \mathsf{Hash}(i',o)\ \&\ i \neq i' \\ &\qquad \Rightarrow \exists p_1,p_2,l.\ (\quad i = p_1\|\mathsf{cp}_1(p_1,p_2)\|l \\ &\qquad\qquad\qquad\qquad \&\ i' = p_2\|\mathsf{cp}_2(p_1,p_2)\|l) \end{aligned}$$

Plug-and-Play library and tooling. Using this approach, we define a modular library for hash functions that allows TAMARIN to explore the full lattice of capabilities. We developed a Python script that allows to check a given protocol for all possible scenarios, only exploring non-redundant scenarios and outputting the minimal threat models under which an attack was found. This yields a push-button tool that produces results that can be as compact, yet comprehensive, as those shown in Table 4.

4.3 Extending PROVERIF for the full lattice

Unlike TAMARIN, PROVERIF does not use MAUDE and does not already support associative commutative operators to base an associative operator on. Hence we took a different approach: as in the previous event-based model we start with a very permissive model, allowing arbitrary collisions, and refine this model using axioms for a given threat model.

Whether two hashes collide depends on a *predicate* `eq_hash`. Instead of giving the precise semantics of this predicate (which would lead to non-termination), we consider the predicate as an *uninterpreted operator* considering arbitrary behaviors. Then, given a threat model, we refine the predicate by specifying some carefully chosen properties that we know the predicate satisfies. For example, for a hash function based on the MD construction (where `f` is the compression function) with an associative $\|$, we could define the following axiom:

```
axiom h1,h2,x:t_output;
   eq_hash(f(h1,x),f(h2,x)) ⟹ eq_hash(h1,h2);
   eq_hash(h,h) ⟹ true;
```

Ideally, we would like to state that if `eq_hash(f(h1,u1),f(h2,u2))` then `u1` and `u2` are equal. However, this is not necessarily the case, as `u1`, `u2` may have been instantiated by messages made of the concatenation of multiples blocks that make them different in the equational theory (as concatenation is not associative); *e.g.,* for `u1=<<a,b>,c>` and `u2=<a,<b,c>>`. In such a case, the messages `f(h1,u1)` and `f(h2,u2)` were not properly computed as in the MD construction.

To provide a more effective model of $\|$ and the MD construction, we extend PROVERIF by adding *recursive computation functions*: powerful generic user-defined functions to manipulate messages that are fully integrated in

[2] The warning only occurred with one the security properties out of the 21 we verified, and even then only for a particular threat-model.

the reasoning engine. For instance, we define the computation function `H` that recursively computes the MD construction (removing the need for an associative $\|$ operator). The soundness of this extension is provided in Appendix D.

```
compfun H(t_output):bitstring =
  forall x:bitstring;
    H(x) if is_var(x) | x = Nil → x
  otherwise forall x1,x2,h:bitstring;
    H(f(h, <x1,x2> )) → H(f(f(h,x1),x2))
  otherwise forall x,h:bitstring;
    H(f(h,x)) → f(H(h),x).
```

For each given input, the computation function tries to match the left-hand side messages and the possible conditions of the given rules sequentially. The first successful rule is applied, *i.e.,* its right hand side message is computed and returned. We suppose that to compute the hash of a term `t` we write `f(Nil,t)`; this can be conveniently hidden to the user by an auxiliary function **letfun** `hash(x:bitstring)=f(Nil,x)`. Then we indeed have that both `H(hash(<<a,b>,c>)` and `H(hash(<a,<b,c>>))` evaluate to `f(f(f(Nil,a),b),c)`.

Computation functions can be used in the specification of axioms. For instance, the following axiom replaces the hash values in the predicates with their MD construction.

```
axiom h,h1,h2:t_output;
  eq_hash(h,h1) && h2 ← H(h1)  ⟹  eq_hash(h,h2).
```

5 Case studies

Using the techniques from the previous section, we have analyzed 20 protocols: 15 of them are based on existing TAMARIN models and 5 are new case studies. We first analyze all of these protocols using the equational theory based hash models in TAMARIN (Section 5.1), exemplifying the limitations of this approach. We then perform an in-depth analysis of the five new models using the event based modeling, completely automating the exploration of the threat model lattice with TAMARIN. We present our analysis methodology in Section 5.2 and a selection of results in Section 5.3. Finally, to complement some of our results, and as a proof of concept to advocate for the generality of our approach, we also used PROVERIF as described in Section 4.3 to analyze two of the original protocol models (available at [15]). The results found with PROVERIF are coherent with the findings of Section 5.3.

5.1 Equational theory based hash models

Using the equational theories described in Section 4.1, we analyzed all case studies mentioned previously, and even with the strongest threat model in the hierarchy described in Section 4.1.1 without input leak ({fstPreImg}), only one potential attack is found, which illustrate the limitation of the equational based modeling. This attack is against the TESLA protocol (v1), which instantiates a Pseudorandom Function (PRF) with a weak construction (HMAC-MD5), but is insecure as soon as preimages can be found. Adding the equation for input leak (Section 4.1.1) results in the scenario {fstPreImg,inLeak}, and triggered regular non-termination issues in TAMARIN. For many protocols the input leak resulted in potential attacks on secrecy. For this particular set of case studies, this is not surprising as the hash function is applied to cryptographic keys. We list the full results in the Appendix in Table 5.

5.2 Fully automated analysis methodology

As described in Section 4.2, we developed a TAMARIN library that allows verification of a specified threat model in the lattice of hash weaknesses. To automate a systematic exploration of the full lattice of threat models, we developed a Python program that computes the set of all minimal, respectively, maximal scenarios that invalidate, respectively, validate the security goals. It allows for parallel verification, and avoids redundant exploration: once a property is falsified for a threat model, we automatically conclude that it is falsified for all stronger threat models (and conversely for verified properties). We also exploit cross-dimension implications discussed in Section 3.1, and avoid calling TAMARIN systematically for each of the 264 distinct scenarios of the lattice. As a heuristic, we start by verifying the set of strongest and weakest threat model, as they may allow to quickly prune the search space.

For a given protocol, TAMARIN may find an attack on some threat model in a very short time, but take much longer to find the same attack in a more complex threat model, and the converse may happen for a security proof. Thus, the optimal order of verifying the scenarios is protocol dependent. We therefore first run each analysis with a short timeout, to ensure that we first find all easy proofs and attacks. We then immediately conclude implied results and prune the corresponding scenarios. We then re-run the remaining scenarios with a longer timeout. We show an example of a fully automatically generated table for Sigma' in Table 4 (detailed in Section 5.3.1).

After the initial, fully automated analysis, one can perform a more in-depth analysis of the attacks found. Notably, multiple attacks may exist for a given threat model, and by default the tools return the first attack found. This may "hide" some interesting attack variants and can cause PROVERIF and TAMARIN to initially report different variants. TAMARIN's *interactive mode* can be used to semi-automatically find *all* variants of attacks for a given threat model. We used both tools to obtain the attack variants discussed in Section 5.3.1.

Experimental results We first discuss the overall performance of our approach, where experiments were performed on an Intel(R) Xeon(R) CPU E5-4650L 2.60GHz server with 756GB of RAM, with 8 threads for a TAMARIN call while PROVERIF is inherently single threaded.

We first ran a test set to compare the efficiency of our new event based threat model (Section 4.2) to the classical

Protocol	Broken properties	Main attack requirements	New?	In-text ref.	Time (s)	Note
Sigma	Sec,Agr(transcript)	chsnPrfx,colExt	[9]	AT(S1)	28	
	Sec,Agr(transcript)	chsnPrfx,colExt	∼ [9]	AT(S2)	manual	collisions on shares
	Sec,Agr(transcript,role)	chsnPrfx	new	AT(S3)	55	role-confusion, no need for colExt
SSH	Agr(nego)	CI(*)	new	AT(SSH1)	3	see Figure 6
	Agr(nego)	idtclPrfx,colExt	[9]	AT(SSH2)	28	
	Agr(nego)	CI(I),sndPreImg,colExt	new	AT(SSH3)	41	
IKEv2	Sec(R)	CI(*)	new	AT(IKE1)	6	CI should be on the cookie
	Auth(I)	idtclPrfx,colExt	[9]	AT(IKE2)	20	
	Agr(cookie,transcript)	∃,colExt	new	AT(IKE3)	9	disagreement on cookies only
Flickr	Auth(I)	hashExt	[21]	AT(F)	9	

Table 3: A selection of the most meaningful attacks we found whose details are given in Section 5. Those attacks are at the design level and their severity depends on whether the attack requirements are met given a specific implementation and threat model. Time: number of minutes it takes for our tool and Tamarin to find the attack. Sec: Secrecy of session data (*e.g.,* session key); only from a given role's perspective if specified, Agr(data): agreement on data, note that disagreement on negotiation data (nego) can lead to downgrade attacks, Auth(X): authentication of role X (R: responder, I:initiator, *:both), ∼: new variant of an existing attack, CI(X): role X must suffer from colliding inputs; see Section 5.3.2.

equational based modeling. We executed all protocols Table 3 first with a perfect ROM model using a basic function symbol and non-associative concatenation, and then with the perfect ROM model part of our event based approach with associative concatenation. Overall, most protocols verify with almost exactly the same time, and some outliers take up to five times longer. Given the substantially increased expressivity, this is very encouraging. We provide detailed results in the long version [16].

All detailed tables are available in the long version [16]: Overall about 5000 scenarios were verified in 150 minutes through 1600 TAMARIN calls. Verification times vary substantially among protocols: most protocols only take a few minutes, while two particular models (SSH and IKE without neutral DH element) take around an hour to complete. Overall, our pruning strategy was very effective: about two thirds of the scenarios were not verified through a TAMARIN call but directly implied by another one. We provide detailed results in the long version [16].

We also checked the verification times needed to automatically find our main attacks (cf. Table 3). Most of our attacks stem from a few scenarios that we identify as the most interesting, and the time needed to uncover an attack under those scenarios is consistently under a minute for our protocols.

Finally, we also measured the performances of PROVERIF in the CPC setting with the MD construct over Sigma and IKE (cf. the long version [16]) , and found them to be in the same order of magnitude as TAMARIN, supporting the generality of our approach. We however stress that that the tool timings are often incomparable in practice: while PROVERIF tends to be faster than TAMARIN for the larger case studies in our set, PROVERIF is single threaded and TAMARIN is multi-threaded, and both operate inherently differently with respect to specific threat models. We mainly chose TAMARIN as the main tool for the automatic lattice exploration because it was more natural to express the full lattice with it.

5.3 Results from automated analysis

We now report on the results obtained by running our TAMARIN-based automatic tool for exploring our lattice of threat models. We first detail the results for Sigma' in Section 5.3.1 and then discuss in Section 5.3.2 a selection of other attacks and insights. In Table 3 we summarize the most interesting attacks that our method automatically found and that we describe in the remainder of this section (we refer to attacks with labels such as AT(S1)). Our attacks are at the design level: their severity depends on whether the discovered attack requirements (including the choice of hash primitive) are met given a specific implementation, threat model and use case.

5.3.1 Detailed analysis results for Sigma'

We analyzed mutual authentication and key secrecy for Sigma'. As argued in [9], even though Sigma' is not deployed, its protocol logic is similar to many widely deployed authentication protocols such as TLS, SSH, IKEv2, which makes it an interesting and relevant case study. The output of our tool for this protocol model is shown in Table 4. Each row contains either one of the strongest threat models under which all of the three properties hold or one of the weakest threat model under which one of the properties is violated. For Sigma', they were actually all violated as soon as one was. This kind of tables (more of them are in the long version [16]) allow to concisely and yet comprehensively describe the security level against any of the threat models in the lattice.

How to read threat model tables. As an example, consider the last row of Table 4: the protocol is broken if CPCs are possible (chsnPrfx) and can be extended thanks to colExt,

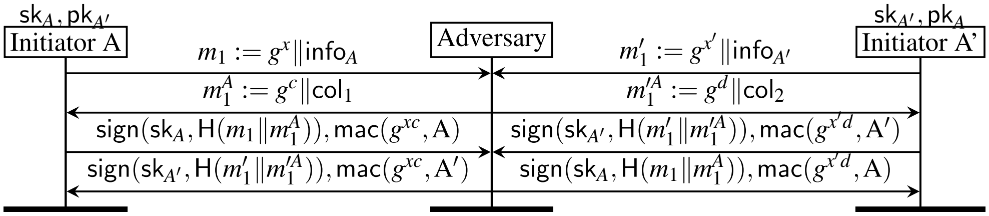
$\mathsf{sk}_A, \mathsf{pk}_{A'}$
Initiator A
$m_1 := g^x \| \mathsf{info}_A$
Adversary
$m_1' := g^{x'} \| \mathsf{info}_{A'}$
$\mathsf{sk}_{A'}, \mathsf{pk}_A$
Initiator A'
$m_1^A := g^c \| \mathsf{col}_1$
$m_1'^A := g^d \| \mathsf{col}_2$
$\mathsf{sign}(\mathsf{sk}_A, \mathsf{H}(m_1 \| m_1^A)), \mathsf{mac}(g^{xc}, \mathrm{A})$
$\mathsf{sign}(\mathsf{sk}_{A'}, \mathsf{H}(m_1' \| m_1'^A)), \mathsf{mac}(g^{x'd}, \mathrm{A'})$
$\mathsf{sign}(\mathsf{sk}_{A'}, \mathsf{H}(m_1' \| m_1'^A)), \mathsf{mac}(g^{xc}, \mathrm{A'})$
$\mathsf{sign}(\mathsf{sk}_A, \mathsf{H}(m_1 \| m_1^A)), \mathsf{mac}(g^{x'd}, \mathrm{A})$

Figure 5: The new CPC attack we found on Sigma'. $\mathsf{col}_i = \mathsf{cp}_i(L_1, L_2)$ for $i \in \{1, 2\}$ where $L_1 := g^x \| \mathsf{info}_A \| g^c$ and $L_2 := g^{x'} \| \mathsf{info}_{A'} \| g^d$. Therefore $\mathsf{H}(m_1 \| m_1^A) = \mathsf{H}(m_1' \| m_1'^A)$ so that A's and A''s signatures can be reused.

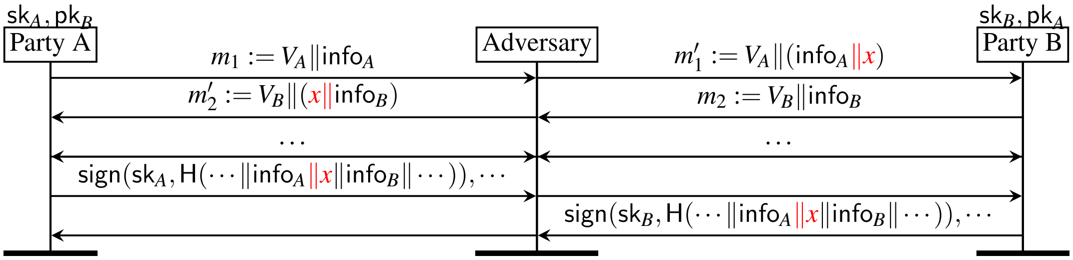
$\mathsf{sk}_A, \mathsf{pk}_B$
Party A
$m_1 := V_A \| \mathsf{info}_A$
Adversary
$m_1' := V_A \| (\mathsf{info}_A \| x)$
$\mathsf{sk}_B, \mathsf{pk}_A$
Party B
$m_2' := V_B \| (x \| \mathsf{info}_B)$
$m_2 := V_B \| \mathsf{info}_B$
$\ldots$
$\ldots$
$\mathsf{sign}(\mathsf{sk}_A, \mathsf{H}(\cdots \| \mathsf{info}_A \| x \| \mathsf{info}_B \| \cdots)), \cdots$
$\mathsf{sign}(\mathsf{sk}_B, \mathsf{H}(\cdots \| \mathsf{info}_A \| x \| \mathsf{info}_B \| \cdots)), \cdots$

Figure 6: Attack AT(SSH1) on SSH: In the set-up phase where A sends the set of its allowed algorithms info_A, the adversary alters info_A by extending it with some message x. The adversary then alters info_B sent by B by putting x before it. Despite the different input messages, the concatenation of info_A and info_B results in the same transcript $\cdots \| \mathsf{info}_A \| x \| \mathsf{info}_B \| \cdots$ on both side.

Threat Models				Auth.
COL	LE	OC	IL	
fstPreImg,chsnPrfx	hashExt	anyTarget	inLeak	✓
allCol				✗
fstPreImg,idtclPrfx	allExt	anyTarget	inLeak	✓
chsnPrfx	colExt			✗

Table 4: Sigma' analysis. ✓: security holds, ✗: attack found.

even when hash outputs are completely fresh values. However, without either colExt or chsnPrfx, the protocol is deemed secure (otherwise this threat model would not represent a minimal violation). Inspecting the attack trace returned by TAMARIN for this threat model, we observe that it corresponds to the CPC attack AT(S1) from [9] described in Section 2.2.1.

The second row shows that an attack occurs if collisions are unconstrained (allCol). This was to be expected as such collisions subsume through a cross-dimension implication CPC with colExt. While the automated analysis uses those implications to prune the search space, we display for clarity all minimal results for the basic partial order of the lattice.

The third row shows that even for the strongest OC, LE, and IL capabilities, the protocol is deemed secure if CPC is impossible (as shown by the COL capability, which is the strongest among the ones that are strictly weaker than chsnPrfx). Similarly, the first row shows that even for the strongest OC and IL capabilities and for {fstPreImg,chsnPrfx}, we find no attacks as long as colExt is impossible.

New attacks. We automatically found the CPC attack from [9], described in Section 2.2.1. We additionally found a variant of this attack where the CPC is replaces the DH shares g^x, g^y instead of the $\mathsf{info}_A, \mathsf{info}_B$ fields. This variant AT(S2) seems harder to exploit as it is unlikely that DH shares allow a large enough search space for finding the CPC.

More interestingly, we also automatically found a novel CPC attack AT(S3). The attacker acts as a MIM in-between two agents A and A' both acting as initiator. It is able to impersonate A towards A', who believes A is acting as a responder, and A' towards A, who believes A' is acting as a responder. The adversary additionally learns both session keys. This attack thus violates session key secrecy, authentication, and agreement on the agents' role. The attack is depicted in Figure 5. A simple fix for this attack is to include the role name under the signature to avoid such role confusion. We show automatically that the attack disappears with this fix.

Finally, we report on a reflection attack that does not require any collision: the attacker simply reflects A's messages to herself, breaking agreement on the agent role. This attack requires that A accepts a session with an agent having the same public key which could be realistic in some specific use cases (*e.g.,* shared credentials, group authentication). The previous fix avoids this attack (and we proved it); as an alternative, an agent could check that her peer's public key is different from her own public key.

5.3.2 Selection of other attacks and insights

Colliding input attacks on SSH and IKE. Using our methodology, we found *colliding input attacks* on both, SSH and IKEv2. Colliding input attacks allow an adversary to alter exchanged messages between two agents such that they are parsed differently by both agents but yield the same bitstring when they are recomposed into a bitstring *prior* to hashing, hence not relying on hash collisions or other hash weakness. An example of such an attack AT(SSH1) is depicted in Figure 6 in the case of SSH, where the attacker can alter the messages such that both parties agree on different sets of algorithms in the set-up phase. The attack relies on the fact that, before hashing, fields of previous messages are concatenated into a bitstring, hence losing the message structure: $(\mathsf{info}_A \| x) \| \mathsf{info}_B$ is the same as $\mathsf{info}_A \| (x \| \mathsf{info}_B)$. For such attacks to be realistic, the protocol needs to drop some length fields (which initially allowed unambiguous message parsing) before hashing or be very liberal about parsing, *e.g.,* accept any list of length-prefixed fields and process them until the input buffer is empty without requiring knowledge of the list size in advance. Our automated verification framework is able to capture such attacks and did so for both SSH and IKE; it is up to the user to assess if such attacks may occur in actual implementations.

While this attack is not practical here, it occurred in an early version of the *EU Federation Gateway Service* [23] for the EU's Covid-19 contact tracing apps due to the hash computation that concatenated two variable-length fields. The attack allowed circumvention of accountability, *e.g.,* for the gateway server to manipulate country data or for countries to repudiate data that they uploaded; it was fixed before deployment.

IKEv2. While the previously mentioned input colliding attack AT(IKE1) seems impractical, it can be made practical by changing the length information at the start of the cookie through an IPC. We automatically found this stronger attack variant AT(IKE2), which corresponds to the one documented in [9], for the threat model {idtclPrfx,colExt}. We also found a low-severity attack AT(IKE3) under {∃,colExt} where the adversary breaks transcript agreement as it uses two colliding hash inputs as the cookies for the initiator and responder.

SSH. In addition to the previous colliding input attack, we automatically found the IPC attack AT(SSH2) from [9] for the threat model {idtclPrfx,colExt}. This is similar to the attack escalation of IKE and appears to be a recurring pattern.

Our analysis also revealed the attack AT(SSH3) under {sndPreImg,colExt} that exploits a second preimage attack and the specific way the transcript is reorganized prior to being hashed; the attack would not be possible if the hash input would simply be the transcript. However, it requires the initiator to be liberal in the way it parses B's negotiation information (similarly to input colliding attacks). This attack violates transcript agreement and allows a MIM attacker to completely tamper with the negotiation information sent by the initiator to the responder, potentially enabling downgrade attacks.

Telegram. We modeled Telegram's key exchange protocol described in [50] and [1, Figure 57]. As some output of the hash function is used as a secret, the minimal scenario scenario for an attack AT(T) is naturally {frshTarget}. However, we found no attack even under the strong threat model {fstPreImg,chsnPrfx,allExt,inLeak}. This seems to indicate that Telegram is secure despite using SHA1 and SHA2-256 as hash functions. The first attack however indicate that PRF like assumptions are needed to prove the security of the protocol, which has been independently identified in [1], and Telegram would benefit from upgrading their hash functions.

Flickr. For the old API of Flickr, we automatically found the length-extension attack AT(F) first documented in [21] under the threat model {hashExt}. We also found variants of these attacks relying either on the inLeak or the fstPreImg capabilities. Our methodology could have then easily spotted the design flaws of this API. After the discovery of the first attack [21], Flickr migrated to the OAuth framework

6 Further related work

Besides the works already mentioned in Section 2, there are many works on hash functions, their design, and their cryptographic properties (see *e.g.,* [38, 39, 43]).

The ROM shares features with the classical symbolic (Dolev Yao) model of hash functions used in nearly all previous symbolic analyses. One exception is the automated TLS 1.3 analysis in [8], which explicitly considers the possibility of a very weak hash function where all inputs collide.

The use of the ROM in symbolic models also connects to the so-called computational soundness question: can we obtain computational guarantees from a symbolic proof? In this field, the only two works that consider hash functions model them as a random oracle [3, 18]. Additionally, [3] shows an impossibility result in the standard model for a hash function that is symbolically represented as a free symbol.

7 Conclusions

We provided the first systematic, automated methodology to discover protocol attacks that exploit weaknesses of hash functions. Our methodology finds attacks that cannot be detected either in previous symbolic analysis or in computational analyses that use the ROM. All our results can be inspected and reproduced using the docker image at [15]. Our extensions to TAMARIN and PROVERIF are not hash-specific and are therefore of independent interest, opening up new applications.

Our case studies reveal several new attack variants, but we did not find a completely new break that warranted, *e.g.,* responsible disclosure. We conjecture that this is because several of our case studies overlap with earlier manual analyses [1, 9]. In future work, we want to apply our methodology to more complex case studies such as full TLS, the EMV payment standard (which uses SHA1), and some blockchain applications, which are even harder to analyze manually.

Acknowledgments

This work has been partly supported by the ANR Research and teaching chair in AI ASAP (ANR-20-CHIA-0024) and ANR France 2030 project SVP (ANR-22-PECY-0006).

References

[1] Martin R Albrecht, Lenka Mareková, Kenneth G Paterson, and Igors Stepanovs. Four Attacks and a Proof for Telegram. Long version, https://mtpsym.github.io/paper.pdf. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[2] Valerie Aurora. Lifetimes of cryptographic hash functions, 2017. https://valerieaurora.org/hash.html (Retrieved Jan 2022).

[3] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the BRSIM/UC Soundness of Dolev-Yao Models with Hashes. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2006.

[4] Elaine Barker, Lidong Chen, Andrew Regenscheid, and Miles Smid. Recommendation for pair-wise key establishment using integer factorization cryptography. In *Special Publication (NIST SP), National Institute of Standards and Technology*, 2009.

[5] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirović, Ralf Sasse, and Vincent Stettler. A Formal Analysis of 5G Authentication. In *Conference on Computer and Communications Security (CCS)*. ACM, 2018.

[6] David Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV Standard: Break, Fix, Verify. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2021.

[7] Mihir Bellare and Phillip Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, 1993.

[8] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[9] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2016.

[10] Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. In *Selected Areas in Cryptography (SAC)*. Springer, 1998.

[11] Simon Blake-Wilson and Alfred Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC)*. Springer, 1999.

[12] Bruno Blanchet, Vincent Cheval, and Cortier Véronique. Proverif with lemmas, induction, fast subsumption, and much more. In *Proceedings of the 43th IEEE Symposium on Security and Privacy (S&P'22)*. IEEE Computer Society Press, May 2022.

[13] Srdjan Capkun, Levente Buttyán, and Jean-Pierre Hubaux. SECTOR: secure tracking of node encounters in multi-hop wireless networks. In *Workshop on Security of ad hoc and Sensor Networks (SASN)*. ACM, 2003.

[14] Sanjit Chatterjee, Alfred Menezes, and Berkant Ustaoglu. A Generic Variant of NIST's KAS2 Key Agreement Protocol. In *Australasian Conference - Information Security and Privacy (ACISP)*. Springer, 2011.

[15] Vincent Cheval, Cas Cremers, Alexander Dax, Hirschi Lucca, Charlie Jacomme, and Steve Kremer. Docker image and models. https://github.com/charlie-j/symbolic-hash-models.

[16] Vincent Cheval, Cas Cremers, Alexander Dax, Hirschi Lucca, Charlie Jacomme, and Steve Kremer. Long version of this paper. https://hal.archives-ouvertes.fr/hal-03795715.

[17] Véronique Cortier, David Galindo, and Mathieu Turuani. A Formal Analysis of the Neuchatel e-Voting Protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018.

[18] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Springer, 2006.

[19] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A Comprehensive Symbolic Analysis of TLS 1.3. In *Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[20] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 1981.

[21] Thai Duong. Flickr's API signature forgery vulnerability, 2009. https://vnhacker.blogspot.com/2009/09/flickrs-api-signature-forgery.html (Retrieved Jan 2022).

[22] Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. Associative Unification and Symbolic Reasoning Modulo Associativity in Maude. In *International Workshop on Rewriting Logic and Its Applications (WRLA)*. Springer, 2018.

[23] EU Federation Gateway Service (EFGS), 2020. https://github.com/eu-federation-gateway-service/efgs-federation-gateway (Retrieved Jan 2022).

[24] Ik Rae Jeong, Jonathan Katz, and Dong Hoon Lee. One-round protocols for two-party authenticated key exchange, 2008.

[25] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, 2014.

[26] John Kelsey and Tadayoshi Kohno. Herding hash functions and the nostradamus attack. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 183–200. Springer, 2006.

[27] Chong Hee Kim and Gildas Avoine. RFID Distance Bounding Protocol with Mixed Challenges to Prevent Relay Attacks. In *Cryptology and Network Security (CANS)*. Springer, 2009.

[28] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[29] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *Advances in Cryptology - CRYPTO 2003*. Springer Berlin Heidelberg, 2003.

[30] Brian A. LaMacchia, Kristin E. Lauter, and Anton Mityagin. Stronger Security of Authenticated Key Exchange. In *Provable Security, First International Conference, ProvSec*. Springer, 2007.

[31] Kristin E. Lauter and Anton Mityagin. Security Analysis of KEA Authenticated Key Exchange Protocol. In *International Conference on Theory and Practice of Public-Key Cryptography*. Springer, 2006.

[32] Gaëtan Leurent and Thomas Peyrin. Sha-1 is a shambles: First chosen-prefix collision on sha-1 and application to the PGP web of trust. In *USENIX Security Symposium*. USENIX Association, 2020.

[33] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, 2006.

[34] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-Bounding Protocols: Verification without Time and Location. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2018.

[35] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Post-collusion security and distance bounding. In *Conference on Computer and Communications Security (CCS)*. ACM, 2019.

[36] Catherine A. Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul F. Syverson. Distance Bounding Protocols: Authentication Logic Analysis and Collusion Attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*. Springer, 2007.

[37] Simon Meier. *Advancing automated security protocol verification*. PhD thesis, ETH Zurich, 2013.

[38] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[39] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography*. Springer, 2021.

[40] Jorge Munilla and Alberto Peinado. Distance bounding protocols for RFID enhanced by using void-challenges and analysis in noisy channels. *Wirel. Commun. Mob. Comput.*, 2008.

[41] A. Perrig, R. Canetti, J.D. Tygar, and Dawn Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy. (S&P)*, 2000.

[42] Kasper Bonne Rasmussen and Srdjan Capkun. Realization of RF distance bounding. In *USENIX Security Symposium*. USENIX Association, 2010.

[43] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *IACR Cryptol. ePrint Arch.*, 2004.

[44] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009.

[45] Yu Sasaki, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. New message difference for MD4. In *Fast Software Encryption - International Workshop FSE*. Springer, 2007.

[46] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. PhD thesis, ETH, 2012.

[47] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, 2012.

[48] Marc Stevens. *A Survey of Chosen-Prefix Collision Attacks*, page 182–220. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021.

[49] Marc Stevens, Arjen Lenstra, and Benne De Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007.

[50] Telegram. Mobile protocol: Detailed description. http://web.archive.org/web/20210126200309/https://core.telegram.org/mtproto/description, 2021.

[51] Gene Tsudik. Message authentication with one-way hash functions. *Comput. Commun. Rev.*, 22(5), 1992.

[52] Zooko Wilcox. Lessons from the history of attacks on secure hash functions, 2017. https://electriccoin.co/blog/lessons-from-the-history-of-attacks-on-secure-hash-functions/ (Retrieved Jan 2022).

[53] Tao Xie, Fanbao Liu, and Dengguo Feng. Fast collision attack on MD5. *IACR Cryptol. ePrint Arch.*, 2013.

[54] Jinmin Zhong and Xuejia Lai. Improved preimage attack on one-block MD4. *Journal of Systems and Software*, 2012.

Extracted from	Protocols	Modeling variations	Attack found
Schmidt et al. [47] & Meier [37]	DH2 [14]	1	✗
	TS1 [24]	1	✗
	TS2 [24]	1	✗
	KAS1 [4]	1	✗
	KAS2 [4]	1	✗
	KEA+ [31]	5	✗
	STS-MAC [11]	2	✗
	NAXOS [30]	3	✗
	UM [10]	4	✗
	TESLAv1 [41]	1	✓**
Mauw et al. [34] [35]	Meadows [36]	3	✗*
	MAD [13]	1	✗*
	Kim & Avoine [27]	1	✗*
	Munilla et al. [40]	1	✗*
	CRCS [42]	2	✗*
original	IKEv2 [25]	2	✗
	Sigma [29]	1	✗
	Telegram KE [50]	1	✗
	SSHv2 [33]	1	✗
	Flickr [21]	1	✗*

* = Attack requires breaking one-wayness of the hash
** = Attack requires to instantiate a PRF with a hash function that is not preimage resistant

Table 5: Our initial list of case studies using the equation based threat models from Section 4.1. While the results can indicate missing proof assumptions, this methodology is rather weak and not effective at finding attacks. This motivated our decision to develop a more fine-grained methodology.

A Second preimage equation

Unfortunately, the tools are unable to handle the equation $\mathsf{H}(\mathsf{pi}^2(z)) = \mathsf{H}(z)$: when internally completing this equation they would need to introduce an infinite number of rewrite rules $\mathsf{H}(\mathsf{pi}^2(\dots \mathsf{pi}^2(z)\dots)) \to \mathsf{H}(z)$ stacking any number of applications of pi^2, because they would all be equal to $\mathsf{H}(z)$. This can be avoided using a trick: additionally providing $\mathsf{H}(z)$ to pi^2 in the equation $\mathsf{H}(\mathsf{pi}^2(\mathsf{H}(z), z)) = \mathsf{H}(z)$ effectively avoids the problem mentioned above. As the attacker is able to compute $\mathsf{H}(z)$ from z this second argument can be added without loss of generality.

B Collision relations

In the body of the paper, in Table 2, we gave intuition for the intent of the basic collision relations. Here we present the formal definitions of each of the possible options in Table 6.

C Proofs of TAMARIN's extension

We provide here the details of our extension to TAMARIN as well as describe how to build the required proofs. We rely on the two main TAMARIN thesis [37, 46], from which we reuse the notations and definitions without reintroducing them.

From a high-level point of view, the TAMARIN proof is split into three part:

Capability	Types of Allowed Collisions	Relation $\sim_c$
$\emptyset$	No collision	$\sim_\bot = \{\}^=$
$\exists$	Existential collisions	$\sim_\exists = \{(\mathsf{c}, \mathsf{c}')\}^=$
fstPreImg	Preimage collisions	$\sim_1 = \{(t, \mathsf{pi}^1(H[t])) : t \in T\}^=$
sndPreImg	Second preimage collisions	$\sim_2 = \{(t, \mathsf{pi}^2(t)) : t \in T\}^=$
chsnPrfx	Chosen-Prefix Collisions (CPC)	$\sim_{\mathsf{CP}} = \{(p_1 \Vert \mathsf{cp}_1(p_1, p_2), p_2 \Vert \mathsf{cp}_2(p_1, p_2)) : p_1, p_2 \in T\}^=$
idtclPrfx	Identical-Prefix Collisions (IPC)	$\sim_{\mathsf{IP}} = \{(p \Vert \mathsf{sp}_1(p), p \Vert \mathsf{sp}_2(p)) : p \in T\}^=$
allCol	All collisions	$\sim_\top = (T \times T)^=$
hashExt	Length-extension collisions	$\sim_{\mathsf{LEa}} = \{(x \Vert s,\ H[x] \Vert s) : x, s \in T\}^=$
colExt	Length-extension closure (closure of $\sim$)	$\mathsf{LEc}(\sim) = \{(x \Vert s,\ y \Vert s) : x, y, s \in T, x \sim y\}^=$

Table 6: Basic collision-relations $\sim_c$ depending on the chosen adversarial capabilities. $\sim^=$ denotes the reflexive, symmetric, and transitive closure of the relation $\sim$. For example, by reflexivity, the "no collisions" relation encodes that two hash outputs are the same exactly when their inputs are the same. The "existential collisions" relation encodes that there exists two constants c and c', for which the hash function output collides. Most of the other relations define that there exist collisions that can be computed for very specific, but not all, input patterns. Collision-relations in different dimensions can be combined by taking their union; *e.g.,*sndPreImg,idtclPrfx corresponds to the relations (*e.g.,* $\sim_{\mathsf{IP}} \cup \sim_2$). The only exception is colExt: if this capability is enabled, we first determine the collision relation $\sim$ based on the other capabilities as above, and then compute $\mathsf{LEc}(\sim^=)$ as in the last row.

1. the validity of exploring possible protocol executions using so-called dependency graphs (Lemma 3.10 [46]);

2. a set of constraint solving rules over dependency graphs that are sound and complete (Theorem 3.33 [46] or Theorem 4 [37]);

3. a set of normal form conditions over dependency graphs, that allows to reduce the set of dependency graphs to consider by removing redundant ones (Lemma 3.19/A.12 [46])).

We note that for the soundness and correctness of TAMARIN, only point 1) and 2) are needed. Point 3) is here to help the constraint solving algorithm terminate.

With the proof of TAMARIN in mind, we can describe our extension as the addition of:

- a builtin concatenation symbol $\Vert$ with an associative (A) equation;

- the corresponding attacker construction/deconstruction rules:

$$\frac{\mathrm{K}^{\downarrow d}(t_1 \Vert \ldots \Vert t_n)}{\mathrm{K}^{\downarrow d}(t_1)\ \ldots\ \mathrm{K}^{\downarrow d}(t_n)} \quad \frac{\mathrm{K}^{\uparrow u}(t_1)\ \ldots\ \mathrm{K}^{\uparrow u}(t_n)}{\mathrm{K}^{\uparrow u}(t_1 \Vert \ldots \Vert t_n)}$$

- two normal form conditions on dependency graph.

N7' There is no construction rule for $\Vert$ that has a premise of the form $K^\uparrow(s \Vert t)$ and all conclusion facts of the form $K^\uparrow(s \Vert t)$ are conclusions of a construction rule for $\Vert$.

N8' The conclusion of a deconstruction rule for $\Vert$ is never of the form $K^{\downarrow d}(s \Vert t)$.

We thus have the following proof obligation:

1. The proof of Theorem 4 [37] holds for an equational theory containing an A symbol;

2. The proof of Lemma A.12 [46] holds with the two added rules N7' and N8'.

Lemma A.12 We remark that the (de)construction rules are duplicate from the one for the multiset operator, and the normal form conditions N7' and N8' are duplicate of N7 and N8 from [46]. As such, the proof for N7 and N8 directly applies to N7' and N8'.

Theorem 4 The original proof holds for an equational theory E for which there is a complete and finitary unification, as mentioned in the first sentence of Section 8.2 [37]. We observe that removing the finitary condition does not change anything to the proof if we allow disjunction over constraints to be potentially infinite. The only difference is that rule $S_\approx$ may now create an infinite disjunction when $unify_E^{vars(\Gamma)}(t_1, t_2)$ is infinite. However, this does not change the fact that the completeness and soundness proof for this rule hold, as we do consider all possibilities thanks to the completeness of the unification (albeit there are infinitely many of them).

Thus, as A does have a complete unification algorithm, Theorem 4 in the infinite interpretation does hold when A is integrated inside the equational theory. This cover the soundness and completeness of the theory behind the constraint solving algorithm with a A symbol.

In practice We directly plugged the maude unification algorithm for A inside TAMARIN, which raises a warning when it encounters a unification case for which the set of unifiers is infinite. The consequence of considering that in theory the constraint solving algorithm may create an infinite disjunction has of course consequences in the practical proof search of TAMARIN as we cannot explore this infinite set of cases. As such, whenever in practice TAMARIN makes a unification query for which the unification algorithm return an infinite set, we must abandon the proof. Note that we can still try to find an attack in such a case.

D Proofs of ProVerif's extension

Our extension consists in the introduction of computation function and their applications within axioms. As axioms, lemmas and queries are all correspondence queries, we provide a formal definition of the satisfaction of correspondence queries with computation functions. Note that as axioms are user-given and not verified by ProVerif, it is crucial for users to have in mind this semantics when writing their axioms.

Semantics of correspondence queries with computation function. Generally, a correspondence query with computation function can always be written as a query of the form:

$$F_1 \wedge \ldots \wedge F_n \wedge x \leftarrow g(M) \Rightarrow \bigvee_{i=1}^{m} \psi_i(x)$$

where c is a computation function and for all $i = 1 \ldots m$, $\psi_i(x)$ are a conjunction of facts. Facts can be event facts $\mathtt{event}(ev)$ (holding when the event ev has been raised during the execution of the trace), attacker facts $\mathtt{attacker}(M)$ (holding when the attacker *knows* the term M), equalities and inequalities between terms, and user-defined predicates $p(M_1, \ldots, M_n)$. We refer the reader to [12] for the detailed semantics of ProVerif's processes and satisfaction of facts on a trace. Given an execution trace T and a fact F, we write $T \models F$ when F holds on the trace T. Similarly, if ψ is a conjunction of facts, we write $T \models \psi$ when all facts in ψ hold on T. This notation allows us to define the semantics of correspondence queries as follows.

Definition 1. *Let P be a process. Let ρ be the correspondence query $F_1 \wedge \ldots \wedge F_n \wedge x \leftarrow g(M) \Rightarrow \bigvee_{j=1}^{m} \psi_j(x)$. We say that ρ holds on P when for all substitutions σ, if*

- *$g(M\sigma)$ can be executed and results in $x\sigma$*
- *$dom(\sigma) = \{x\} \cup vars(F_1, \ldots, F_n, M)$*

then for all traces T of P, for all substitutions θ, if $T \models F_1\sigma\theta \wedge \ldots \wedge F_n\sigma\theta$ then there exists a substitution θ' and $j \in \{1, \ldots, m\}$ such that $F_i\sigma\theta = F_i\sigma\theta'$ for $i = 1 \ldots n$, $M\sigma\theta = M\sigma\theta'$, $x\sigma\theta = x\sigma\theta'$ and $T \models \psi_j(x\sigma)\theta'$.

As the above definition applies to general correspondence queries with computation functions, we could, in theory, add queries and lemmas with computation functions to ProVerif. However, in practice, as we do not yet have an algorithm to automatically verify correspondence queries with arbitrary computation functions, we limit the usage of computation functions to axioms, i.e. users are in charge of ensuring that the axioms hold on the input process.

Let us illustrate the definition on the axiom given in Section 4.3:

```
axiom h,h1,h2:t_output;
  eq_hash(h,h1) && h2 ← H(h1)  ⟹  eq_hash(h,h2).
```

where `H` is the computation function defined in Section 4.3. If $\sigma = \{$`h_1` $\mapsto$ `f(Nil,<x,<b,c>>)`$\}$, we have that `H(h_1`σ`)` can be executed and results in `f(f(f(Nil,x),b),c)`. Furthermore, for all substitutions θ, we have indeed that `f(f(f(Nil,x),b),c)`θ produces the same hash as `f(Nil,<x,<b,c>>)`θ in the MD construction. Hence, as expected if `eq_hash(h,h_1)`$\sigma\theta$ holds then `eq_hash(h,h_2)`$\sigma\theta$ holds as well.

Note that in the above example, we only considered one particular σ. As a user, one needs to ensure that the axiom holds for all substitutions σ as indicated in the definition of satisfaction of query.

Application of axioms during the saturation procedure. The internal procedure of ProVerif relies on Horn clauses that are logical statements of the form $H \rightarrow C$, where $H = G_1 \wedge \ldots \wedge G_n$ are the hypotheses of the clause. Intuitively, if $G_1, \ldots, G_n$ hold on a given trace then C also holds on this trace.

ProVerif translates processes into sets of Horn clauses and saturates them before proving the query on the saturated set of Horn clauses. The details of the procedure is out of scope of this paper (see [12]) as we only modified how axioms and lemmas are applied during the saturation algorithm, i.e. how they are applied on a Horn clause.

Consider an axiom $\bigwedge_{i=1}^{n} F_i \wedge x \leftarrow g(M) \Rightarrow \bigvee_{j=1}^{m} \psi_j(x)$. The axiom is applied on a clause $H \rightarrow C$ when there exists a substitution σ such that $F_i\sigma$ is in H for all $i \in \{1, \ldots, n\}$ and $g(M\sigma)$ executes and results in $x\sigma$. The application of the lemma intuitively then produces a set of m clauses $\{H \wedge \psi_j(x)\sigma \rightarrow C\}_{j=1}^{m}$, i.e. each disjunct of the conclusion of the lemma produces a new Horn clause where the instantiated disjunct is added in the hypothesis of the Horn clause.

The soundness of this transformation is straightforward from the semantics of axioms. Take a trace T and a substitution θ such that $T \models H\theta$. Since all $F_i\sigma$ are in H and $g(M\sigma)$ executes and results in $x\sigma$, we deduce from the definition that there exists a substitution θ' and $j \in \{1, \ldots, m\}$ such that $T \models \psi_j(x)\sigma\theta'$ and $F_i\sigma\theta = F_i\sigma\theta'$ for $i = 1 \ldots n$, $M\sigma\theta = M\sigma\theta'$ and $x\sigma\theta = x\sigma\theta'$. Thus, to build $C\theta$, we can use the clause $H \wedge \psi_j(x)\sigma \rightarrow C$ by taking $\theta'' = \theta \cup \theta'$ since $T \models H\theta''$ and $T \models \psi_j(x)\sigma\theta''$.