Oracle Simulation: a Technique for Protocol Composition with Long Term Shared Secrets

Hubert Comon
LSV, CNRS & ENS Paris-Saclay &
Inria & Université Paris-Saclay

Charlie Jacomme
LSV, CNRS & ENS Paris-Saclay &
Inria & Université Paris-Saclay

Guillaume Scerri
Université Versailles Saint-Quentin &
Inria

ABSTRACT

We provide a composition framework together with a variety of composition theorems allowing to split the security proof of an unbounded number of sessions of a compound protocol into simpler goals. While many proof techniques could be used to prove the subgoals, our model is particularly well suited to the *Computationally Complete Symbolic Attacker* (CCSA) model.

We address both sequential and parallel composition, with state passing and long term shared secrets between the protocols. We also provide with tools to reduce multi-session security to single session security, with respect to a stronger attacker. As a consequence, our framework allows, for the first time, to perform proofs in the CCSA model for an unbounded number of sessions.

To this end, we introduce the notion of $\mathcal{O}$-simulation: a simulation by a machine that has access to an oracle $\mathcal{O}$. Carefully managing the access to long term secrets, we can reduce the security of a composed protocol, for instance $P\|Q$, to the security of P (resp. Q), with respect to an attacker simulating Q (resp. P) using an oracle $\mathcal{O}$. As demonstrated by our case studies the oracle is most of the time quite generic and simple.

These results yield simple formal proofs of composed protocols, such as multiple sessions of key exchanges, together with multiple sessions of protocols using the exchanged keys, even when all the parts share long terms secrets (e.g. signing keys). We also provide with a concrete application to the SSH protocol with (a modified) forwarding agent, a complex case of long term shared secrets, which we formally prove secure.

CCS CONCEPTS

• **Security and privacy** → *Formal security models*; *Logic and verification.*

KEYWORDS

formal methods; composition framework; computational model; CCSA model; long term shared secrets

ACM Reference Format:
Hubert Comon, Charlie Jacomme, and Guillaume Scerri. 2020. Oracle Simulation: a Technique for Protocol Composition with Long Term Shared Secrets. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3372297.3417229

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417229

1 INTRODUCTION

This paper is concerned with the security proofs of composed protocols. This topic has been widely studied in the last two decades. For instance, Universal Composability (UC) and simulation based reductions [3, 4, 14–16, 25] and other game-based composition methods [9–11, 28] address this issue. While the former proceed in a more bottom-up manner (from secure components in any environment, construct secure complex protocols), the latter proceed in a more top-down way: from the desired security of a complex protocol, derive sufficient security properties of its components. Such "top-down" proofs design allows more flexibility: the security requirements for a component can be weaker in a given environment than in an arbitrary environment. The counterpart is the lack of "universality": the security of a component is suitable for some environments only.

We follow the "top-down" approach. While we aim at designing a general methodology, our target is the management of formal security proofs in the Computationally Complete Symbolic Attacker (CCSA) model [6]. As a side result of our work, we provide with a way of proving the security of an arbitrary number of sessions (that may depend on the security parameter) in the CCSA model.

When trying to (de-)compose security properties, the main difficulty comes from the fact that different protocols may share some secrets. This is typically the case for multiple sessions of the same protocol, or for key exchange protocols, which result in establishing a shared secret that will be later used in another protocol. Protocols may also share long term secrets, for instance the same signing key may be used for various authentication purposes. Another example is the SSH protocol with the agent forwarding feature [32], which we will consider later. The forwarding feature allows to obtain, through previously established secure SSH connections, signatures of fresh material required to establish new connections. It raises a difficulty, as signatures with a long term secret key are sent over a channel established using the same long term secret key.

As far as we know, the existing composition results that follow the "top-down" approach cannot be used in situations where there is both a "state passing", as in key exchange protocols, and shared long term secrets. For instance, in the nice framework of [10], the same public key cannot be used by several protocols, a key point for reducing security of multiple sessions to security of one session.

When decomposing the security of a composed protocol into the security of its components, we would like to break a complex proof into simpler proofs, while staying in the same proof framework. This is also a difficulty since the attacker on a protocol component

might use the other components: we need a proof with respect to a stronger attacker. In [10], such a strong attacker can be simulated by a standard one, because there is no shared long term secret.

1.1 Our contributions

We provide with a composition framework that reduces the security of a compound protocols to the security of its components. We allow both state passing and shared long term secrets. We stay in the same proof framework of the CCSA model.

The starting idea is simple: if we wish to prove the security of a composed protocol $P||Q$, it is sufficient to prove the security of P against an attacker that may simulate Q, maybe with the help of an oracle. If $\overline{n}$ are the secrets shared by P and Q, this simulation has to be independent of the distribution of $\overline{n}$. This is actually an idea that is similar to the key-independence of [11].

Therefore, we first introduce the notion of $\mathcal{O}$-simulation, in which an oracle $\mathcal{O}$ holds the shared secrets: if Q is $\mathcal{O}$-simulatable and P is secure against an attacker that has access to $\mathcal{O}$, then $P||Q$ is secure. Intuitively, $\mathcal{O}$ defines an interface through which the secrets can be used (e.g. obtaining signatures of only well tagged messages). $\mathcal{O}$ simulatable protocols conform to this interface.

We extend this basic block to arbitrary parallel and sequential compositions, as well as replication of an unbounded number of copies of the same protocol. In the latter case, the security of a single copy of P against an attacker that has access to an oracle allowing to simulate the other copies, requires to distinguish the various copies of a same protocol. In the universal composability framework, this kind of properties is ensured using explicit session identifiers. We rather follow a line, similar to [26], in which the session identifiers are implicit.

Our main composition Theorems are generic: the classical game based setting can be used to prove the subgoals. They are also specially well-suited for the CCSA model, which allows to complete computational proofs of real life protocols [5, 18, 30], while only relying on first order logic and cryptographic axioms. Many such axioms can easily be generalized so as to be sound with respect to an attacker that has access to oracles (we will see examples later).

A proof using such axioms is valid for an attacker who has access to an environment, while abstracting all the details of the environment and its interactions with the attacker. Moreover, as our reductions from one session to multiple sessions are uniform, we may now complete proofs in the CCSA model for a number of session that is parameterized by the security parameter. This was a limitation (and left as an open issue) in all previous CCSA papers.

We illustrate our composition results showing how to split the security of any (multi-session with shared long term secret) composed key exchange into smaller proofs. We then complete the formal proof of security of a Diffie-Hellman key exchange (ISO 9798-3 [1]) for any number of sessions in parallel.

We generalize the application to key exchanges performing key confirmations, i.e. using the derived key in the key exchange (as in TLS). The generalization is simple, which is a clue of the usability of our framework.

To illustrate the usability of our framework, we use all our results to prove the security of the SSH [32] protocol with a modified agent forwarding, a complex example of key exchange, with both key confirmation and long term shared secrets. The modification, which consists in the addition of a tag to specify if the signature was performed remotely, is necessary for the protocol to satisfy some natural security properties related to the agent forwarding.

1.2 Related Works

We introduce the composition problem through a process algebra: protocols are either building blocks (defined,e.g, with a transition system) or composed using parallel and sequential composition, and replication. This prevents from committing to any particular programming language, while keeping a clean operational semantics. This approach is also advocated in [10], which follows a similar approach. Other works on composition (e.g., [3, 28]) rely on specific execution models.

Our starting idea, to prove a component w.r.t. a stronger attacker that has access to the context, is not new. This is the basis of many works, including [9–12]. The main difference, that we wish to emphasize, is that these works do not support long term shared secrets, used in different components. Notably, the oracles of [10] are only used to decompose protocols with state passing. Our notion of simulatability allows sharing long term secret by granting the attacker access to oracles that depend on the secrets (for instance, signing oracles). It also allows a symmetric treatment for proofs of a protocol and proofs of its context.

For several specific problems, typically key exchanges, there are composition results allowing to prove independently the key exchange protocol and the protocol that uses the exchanged key [9, 11, 12, 23, 26]. In such examples, the difficulty also comes from the shared secret, especially when there is a key confirmation step. In that case, the derived key is used for an integrity check, which is part of the key exchange. Then the property of the key exchange: "the key is indistinguishable from a random" does not hold after the key confirmation and thus cannot be used in the security proof of the protocol that uses this exchanged key. In [11], the authors define the notion of key independent reduction, where, if an attacker can break a protocol for some key distribution, he can break the primitive for the same distribution of the key. This is related to our notion of simulatability, as interactions with shared secrets are captured by an oracle for fixed values of the key, and thus attacks on the protocol for a fixed distribution are naturally translated into attacks against the primitive for the same distribution. Key exchanges with key confirmation are therefore a simple application of our composition results. Along the same line, [23] extends [12] to multi staged key exchanges, where multiple keys might be derived during the protocol. While we do not directly tackle this in our paper, our framework could be used for this case.

The authors of [9] also provide results allowing for the study of key renewal protocols (which we consider in the long version [17]), and has the advantage to be inside a mechanized framework, while we only cast our results inside a mechanizable framework. It does not however consider key confirmations.

The UC framework initiated by [15] and continued in [4, 14, 25] is a popular way of tackling composition. As explained above, this follows a "bottom-up" approach, in which protocols must be secure in any context, which often yield very strong security properties, some of which are not met in real life protocols. Moreover, to handle

multiple sessions of a protocol using a shared secret, joint-state theorems are required. This requires a tagging mechanism with a distinct session identifier (sid) for each session. Relaxing this condition, the use of implicit session identifiers was established in [27] for the UC framework, ideas continued in [26] for Diffie-Hellman key exchanges, where they notably provide a proof of the ISO 9798-3 [1] protocol.

We do not consider a composition that is universal: it depends on the context. This allows us to relax the security properties regarding the protocol, and thus prove the compositional security of some protocols that cannot be proved secure in the UC sense. We also rely on implicit sids to prove the security of multiple sessions. Some limitations of the UC framework are discussed in [12, Appendix A].

In [3], the authors also address the flexibility of UC (or reactive simulatability) showing how to circumvent some of its limitations. The so-called "predicates" are used to restrict the order and contents of messages from environment and define a conditional composability. Assuming a joint-state conditional composability theorem, secret sharing between the environment and the protocol might be handled by restricting the accepted messages to the expected use of the shared secrets. However, the framework does not cover how to prove the required properties of (an instance of) the environment.

Protocol Composition Logic is a formal framework [22] designed for proving, in a "Dolev-Yao model", the security of protocols in a compositional way. Its computational semantics is very far from the usual game-based semantics, and thus the guarantees it provides [21] are unclear. Some limitations of PCL are detailed in [20].

The compositional security of SSH, in the sense of [12], has been studied in [31]. They do not consider however the agent forwarding feature. It introduces important difficulties since the key exchange is composed with a second key exchange that uses both the first derived key and the same long term secrets. SSH has also been studied, without agent forwarding, in [13], where the implementation is derived from a secure modelling in CryptoVerif [8].

Summing up, our work is strongly linked to previous composition results and captures analogues of the following notions in our formalism: implicit disjointness of local session identifiers [27], single session games [12], key-independent reductions [11] and the classical proof technique based on pushing part of a protocol inside an attacker, as recently formalized in [10]. We build on all these works and additionally allow sharing long term secrets, thanks to a new notion of O-simulatability. This fits with the CCSA model: the formal proofs of composed protocols are broken into formal proofs of components. All these features are illustrated by a proof of SSH with (a modified) agent forwarding.

2 PROTOCOLS AND INDISTINGUISHABILITY

We first recall some features of the CCSA model. Although this model is not used until the case studies, it may be useful for an easier understanding of the protocol semantics.

2.1 Syntax and semantics of terms

To enable composition with long term shared secrets, we must be able to specify precisely the shared randomness between protocols. We use symbols from an alphabet of *names*, to represent the random samplings. The same symbol used twice represents the same (shared) randomness. Those names can be seen as pointers to a specific randomness, where all the randomness has been sampled upfront at the beginning of the protocol. This idea stems from the CCSA model [6], from which we re-use exactly the same term semantics. This is one of the reason why our results, while applicable in a broader context, fit naturally in the CCSA model. Let us recall the syntax and semantics of terms drawn from the CCSA model.

Syntax. We use terms built over explicit names to denote messages computed by the protocol. The terms are defined with the following syntax:

t	::=	n	names
	|	$n_{\vec{i}}$	indexed names
	|	x	variable
	|	$f(t_1, \ldots, t_n)$	operation of arity n

A key addition to the CCSA model is that some names can be indexed by sequences of index variables. This is necessary so that we may later on consider the replication of protocols. When a replicated protocol depends on a name n_i, the first copy (session) of the protocol uses n_1, the second n_2, ... Names without index models randomness shared by all sessions of the protocol. Variables are used to model the attacker inputs, and function symbols allows to model the cryptographic computations.

Semantics. Terms are interpreted as bitstrings. As in the computational model, the interpretation depends on some security parameter η. As we assume that all the randomness is sampled at the beginning, the interpretation depends on an infinitely long random tape ρ_s. We then leverage the notion of a *cryptographic library*[1], that provides an interpretation for all names and function symbols. A cryptographic library $\mathcal{M}_f$ provides for each name n a Probabilistic Polynomial Time Turing Machine (PPTM for short) $\mathcal{A}_n$, that is given access to the random tape ρ_s. As an additional input, all machines will always be given the security parameter in unary. Each $\mathcal{A}_n$ extracts a bit-string of length η from the random tape. Different names extract non-overlapping parts of the random tape. In the interpretation, we give to all the PPTM the same random tape ρ_s, so each name is always interpreted with the same value in any term (and thus any protocol), and all names are interpreted independently.

$\mathcal{M}_f$ also provides for each function symbol f (encryption, signature,...) a PPTM $\mathcal{A}_f$, that must be deterministic. To model randomized cryptographic primitives, additional randomness must be given to the function symbol as extra names (cf. example 2.1).

Given $\mathcal{M}_f$, the semantic mapping $[\![\cdot]\!]^{\eta,\sigma}_{\rho_s}$ evaluates its argument, a formal term, given an assignment σ of its variables to bit-strings and a random tape ρ_s. For instance, if n is a name, $[\![n]\!]^{\eta}_{\rho_s} = \mathcal{A}_n(1^\eta, \rho_s)$ (extracts a bit-string of length η from the random tape ρ_s) and $[\![\mathsf{sign}(x,k)]\!]^{\eta,\{x\mapsto m\}}_{\rho_s} = \mathcal{A}_{\mathsf{sign}}(m, \mathcal{A}_k(1^\eta, \rho_s))$.

2.2 Syntax of the protocols

The summary of the protocol syntax is given in fig. 1. An elementary protocol models a thread running on a specific computer. let denotes variable binding inside a thread, $\mathtt{in}(c, x)$ (resp. $(\mathtt{out}(c, m))$ denotes an input (resp. an output) of the thread over the channel c,

[1] This corresponds in the CCSA model to the notion of functional model.

elementary protocols:

P_{el}	::=	let $x = t$ in P_{el}	variable binding
	|	$\mathtt{in}(c, x).P_{el}$	input
	|	$\mathtt{out}(c, m).P_{el}$	output
	|	if $s = t$ then P_{el} else P_{el}	conditional
	|	$\mathbf{0}$	success
	|	$\bot$	failure

protocols:

P, P'	::=	P_{el}	
	|	$P_{el}; P$	sequential composition
	|	$P \| P'$	parallel composition
	|	$\|^{i \le N} P$	bounded replication
	|	$\|^{i} P$	unbounded replication

Figure 1: The protocol algebra

where all channels are taken out of a set C. For simplicity, channel identifiers are constants or indexed constants. In particular, they are known to the attacker. The if then else constructs denotes conditionals, $\mathbf{0}$ is a successfully terminated thread and $\bot$ is an aborted thread.

For protocols, our goal is to state and prove general composition results: we first consider sequential composition (the ; operator), where $\mathbf{0}; P$ reduces to P, while $\bot; P$ reduces to $\bot$. In most cases, we will omit $\mathbf{0}$. We also consider parallel composition (the $\|$ operator), a fixed number N of copies running concurrently $\|^{i \le N}$, as well as an arbitrary number of copies running concurrently $\|^{i}$. For instance, we can express a (two-parties) key-exchange consisting of an initiator I and a responder R with $I \| R$, the key exchange followed by a protocol using the exchanged key $(I; P^I) \| (R; P^R)$, as well as any number of copies of the resulting protocol running in parallel: $\|^{i}((I; P^I) \| (R; P^R))$. We can also consider an arbitrary iteration of a protocol, "$;^{i}$", which could be used for expressing, for instance, key renewal. For simplicity, this latter construction is not presented in the current paper (see [17] for details).

We allow terms inside a protocol to depend on some free variables and, in this case, we denote $P(x_1, \ldots, x_n)$ a protocol, which depends on free variables $x_1, \ldots, x_n$. $P(t_1, \ldots, t_n)$ denotes the protocol obtained when instantiating each x_i by the term t_i.

We denote $\mathcal{N}(P)$ (resp $C(P)$) the set of names (resp. channel names) of P.

Example 2.1. Given a randomized encryption function enc, we let $P(c, x_1, x_2)$ be the protocol $\mathtt{in}(c, x).\mathtt{out}(c, \mathsf{enc}(x, x_1, x_2))$. Given names sk, r representing respectively a secret key and a random seed, $E_N := \|^{i \le N} P(c_i, r_i, sk)$ is then the protocol allowing the attacker to obtain cyphertexts for an unknown secret key sk. Unfolding the definitions, we get:

$$E_N := P(c_1, r_1, sk) \| \ldots \| P(c_n, r_n, sk)$$

The generalization giving access to encryption for five secret keys is expressed with $\|^{i} \|^{j \le 5} P(c_{j,i}, r_{j,i}, sk_j)$.

2.3 Semantics of the protocols

We give here some essential features of the formal execution model, which we need to formalize our composition results.

$$\frac{\phi, (P, \sigma) \underset{\mathcal{A}}{\longrightarrow} \phi', (P', \sigma')}{\phi, (P; Q, \sigma) \underset{\mathcal{A}}{\longrightarrow} \phi', (P'; Q, \sigma')} \qquad \frac{}{\phi, (\mathbf{0}; Q, \sigma) \underset{\mathcal{A}}{\longrightarrow} \phi, (Q, \sigma)}$$

$$\frac{\phi, (P, \sigma) \underset{\mathcal{A}}{\longrightarrow} \phi', (P', \sigma')}{\phi, (P, \sigma) \| E \underset{\mathcal{A}}{\longrightarrow} \phi', (P', \sigma') \| E}$$

Figure 2: Operational Semantics (excerpt)

A (global) state of a protocol consists in a *frame*, which is a sequence of bit-strings modelling the current attacker knowledge, and a finite multiset of pairs (P, σ), where P is a protocol and σ is a local binding of variables. Intuitively, each of the components of the multiset is the current state of a running thread. We write such global states $\phi, (P_1, \sigma_1) \| \cdots \| (P_n, \sigma_n)$.

The transition relation between global states is parameterized by an attacker $\mathcal{A}$ who interacts with the protocol, modelled as a PPTM with its dedicated random tape ρ_r. The attacker chooses which of the threads is going to move and computes, given ϕ, the input to that thread. In the following, the configuration of the protocol and the security parameter are (also) always given to the attacker, which we do not make explicit for simplicity.

We give some of the rules describing the Structural Operational Semantics in fig. 2. The full semantics can be found in [17]. The transition relation $\underset{\mathcal{A}}{\longrightarrow}$ between configurations depends on the attacker $\mathcal{A}$, the security parameter η and the random samplings ρ_s (to interpret terms) and ρ_r (the randomness of the attacker). In $P; Q$, P has to be executed first. When it is completed (state $\mathbf{0}$), then the process can move to Q, inheriting the variable bindings from P. If P is not waiting for an input from the environment, it can move independently from any of the other parallel processes.

The semantics of inputs (not detailed for simplicity) reflects the interactions with the attacker. $\mathcal{A}$ computes the input to the protocol, given a frame ϕ and its own random tape ρ_r. Therefore transitions depend not only on the attacker machines, but also[2] on the name samplings ρ_s (secret coins) and ρ_r (attacker's coins).

Example 2.2. Continuing example 2.1, the initial configuration corresponding to E_2 is $\emptyset, (P(c_1, r_1, sk), \emptyset) \| (P(c_2, r_2, sk), \emptyset)$, where the attacker knowledge is empty and no local variables are bound. We provide in fig. 3 one of the possible reductions, for some attacker $\mathcal{A}$ that first sends a message over channel c_1 and then c_2.

We assume *action determinism* of the protocols [19]: given an input message on a given channel, at most one of the threads may move to a non-abort state. This means that each thread checks first that it is the intended recipient of the message. This also means that each output has to be triggered by an input signal: none of the P_i starts with an output action. We remark that in practice, protocols are action determinate.

For replicated protocols $\|^{i \le N} P$ or $\|^{i} P$, the names in P that are indexed by the variable i are renamed as follows: $\|^{i \le N} P$ is the

[2]They actually also depend on the oracle's coins, when $\mathcal{A}$ is interacting with an external oracle, which we explain later.

$$\emptyset, (P(c_1, r_1, sk), \emptyset) \| (P(c_2, r_2, sk), \emptyset)$$
$$\xrightarrow[\mathcal{A}]{} \emptyset, (\mathsf{out}(c_1, \mathsf{enc}(x, r_1, sk), \{x \mapsto m\}) \| (P(c_2 r_2, sk), \emptyset)$$
$m = \mathcal{A}(\emptyset, \rho_r)$ is the first input message computed by the attacker
$$\xrightarrow[\mathcal{A}]{} \phi, (P(c_2, r_2, sk), \emptyset)$$
$\phi = [\![\mathsf{enc}(x, r_1, sk)]\!]_{\rho_s}^{\eta, \{x \mapsto m\}}$ is the interpretation of the output received by the attacker
$$\xrightarrow[\mathcal{A}]{} \phi, (\mathsf{out}(c_2, \mathsf{enc}(x, r_1, sk), \{x \mapsto m_2\})$$
$m_2 = \mathcal{A}(\phi, \rho_r)$ is the second input message computed by the attacker
$$\xrightarrow[\mathcal{A}]{} (\phi, [\![\mathsf{enc}(x, r_2, sk)]\!]_{\rho_s}^{\eta, \{x \mapsto m_2\}}), \mathbf{0}$$

Figure 3: Reduction example

protocol $P\{i \mapsto 1\} \| \ldots \| P\{i \mapsto N\}$ and

$$\phi, (\|^i P, \sigma) \| E \xrightarrow[\mathcal{A}]{} \phi, (\|^{i \leq \mathcal{A}(\rho_r, \phi)} P, \sigma) \| E.$$

In other words, the attacker chooses how many copies of P will be considered, which may depend, in particular, on the security parameter. $\mathcal{A}(\rho_r, \phi)$ must be a natural number in unary.

2.4 Stateless oracle machines

For reasons that have been explained in the introduction, we wish to extend the semantics of protocols and their indistinguishability to attackers that have access to an oracle. At this stage, we need stateless oracles in order to be compositional. Let us explain this. Assume we wish to prove a property of R in the context $P \| Q \| R$. The idea would be to prove R, interacting with an attacker that simulates $P \| Q$. This attacker is itself a composition of an attacker that simulates P and an attacker that simulates Q. The protocols P, Q, R share primitives and secrets, hence the simulation of P, Q requires access to an oracle that holds the secrets. If such an oracle was stateful, we could not always build a simulator for $P \| Q$ from simulators of P, Q respectively, since oracle replies while simulating Q could depend on oracle queries made while simulating P, for instance.

The oracles depend on a security parameter η (that will not always be explicit), (secret) random values and also draw additional coins: as a typical example, a (symmetric key) encryption oracle will depend on the key k and use a random number r to compute $\mathsf{enc}(m, r, k)$ from its query m. Therefore, the oracles can be seen as deterministic functions that take two random tapes as inputs: ρ_s for the secret values and ρ_O for the oracle coins.

Formally, oracles take as input tuples $(\overline{m}, r, s)$ where $\overline{m}$ is a finite sequence of bit-strings, r is a handle for a random value and s is a handle for a secret value. r and s are respectively used to extract the appropriate parts of ρ_O, ρ_s respectively, in a deterministic way: the randomness extracted from ρ_O is uniquely determined by $\overline{m}, r, s$ and the extractions for different values do not overlap. See appendix A for more details.

In what follows, we only consider oracles that are consistent with a given cryptographic library $\mathcal{M}_f$. Such oracles only access ρ_s through some specific names. This set of names is called the *support* of the oracle.

Example 2.3. We present the oracle $O_k^{\mathsf{enc},\mathsf{dec}}$ that provides both an encryption and a decryption oracle for the key k. On input $< m, r, s >$, the oracle:

- returns $\perp$ if $s \neq 1$ (s can be used to select a specific key, and the oracle only provides access to a single one);
- samples a bit-string r of length η from ρ_O, from a position determined by the input;
- if $m = (\text{“}dec\text{”}, n)$, returns $[\![\mathrm{dec}(x, y, k)]\!]_{\rho_s}^{\{x \mapsto n, y \mapsto r\}}$
- if $m = (\text{“}enc\text{”}, n)$, returns $[\![\mathrm{enc}(x, y, k)]\!]_{\rho_s}^{\{x \mapsto n, y \mapsto r\}}$

The support of the oracle is $\{k\}$, the only name used inside it.

An oracle machine (PPTOM) is a PPTM, equipped with an additional tape, on which the queries to the oracle are written and from which the oracle replies are read. We often write explicitly the machine inputs, as in $\mathcal{A}^{O(\rho_s, \rho_O)}(\omega, \rho_r)$, where ω is the input data of $\mathcal{A}$, ρ_r is its random tape and ρ_s, ρ_O are the random tapes accessible to the oracle. These definitions extend to multiple oracles $\langle O_1, \ldots, O_n \rangle$, prefixing the query with an index in $\{1, \ldots, n\}$.

Note that once the oracle's random tape is fixed, we ensure that all our oracles are deterministic. While not strictly necessary, this ensures that the various parts of the adversary do not need to explicitly share states, as they can always recompute the answer to oracle calls. This greatly simplifies proofs.

2.5 Computational indistinguishability

To define the classical notion of indistinguishability, we describe how protocols may be seen as oracles, that an attacker can interact with. Given a protocol P and a cryptographic library $\mathcal{M}_f$, the oracle O_P is an extension of the previous oracles: it takes as an additional input an history tape that records the previous queries. Given a query m with history h (now the components r, s are useless), the oracle replies what would be the output of P, given the successive inputs h, m. It also appends the query m to the history tape.

The machines that interact with O_P are also equipped with the history tape that is read-only: the history can only be modified by the oracle. Since P may use secret data, the oracle may access a secret tape ρ_s; this will be explicit.

An oracle may implement multiple parallel protocols: the oracle $O_{\langle P_1, \ldots, P_n \rangle}$ first checks which P_i is queried (there is at most one such i, by action determinism) and then replies as O_{P_i}.

Finally, we may consider oracles that combine protocols oracles and stateless oracles. $\mathcal{A}^{\langle O_1, \ldots, O_m \rangle, \langle O_{P_1}, \ldots, O_{P_n} \rangle}$ is also written $\mathcal{A}^{O_1, \ldots, O_m, O_{P_1}, \ldots, O_{P_n}}$.

Definition 2.4. Given a cryptographic library $\mathcal{M}^f$, protocols P, Q and a stateless oracle O , P is *O-indistinguishable from Q*, which we write $P \cong_O Q$, when, for every PPTOM $\mathcal{A}$,

$$|\mathbb{P}_{\rho_s, \rho_r, \rho_O}\{\mathcal{A}^{O(\rho_s, \rho_O), O_P(\rho_s)}(1^\eta, \rho_r) = 1\} - \mathbb{P}_{\rho_s, \rho_r, \rho_O}\{\mathcal{A}^{O(\rho_s, \rho_O), O_Q(\rho_s)}(1^\eta, \rho_r) = 1\}|$$

is negligible in η.

We will later see several examples of O-indistinguishability. Remark that the oracle only increases the capabilities of the attacker, and thus for any P, Q and O, $P \cong_O Q$ implies $P \cong Q$.

3 SIMULATABILITY

As an example, consider a protocol P that uses a shared encryption key k, and may only accept encrypted messages whose plaintext satisfy a condition T_P (e.g. tagged messages). Intuitively, this protocol may be composed with any context that uses the shared key to encrypt plaintexts that cannot be confused with messages intended for P. We capture this by proving this protocol while giving the adversary access to an oracle O, where O lets the adversary encrypt any message as long as it does not satisfy T_P. Then if P is "secure", P composed with any context that can be simulated using only O to access k is "secure". We define now this notion of simulatability.

3.1 Protocol simulation

The goal in the rest of the paper is to use this notion of simulatability to obtain composability results. Suppose one wants to prove $P\|Q \cong P\|R$, knowing that $Q \cong_O R$ and P is O simulatable. The way to obtain a distinguisher for $Q \cong_O R$ from one on $P\|Q \cong P\|R$ is to "push" the (simulated version) of P within the distinguisher. A protocol P is then simulatable if there exists a simulator $\mathcal{A}^O$ that can be "pushed " inside any distinguisher $\mathcal{D}$. We formalize this construction below, where a protocol is simulatable if and only if any distinguisher $\mathcal{D}$ behaves in the same way if the protocol oracle O_P is replaced by its simulator $\mathcal{A}^O$. We define formally $\mathcal{D}[\mathcal{A}^O]^O$ the replacement of O_P inside $\mathcal{D}^{O,O_P}$.

Definition 3.1. Given an oracle O, a cryptographic library $\mathcal{M}_f$, a protocol P, PPTOMs $\mathcal{D}^{O,O_P}(\rho_{r_\mathcal{D}}, 1^\eta)$ and $\mathcal{A}^O(\cdots, 1^\eta)$, we define $\mathcal{D}[\mathcal{A}^O]^O(\rho_r, 1^\eta)$ as the PPTOM that:

(1) Splits its random tape ρ_r into ρ_{r_1}, ρ_{r_2}
(2) Simulates $\mathcal{D}^{O,O_P}(\rho_{r_2}, 1^\eta)$ by replacing every call to O_P with a computation of $\mathcal{A}^O$: each time $\mathcal{D}$ enters a state corresponding to a call to O_P, $\mathcal{D}[\mathcal{A}^O]$ appends the query m to a history θ (initially empty), executes the subroutine $\mathcal{A}^{O(\rho_s,\rho_O)}(\rho_{r_1}, \theta, 1^\eta)$ and behaves as if the result of the subroutine was the oracle reply.
(3) Prefixes each random handle of an oracle call of $\mathcal{D}$ with 0 and random handle of an oracle call of $\mathcal{A}$ with 1.
(4) Outputs the final result of $\mathcal{D}$.

$\mathcal{D}[\mathcal{A}^O]^O$ must simulates $\mathcal{A}^O$ and $\mathcal{D}$ so that they do not share randomness. To this end, $\mathcal{D}[\mathcal{A}^O]^O$ first splits its random tape ρ_r into ρ_{r_1} (playing the role of ρ_O) and ρ_{r_2} (playing the role of $\rho_\mathcal{D}$). The oracle queries are prefixed by distinct handles for the same reason. $\mathcal{D}^{O,O_P}$ has access to the shared secrets via both O and O_P, while $\mathcal{D}[\mathcal{A}^O]^O$ only has access to them through the oracle O . Remark that if $\mathcal{A}^O$ and $\mathcal{D}^{O,O_P}$ has a run-time polynomially bounded, so does $\mathcal{D}[\mathcal{A}^O]^O$.

To define the central notion of O-simulatability, the distribution produced by any distinguisher interacting with the simulator must be the same as the distribution produced when he is interacting with the protocol. However, as we are considering a set of shared secrets $\overline{n}$ that might be used by other protocols, we need to ensure this equality of distributions for any fixed concrete value $\overline{v}$ of the shared secrets. Then, even if given access to other protocols using the shared secrets, no adversary may distinguish the protocol from its simulated version.

Definition 3.2. Given a sequence of names $\overline{n}$, an oracle O with support $\overline{n}$, a cryptographic library $\mathcal{M}_f$, a protocol P, then, $\nu\overline{n}.P$ is *O-simulatable* if and only if there exists a PPTOM $\mathcal{A}_P^O$ such that for every PPTOM $\mathcal{D}^{O,O_P}$, for every η, every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}, c \in \{0,1\}^\star$,

$$\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}_P^O]^O(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

Note that our definition of simulatability is a very strong one as it requires a perfect equality of distributions, as opposed to computational indistinguishability. This is intuitively what we want: O-simulation expresses that P only uses the shared secrets as O does. This notion is not intended to capture any security property.

While this definition intuitively captures the proof technique used to allow composition, it does not provide insight about how to prove the simulatability. Another equivalent definition states that a protocol is simulatable if there exists a simulator that can produce exactly the same distribution of messages as the protocol interacting with any attacker. The formal definition corresponding to this intuition, and the proof that is is equivalent to our main definition are provided in appendix B.

Example 3.3. We fix first $\mathcal{M}_f$ (in an arbitrary way). We consider the following handshake protocol, in which n, r, k, r' are names:

$$\begin{array}{lll} & A = & \mathsf{in}\,(c_A, x_0).\mathsf{out}(c_B, \mathsf{enc}(n, r, k)).\ \mathsf{in}\,(c_A, x). \\ & & \quad \mathsf{if}\ \mathsf{dec}(x, k) = \langle n, 1\rangle\ \mathsf{then}\ \mathsf{out}(c_B, \mathsf{ok}) \\ \| & B = & \mathsf{in}\,(c_B, y).\mathsf{out}(c_A, \mathsf{enc}(\langle \mathsf{dec}(y, k), 1\rangle, r', k)) \end{array}$$

We consider the oracle $O_k^{\mathsf{enc},\mathsf{dec}}$ from example 2.3. We can easily prove that $\nu k.A$ is $O_k^{\mathsf{enc},\mathsf{dec}}$-simulatable, as the attacker can sample an arbitrary n', use the oracle to compute $\mathsf{enc}(n', r_1, k)$ (which as the same distribution as $\mathsf{enc}(n', r, k)$ for any fixed value of k) with the request $<$ ("enc", n), 1, 1 $>$, and $\mathsf{dec}(x, k)$ with the request $<$ ("dec", x), 1, 1 $>$.

Intuitively, the shared secret k is only used inside A in ways that are directly simulatable with the oracle, and A is thus $O_k^{\mathsf{enc},\mathsf{dec}}$-simulatable.

Thanks to the definition of appendix B, proving simulatability is in practice a syntactic verification. For instance, $\nu k.P$ is $O_k^{\mathsf{enc},\mathsf{dec}}$-simulatable (example 2.3) if k only appears in P in key position of an encryption or a decryption, and all encryptions use fresh randoms. With this definition, simulatability is stable under composition operators. This allows to reduce the simulation of large processes to the simulation of simpler processes.

Theorem 3.4. *Given an oracle O, protocols P, Q, and $\overline{n} = \mathcal{N}(P) \cap \mathcal{N}(Q)$, if $\nu\overline{n}.P$ and $\nu\overline{n}.Q$ are O-simulatable, then $\nu\overline{n}.P\|Q$ and $\nu\overline{n}.P;Q$ are O-simulatable.*

3.2 Generic oracles for tagged protocols

In order for our definition of simulatability to be useful, the design of oracles is a key point. They need to be:

(1) generic/simple, yet powerful enough so that protocols can be easily shown to be simulatable,

(2) restrictive enough so that proving protocols in the presence of oracles is doable.

We provide here with examples of such oracles, namely generic tagged oracles for signature, that will be parameterized by arbitrary functions, together with security properties that are still true in the presence of tagged oracles. We see tagging as a boolean function T computable in polynomial time over the interpretation of messages. For instance, if the messages of protocol P are all prefixed with the identifier id_P, T is expressed as $T(m) := \exists x.m = \langle id_P, x\rangle$. In a real life protocol, id_P could for instance contain the name and version of the protocol.

Intuitively tagged oracles produce the signature of any properly tagged message and allow to simulate P.

With these oracles, an immediate consequence of the composition Theorems found in section 4 is the classical result that if two protocols tag their messages differently, they can be safely composed [2]. Note that as our tag checking function is an arbitrary boolean function: tagging can be implicit, as illustrated in our applications in section 5.

As an example, we provide two oracles, one for encryption and one for signing, that allow to simulate any protocol that only produces messages that are well tagged for T.

Definition 3.5. Given a name sk and a tagging function T, we define:

$$O^{\mathsf{sign}}_{T,sk}(m) := \text{if } T(m) \text{ then output}(\mathsf{sign}(m, sk))$$
$$O^{\mathsf{dec}}_{T,sk}(m) := \text{if } T(\mathsf{dec}(m, sk)) \text{ then output}(\mathsf{dec}(m, sk))$$

Any well-tagged protocol according to T, i.e. a protocol that only decrypts or signs well tagged messages, will be simulatable using the previous oracles. Hence we meet the goal 1 stated at the beginning of this section, as this can be checked syntactically on a protocol. We provide, as an example, the conditions for a tagged signature.

Example 3.6. Any protocol P whose signatures are all of the form if $T(t)$ then $\mathsf{sign}(t, sk)$ for some term t (that does not use sk) is immediately $\nu sk.P$ $O^{\mathsf{sign}}_{T,sk}$-simulatable. Indeed, informally, all internal values of the protocol except sk can be picked by the simulator from its own randomness, while all terms using sk can be obtained by calls to the tagged signing oracle, as all signed terms in P are correctly tagged. Let us emphasize that the simulation holds for any specific value of sk, as the distribution of outputs is the same, whether it is the simulator that draws the internal names of P, except sk, or P itself.

As we need to perform cryptographic proofs in the presence of oracles, it is useful to define security properties that cannot be broken by attackers with access to these oracles (without having to consider the specific calls made to these oracles). The games defining these properties slightly differ from the classical security games. Consider the example of signatures and the usual EUF-CMA game. If the attacker is, in addition, equipped with an oracle O that signs tagged messages, he immediately wins the EUF-CMA game, "forging" a signature by a simple call to O. We thus define a tagged unforgeability game (EUF-CMA$_{T,sk}$), derived from the EUF-CMA game [24], where the adversary wins the game only if he is able to produce the signature of a message that is not tagged.

Definition 3.7. A signature scheme (Sign, Vrfy) is EUF-CMA$_{T,sk}$ secure for oracle O and interpretation of keys $\mathcal{A}_{sk}$ if, for any PPTOM $\mathcal{A}$, the game described in fig. 4 returns **true** with probability (over ρ_r, ρ_s, ρ_O) negligible in η.

Game EUF-CMA$^{\mathcal{A}}_{T,sk}(\eta, \rho_r, \rho_s, \rho_O)$:
List ← []
(pk, sk) ← ($[\![pk]\!]_{\rho_s}$, $[\![sk]\!]_{\rho_s}$)
(m, σ) ← $\mathcal{A}^{O(\rho_s,\rho_O),\mathsf{Sign}}$(pk, η, ρ_r)
Return $\neg T(\mathsf{m}) \wedge \mathsf{Vrfy}(\mathsf{pk}, \mathsf{m}, \sigma) \wedge \mathsf{m} \notin \mathsf{List}$

Oracle Sign(m):
List ← (m : List)
σ ← Sign(sk, m)
Return σ

Figure 4: Game for Tagged Unforgeability (EUF-CMA$_{T,sk}$)

The main goal of the previous definition is to allow us to prove protocols in the presence of oracles (hence composed with simulated ones), reaching the goal 2 stated at the beginning of the section. More precisely, one can, for instance, simply design a classical game based proof, reducing the security of the protocol to the security of the EUF-CMA$_{T,sk}$ game rather than the classical EUF-CMA game. This reasoning is valid as EUF-CMA implies EUF-CMA$_{T,sk}$ even in the presence of the corresponding oracle.

Proposition 3.8. *If a signature scheme* (Sign, Vrfy) *is EUF-CMA secure for keys given by* $\mathcal{A}_{sk}$*, then* (Sign, Vrfy) *is EUF-CMA*$_{T,sk}$ *secure for the oracle* $O^{sign}_{T,sk}$ *and the interpretation of keys* $\mathcal{A}_{sk}$*.*

4 MAIN COMPOSITION THEOREMS

We distinguish between two complementary cases. First, protocols composed in a way where they do not share states besides the shared secrets (e.g. parallel composition of different protocols using the same master secret key). Second, protocols passing states from one to the other (e.g. a key exchange passing an ephemeral key to a secure channel protocol). We finally extend these composition results to self-composition, i.e. proving the security of multiple sessions from the security of a single one.

4.1 Composition without state passing

Essentially, if two protocols P, Q are indistinguishable, they are still indistinguishable when running inside any simulatable context. The context must be simulatable for any fixed values of the shared names of P, Q and the context. The context can contain parallel or sequential composition as illustrated by the following example.

Example 4.1. Let P, Q, R, S be protocols and O an oracle. Let $\overline{n} = \mathcal{N}(P\|Q) \cap \mathcal{N}(R\|S)$. If $P \cong_O Q$ and $\nu\overline{n}.R\|S$ is O-simulatable, then theorem 4.3 yields $P\|R \cong_O Q\|R$, $R; P \cong_O R; Q$ and $(R; P)\|S \cong_O (R; Q)\|S$.

We generalize the previous example to any simulatable context and to n protocols. For any integer n, we denote by $C[\__1, \ldots, \__n]$ a *context*, i.e. a protocol built using the syntax of fig. 1 and distinct symbols $\__i$, viewed as elementary processes. $C[P_1, \ldots, P_n]$ is the protocol in which each hole $\__i$ is replaced with P_i. We say that a hole is terminal if it is not followed by any sequential composition.

Example 4.2. In the three cases of example 4.1, in order to apply the next theorem, we respectively use as contexts $C[\__1] := \__1 \| R$, $C[\__1] := R; \__1$ and $C[\__1] := (R; \__1) \| S$.

In this first theorem, no values (e.g. ephemeral keys) are passed from the context to the protocols. In particular, the protocols do not have free variables which may be bound by the context.

Theorem 4.3. *Given a cryptographic library $\mathcal{M}_f$ and an oracle O, let $P_1, \ldots, P_n, Q_1, \ldots, Q_n$ be protocols and $C[\__1, \ldots, \__n]$ be a context such that all their channels are disjoint, 0 some constant, $\overline{n}$ a sequence of names and $c_1, \ldots, c_n$ fresh channel names. If*

(1) $\mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n, Q_1, \ldots, Q_n) \subseteq \overline{n}$
(2) $\nu \overline{n}.C[\mathsf{out}(c_1, 0), \ldots, \mathsf{out}(c_n, 0)]$ *is O-simulatable*
(3) $P_1 \| \ldots \| P_n \cong_O Q_1 \| \ldots \| Q_n$

Then $C[P_1, \ldots, P_n] \cong_O C[Q_1, \ldots, Q_n]$

In addition, the advantage of the adversary in distinguishing $C[P_1, \ldots, P_n]$ and $C[Q_1, \ldots, Q_n]$ is bounded by the advantage of the adversary in distinguishing $P_1 \| \cdots \| P_n$ and $Q_1 \| \cdots \| Q_n$, with a polynomial overhead on the runtime of the simulator. Note that the bound we obtain for the reduction is polynomial in the running time of the context. The intuition behind the proof of the Theorem is as follows, where we denote $\overline{C} := C[\mathsf{out}(c_1, 0), \ldots, \mathsf{out}(c_n, 0)]$. Intuitively, $\overline{C}$ abstracts out the components P_i, only revealing which P_i is running at any time. We start by showing that $\overline{C} \| P_1 \| \ldots \| P_n \cong_O \overline{C} \| Q_1 \| \ldots \| Q_n$ implies $C[P_1, \ldots, P_n] \cong_O C[Q_1, \ldots, Q_n]$. This is done by a reduction, where we mainly have to handle the scheduling, which is possible thanks to the simulatability of $\overline{C}$, and the action determinism of the protocols. In a sense, this means that indistinguishability for protocols in parallel implies indistinguishability for any scheduling of those protocols. Secondly, by simulating the context thanks to proposition B.2, the hypothesis of the theorem implies $\overline{C} \| P_1 \| \ldots \| P_n \cong_O \overline{C} \| Q_1 \| \ldots \| Q_n$. The second part is where our notion of simulatability comes into play, and where it is essential to deal carefully with the shared secrets. The proof of theorem 4.3 can be found in Appendix C, and other proofs in [17].

Given a protocol P and a context C, for theorem 4.3 to be used, we need an oracle such that:

(1) the context C is simulatable with the oracle O,
(2) the protocol P is secure even for an attacker with access to O ($P \cong_O Q$).

Our goal is to find an oracle that is generic enough to allow for a simple proof of indistinguishability of P and Q under the oracle, but still allows to simulate the context. Notably, if we take as oracle the protocol oracle corresponding to the context itself, we can trivially apply theorem 4.3 but proving $P \cong_O Q$ amounts to proving $C[P] \cong C[Q]$.

4.1.1 Application to tagged protocols. We consider two versions of *SSH*, calling them *SSH*2 and *SSH*1, assuming that all messages are prefixed respectively with the strings "SSHv2.0" and "SSHv1.0". Both versions are using the same long term secret key sk for signatures. We assume that both versions check the string prefix.

To prove the security of *SSH*2 running in the context of *SSH*1, we can use theorem 4.3. We denote I the idealized version of *SSH*2, the desired conclusion is then $SSH2 \| SSH1 \cong I \| SSH1$. We use $C[\__1] = \__1 \| SSH1$, it is then sufficient to find an oracle O such that:

(1) $\nu sk.SSH1$ is O-simulatable (the simulatability of C directly follows),
(2) $SSH2 \cong_O I$

Defining T_{SSH1} as the function checking the prefix, *SSH*1 is trivially $O^{\mathsf{sign}}_{T_{SSH1}, sk}$ simulatable (see definition 3.5) as *SSH*1 does enforce the tagging checks. We thus let O be $O^{\mathsf{sign}}_{T_{SSH1}, sk}$.

Assuming that sign verifies the classical EUF-CMA axiom, by proposition 3.8, it also verifies the tagged version EUF-CMA$_{T_{SSH1}, sk}$. To conclude, it is then sufficient to prove that $SSH2 \cong_O I$ with a reduction to EUF-CMA$_{T_{SSH1}, sk}$.

4.1.2 Example of application to encrypt and sign. For performances considerations, keys are sometimes used both for signing and encryption, for instance in the EMV protocol. In [29], an encryption scheme is proven to be secure even in the presence of a signing oracle using the same key. Our Theorem formalizes the underlying intuition, i.e., if a protocol can be proven secure while using this encryption scheme, it will be secure in any context where signatures with the same key are also performed.

4.2 Composition with state passing

In some cases, a context passes a sequence of terms to another protocol. If the sequence of terms is indistinguishable from another one, we would like the two experiments, with either sequences of terms, to be indistinguishable.

Example 4.4. Let us consider once again the protocol $P(x_1, x_2) := \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(x, x_1, x_2))$ of example 2.1. We assume that we have a function kdf, which, given a random input, generates a suitable key for the encryption scheme. With *seed* a random name, let $C[\__1] := \mathsf{let}\ sk = \mathsf{kdf}(seed)\ \mathsf{in}\ \__1$. $C[\|^i P(r_i, sk)]$ provides an access to an encryption oracle for the key generated in C:

$$C[\|^i P(r_i, sk)] := \begin{array}{l} \mathsf{let}\ sk = \mathsf{kdf}(seed)\ \mathsf{in} \\ \qquad \|^i (\mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(x, r_i, sk))) \end{array}$$

Based on the fact that $C[\mathsf{out}(c, sk)] \cong_O C[\mathsf{out}(c, n)]$ for an oracle O and a fresh name n such that $\nu sk.P$ is O-simulatable, we can conclude that $C[\|^i P(r_i, sk)] \cong C[\|^i P(r_i, n)]$.

A classical example is a key exchange, used to establish a secure channel. The situation is dual with respect to theorem 4.3: contexts must be indistinguishable and the continuation simulatable.

Theorem 4.5. *Let C, C' be n-ary contexts such that each hole is terminal, and let $P_1(\overline{x}), \ldots, P_n(\overline{x})$ be parameterized protocols, such that all channel sets are pairwise disjoint. Given a cryptographic library $\mathcal{M}_f$, an oracle O , $\overline{n} \supseteq \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n)$, $\overline{t_1}, \ldots, \overline{t_n}, \overline{t'_1}, \ldots, \overline{t'_n}$ sequences of terms, if*

(1) $C[\mathsf{out}(c_1, \overline{t_1}), \ldots, \mathsf{out}(c_n, \overline{t_n})] \cong_O C'[\mathsf{out}(c_1, \overline{t'_1}), \ldots, \mathsf{out}(c_n, \overline{t'_n})]$
(2) $\nu \overline{n}.\mathsf{in}(c_1, \overline{x}).P_1(\overline{x}) \| \ldots \| \mathsf{in}(c_n, \overline{x}).P_n(\overline{x})$ *is O-simulatable*

then $C[P_1(\overline{t_1}), \ldots, P_n(\overline{t_n})] \cong_O C'[P_1(\overline{t'_1}), \ldots, P_n(\overline{t'_n})]$

In addition, the advantage of the adversary in distinguishing $C[P_1(\overline{t_1}), \ldots, P_n(\overline{t_n})]$ and $C[P_1(\overline{t'_1}), \ldots, P_n(\overline{t'_n})]$ is bounded by the advantage of the adversary in distinguishing $\widetilde{C}$ and $\widetilde{C'}$, with a polynomial overhead on the runtime of the simulator.

For the sake of simplicity, we often omit channels. When we do so, we only assume that they are all distinct. The following example shows how theorems 4.3 and 4.5 can be used to derive the security of one session of a key exchange composed with a protocol.

Example 4.6. Let us consider a key exchange $I\|R$ where x^I (resp. x^R) is the key derived by the initiator I (resp. the responder R) in case of success. We denote $KE[\__1, \__2] := I; \__1 \| R; \__2$ the composition of the key exchange with two continuations; the binding of x^I (resp. x^R) is passed to the protocol in sequence. Consider possible continuations $P^I(x^I), P^R(x^R)$ that use the derived keys and ideal continuations (whatever "ideal" is) $Q^I(x^I), Q^R(x^R)$. We sketch here how to prove $KE[P^I(x^I), P^R(x^R)] \cong KE[Q^I(x^I), Q^R(x^R)]$ (i.e., the security of the channel established by the key exchange). This will be generalized to multi-sessions in section 5. We use both theorems 4.3 and 4.5. Assume, with k a fresh name, that:

(1) O_{ke} is an oracle allowing to simulate the key exchange
(2) $O_{P,Q}$ allows to simulate $in(x).P^I(x)\|in(x).P^R(x)$ and $in(x).Q^I(x)\|in(x).Q^R(x)$
(3) $P^I(k)\|P^R(k) \cong_{O_{ke}} Q^I(k)\|Q^R(k)$
(4) $KE[out(x^I), out(x^R)] \cong_{O_{P,Q}} KE[out(k), out(k)]$

Hypothesis 3 captures the security of the channel when executed with an ideal key, and Hypothesis 4 captures the security of the key exchange. Both indistinguishability are for an attacker that can simulate the other part of the protocol.

Using theorem 4.3 with Hypothesis 1 and 3 yields

$$KE[P^I(k), P^R(k)] \cong KE[Q^I(k), Q^R(k)]$$

Hypothesis 2 and 4 yield, with two applications of theorem 4.5, one for P and one for Q, that $KE[P^I(x^I), P^R(x^R)] \cong KE[P^I(k), P^R(k)]$ and $KE[Q^I(x^I), Q^R(x^R)] \cong KE[Q^I(k), Q^R(k)]$. Transitivity allows us to conclude that the key exchange followed by the channel using the produced key is indistinguishable from the key exchange followed by the ideal secure channel:

$$KE[P^I(x^I), P^R(x^R)] \cong KE[Q^I(x^I), Q^R(x^R)]$$

4.3 Unbounded replication

An important feature of a compositional framework is the ability to derive the security of a multi session protocol from the analysis of a single session. To refer to multiple sessions of a protocol, we consider that each session uses some fresh randomness that we see as a local session identifier.

The main idea behind the Theorem is that the oracle will depend on a sequence of names of arbitrary length. This sequence of names represents the list of honest randomness sampled by each party of the protocol, and the oracle enables simulatability of those parties.

The following Theorem allows to prove the security of an arbitrary number of sessions of a protocol, even when the number of sessions depends on the security parameter.

Theorem 4.7. *Let O_r, O be oracles both parameterized by a sequence of names $\overline{s}$. Let $\overline{p}$ be a sequence of names, $P_i(\overline{x}, \overline{y})$ and $Q_i(\overline{x}, \overline{y}, \overline{z})$ be parameterized protocols, such that the set of indexed names of P and Q is disjoint of the oracles support. If we have, for sequences of names $\overline{lsid}^P, \overline{lsid}^Q$, with $\overline{s} = \{\overline{lsid_i}^P, \overline{lsid_i}^Q\}_{i\in\mathbb{N}}$:*

(1) $\forall\, i \geq 1, \nu\overline{p}, \overline{lsid_i}^P.P_i(\overline{p}, \overline{lsid_i}^P)$ is O_r simulatable.
(2) $\forall\, i \geq 1, \nu\overline{p}, \overline{lsid_i}^Q.Q_i(\overline{p}, \overline{lsid_i}^Q, \overline{s})$ is O_r simulatable.
(3) $\overline{s}$ is disjoint of the support of O.
(4) $P_0(\overline{p}, \overline{lsid_0}^P) \cong_{O_r, O} Q_0(\overline{p}, \overline{lsid_0}^Q, \overline{s})$

then, $||^i P_i(\overline{p}, \overline{lsid_i}^P) \cong_O ||^i Q_i(\overline{p}, \overline{lsid_i}^Q, \overline{s})$

To prove the security of an unbounded number of sessions in parallel, we can with this Theorem only prove the security of a single session (Hypothesis 4), if this proof holds when the attacker has access to an oracle that allows to simulate the other sessions of the protocol (Hypothesis 1,2). The extra oracle O allows to apply in sequence our theorems, and we thus require that this oracle does not interfere with the replication (Hypothesis 3).

The extra argument $\overline{s}$ of Q is not necessary for the above theorem. It is however useful in our applications, where Q is an idealized version. To prove this result, we use the explicit advantages that can be derived from our composition Theorems, which increases polynomially with respect to the number of sessions, and apply a classical hybrid argument to conclude.

In our applications (section 5), the main idea is to first use theorem 4.7 to reduce the multi-session security of a key exchange or a communication channel to a single session, and then use theorems 4.3 and 4.5 to combine the multiple key exchanges and the multiple channels.

5 APPLICATION TO KEY EXCHANGES

Although our framework is not tailored to key exchanges or any specific property, we choose here to focus on its application to key exchanges. We outline how our theorems may be used to prove the security of a protocol using a key derived by a key exchange in a compositional way. (Let us recall that the key exchange and the protocol using the derived key may share long term secrets). A formal corollary can be found in the long version [17].

5.1 Our model of key exchange

In order to obtain injective agreement, key exchanges usually use fresh randomnesses for each session as local session identifiers. For instance in the case of a Diffie-Hellman key exchange, the group shares may be seen as local session identifiers.

As in example 4.6, KE is a key exchange with possible continuations. In addition, we consider multiple copies of KE, indexed by i, and local *sid* for each copy:

$$KE_i[\__1, \__2] := I(lsid_i^I, id^I); \__1 \| R(lsid_i^R, id^R); \__2$$

id^X is the identity of X and $lsid^X$ represents the randomness that is be used by X to derive its local session identifier.

In the key exchange, I binds x^I to the key that it computes, x^I_{lsid} to the value of *lsid* received from the other party and x^I_{id} the received identity. Symmetrically, R binds the variables x^R, x^R_{lsid} and x^R_{id}. For simplicity, we only consider here the case of two fixed honest identities. The general case is considered in [17].

If we denote $P_i^I(x^I)\|P_i^R(x^R)$ the continuation meant to use the secret key derived in the key exchange, $KE_i[P_i^I(x^I), P_i^R(x^R)]$ is the composition of a session of the key exchange with the protocol where the values of x^I, x^R (computed keys) are passed respectively to $P_i^I(x^I)$ or $P_i^R(x^R)$. With Q an idealized version of P (however it

is defined), the security of the composed protocol is expressed as

$$\|^i KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^i KE_i[Q_i^I(x^I), Q_i^R(x^R)]$$

Intuitively, from the adversary point of view, P is equivalent to its idealized version, even if the key is derived from the key exchange as opposed to magically shared.

Equivalently, the security of the composed protocol is expressed as $\|^{i\leq N} KE_i[P_i^I(x^I), P_i^R(x^R)] \cong \|^{i\leq N} KE_i[Q_i^I(x^I), Q_i^R(x^R)]$, provided that the adversary advantage is polynomial in N.

5.2 Proofs of composed key exchange security

Following the same applications of theorems 4.3 and 4.5 as in example 4.6, we decompose the problem into the following goals:

(1) find an oracle $O_{P,Q}$ to simulate multiple sessions of P or Q,
(2) design an oracle O_{ke} to simulate multiple sessions of KE
(3) complete a security proof under O_{ke} for multiple sessions of the protocol using fresh keys,
(4) complete a security proof under $O_{P,Q}$ for multiple sessions of the key exchange.

We further reduce the security of the protocol to smaller proofs of single sessions of the various components of the protocols under well chosen oracles. The following paragraphs successively investigate how to simplify the goals (1),(2),(3),(4) above.

We denote $\overline{p} = \{id^I, id^R\}$ and assume that they are the only shared names between KE, P and Q and are the only names shared by two distinct copies P_i, P_j (resp. Q_i, Q_j). We also denote $\overline{s} = \{lsid_i^I, lsid_i^R\}_{i\in\mathbb{N}}$ the set of all copies of the local session identifiers.

5.2.1 Protocol simulatability. Assume that there is an oracle $O_{P,Q}$ such that $\nu\overline{p}.in(x^I).P_i^I(x^I)\|in(x^R).P_i^R(x^R)$ is $O_{P,Q}$ simulatable (and a similar result replacing P with Q), then, thanks to theorem 3.4, $\nu\overline{p}.\|^i(in(x^I).P_i^I(x^I)\|in(x^R).P_i^R(x^R))$ is $O_{P,Q}$ simulatable (and similarly for Q). This meets the condition (2) of theorem 4.5.

5.2.2 Key exchange simulatability. For the simulation of the key exchange context, we need N copies of KE and, in each of them, the initiator (resp. the responder) may communicate with N possible responders (resp. initiators). We therefore use theorem 4.3 with a context C with $2N^2$ holes. C is the parallel composition of N contexts and, as above, we use theorem 3.4 to get the condition (1) of theorem 4.3. Let KE_i' be[3]

$$KE_i[\ \underset{1\leq j\leq N}{\text{if}}\ x_{lsid}^I = lsid_j^R \text{ then } \mathsf{out}(\langle i,j\rangle) \text{ else } \bot,\\ \underset{1\leq j\leq N}{\text{if}}\ x_{lsid}^R = lsid_j^I \text{ then } \mathsf{out}(\langle i,j\rangle) \text{ else } \bot]$$

$\overline{C}$ is then $\|^{i\leq N} KE_i'$ and C can be inferred by replacing each $\mathsf{out}(\langle i,j\rangle)$ with a hole. Then, assuming that $\nu\overline{p}.KE_i'$ is O_{ke} simulatable, we get, thanks to theorem 3.4 the condition (1) of theorem 4.3.

5.2.3 Security of the protocol. Our goal is $\|^i P_i(k_i) \cong_{O_{ke}} \|^i Q_i(k_i)$. Based on theorem 4.7, we only need an oracle O_r so that:

(1) $\forall\ 1 \leq i, \nu\overline{p}, k_i.P_0(k_i)$ is O_r simulatable,
(2) $\forall\ 1 \leq i, \nu\overline{p}, k_i.Q_0(k_i)$ is O_r simulatable,
(3) $\overline{s}$ is disjoint of the support of O_{ke},
(4) $P_0(k_0) \cong_{O_r, O_{ke}} Q_0(k_0)$.

[3] we denote $\underset{1\leq j\leq N}{\text{if}}\ c_i \text{ then } a_i \text{ else } a' := \text{if } c_1 \text{ then } a_1 \text{ else if } c_2 \cdots \text{ then } a_n \text{ else } a'$

We use the fresh names k_i to model fresh magically shared keys, and use them as local sids for theorem 4.7. The intuition is similar to the notion of Single session game of [12], where the considered protocols are such that we can derive the security of multiple sessions from one session. For instance, if the key is used to establish a secure channel, revealing the other keys does not break the security of one session, but allows to simulate the other sessions.

5.2.4 Security of the key exchange. The security of the key exchange is more bothersome, in the sense that it cannot simply be written with a classical replication. The partnering of sessions is not performed beforehand, so we must consider all possibilities. We may express the security of a key exchange by testing the real-or-random for each possible session key. We denote $k_{i,j}$ the fresh name corresponding to the ideal key that will be produced by the i-th copy of the initiator believing to be partnered with the j-th copy of the responder. The security of the key exchange is captured through the following indistinguishability:

$$\|^{i\leq N} KE_i[\mathsf{out}(x^I), \mathsf{out}(x^R)] \cong_{O_{P,Q}} \\ \|^{i\leq N}\ KE_i[\ \underset{1\leq j\leq N}{\text{if}}\ x_{lsid}^I = lsid_j^R \text{ then } \mathsf{out}(k_{i,j}) \text{ else } \bot,\\ \underset{1\leq j\leq N}{\text{if}}\ (x_{lsid}^R = lsid_j^I) \text{ then } \mathsf{out}(k_{j,i}) \text{ else } \bot]$$

where the advantage of the attacker is polynomial in N.

Using a classical cryptographic hybrid argument (detailed in [17]), we reduce the security of multiple sessions to the security of one session in parallel of multiple corrupted sessions; the security of each step of the hybrid game is derived from eq. (1) using theorem 4.5. It is expressed, with $state_i^X = \langle x^X, lsid_i^X, x_{lsid}^X\rangle$, as

$$\begin{aligned}&\|^{i\leq N} KE_i[\mathsf{out}(\langle state_i^I\rangle), \mathsf{out}(\langle state_i^R\rangle)] \cong_{O_{P,Q}}\\ &\quad\|^{i\leq N-1} KE_i[\mathsf{out}(\langle state_i^I\rangle), \mathsf{out}(\langle state_i^R\rangle)]\\ &\quad\|\ KE_N[\ \text{if } x_{lsid}^I = lsid_N^R \text{ then } \mathsf{out}(\langle k, lsid_N^I, x_{lsid}^I\rangle)\\ &\qquad\text{else if } x_{lsid}^I \notin \{lsid_i^R\}_{1\leq i\leq N-1} \text{ then } \bot,\\ &\qquad\text{else } \mathsf{out}(\langle state_i^I\rangle),\\ &\qquad\text{if } x_{lsid}^R = lsid_N^I \text{ then } \mathsf{out}(\langle k, lsid_N^R, x_{lsid}^R\rangle)\\ &\qquad\text{else if } x_{lsid}^R \notin \{lsid_i^I\}_{1\leq i\leq N-1} \text{ then } \bot,\\ &\qquad\text{else } \mathsf{out}(\langle state_i^R\rangle)]\end{aligned} \tag{1}$$

We further reduce the problem to proving the security of a single session even when there is an oracle simulating corrupted sessions. To this end, we need to reveal the dishonest local session's identifiers to the attacker, but also to allow him to perform the required cryptographic operations, e.g. signatures using the identities.

We define, for $X \in \{I, R\}$, $\overline{s}^X$ as the set of copies of the local session identifiers of I or R, except a distinguished one (indexed 0 below) and $\overline{s} = \overline{s}^I \cup \overline{s}^R$. To obtain the security of multiple sessions of the key exchange, we then only have to use theorem 4.3[4] with the following assumption: :

(1) $\forall 1 \leq i \leq N-1, \nu lsid_i^I, id^I, lsid_i^R, id^R.$
$KE_i[\mathsf{out}(x^I), \mathsf{out}(x^R)]\|\mathsf{out}(\langle lsid_i^R, lsid_i^I\rangle)$ is O_T simulatable.
(2) $\overline{s}$ is disjoint of the support of $O_{P,Q}$.
(3) $KE_N[\mathsf{out}(\langle x^I, lsid_N^I, x_{lsid}^I\rangle), \mathsf{out}(\langle x^R, lsid_N^R, x_{lsid}^R\rangle)]$
$\cong_{O_T, O_{P,Q}} KE_N[C_{I,R}, C_{R,I}]$

[4] We also use theorem 3.4 to get the simulatability of N sessions in parallel from the simulatability of each session. In [17], this is formalized inside a dedicated Proposition.

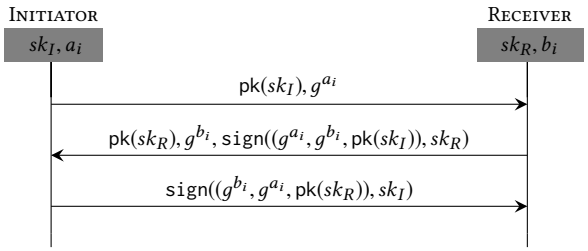

Figure 5: ISO 9798-3 Diffie Hellman key exchange

$$\text{with } C_{X,Y} := \begin{cases} \text{if } x^X_{lsid} = lsid^Y_N \text{ then } \mathsf{out}(\langle k, lsid^X_N, x^X_{lsid}\rangle) \\ \text{else if } x^X_{lsid} \notin \overline{s}^Y \text{ then } \bot \\ \text{else } \mathsf{out}(\langle x^X, lsid^X_N, x^X_{lsid}\rangle) \end{cases}$$

Intuitively, if the initiator believes to be talking to the honest responder, then he outputs the ideal key, and if it is not talking to any simulated corrupted party, it raises a bad event.

Note that while the structure of the proof does not fundamentally change from other proofs of key exchanges, e.g [12], each step of the proof becomes straightforward thanks to our composition results. Our proofs are also more flexible, as shown by the extension to key exchanges with key confirmation in section 7.

6 BASIC DIFFIE-HELLMAN KEY EXCHANGE

We outline here the application of our framework to the ISO 9798-3 protocol, proven UC composable in [26]. We use our result to extend the security proof to a context with shared long term secrets. We present the protocol in fig. 5, and show how to instantiate the required values and oracles to perform the proof presented in section 5.2. The formal proofs (using the CCSA model [6]) are provided in [17].

The identity of each party is its long term signing key, and thus, we use sk_I and sk_R as id_I and id_R. Each session of the key exchange instantiates a fresh Diffie-Hellman share, that can be seen as a local session identifier. We use g^{a_i} and g^{b_i} as $lsid^I_i$ and $lsid^R_i$. These values can also be used as implicit tagging since any signed message either depends on a_i or b_i. To define this implicit tagging, we extend the tagging function T of definition 3.5 so that it may depend on a second argument of arbitrary length, yielding $T(m, \overline{s})$, the corresponding signing oracle being denoted $O^{\mathsf{sign}}_{T,sk,\overline{s}}$. The exact definition is given later in definition 8.2. Letting $\overline{s}^I = \{a_i | i \geq 1\}$ and $\overline{s}^R = \{b_i | i \geq 1\}$, we define the implicit tagging functions T^I and T^R as $T^X(m, \overline{s}) := \exists s \in \overline{s}^X, \exists m_1, m_2.m = (m_1, g^s, m_2)$. With $O_T = O^{\mathsf{sign}}_{T^I,sk_I,\overline{s}}, O^{\mathsf{sign}}_{T^R,sk_R,\overline{s}}, O_{\overline{s}}$, where $O_{\overline{s}}$ simply reveals the elements in $\overline{s}$, we obtain the simulatability of multiple sessions of the key exchange (hypothesis 2 of section 5.2). For hypothesis 4 of section 5.2 we use the method of section 5.2.4: we consider the key exchange KE_0, followed either by the output of the computed keys, the computed $lsid$ and the real $lsid$ or by $C_{I,R}, C_{R,I}$.

The protocol sketched in fig. 5 actually assumes some verifications; for instance the value sent in the first message should match g^{a_i} in the last message. Therefore, when the protocol of fig. 5 is successfully completed, we can prove that if $x^I_{lsid} \neq g^{b_0}$, then $x^I_{lsid} \in \{g^{b_i} | i \in \mathbb{N}\}$, i.e. $T^I(x^R_{lsid})$ is true (and similarly for R). It follows that $C_{I,R}, C_{R,I}$ either corresponds to a matching conversation between the sessions with sids g^{a_0}, g^{b_0}, in which case the output is (twice) an ideal key k, or else it is a matching conversation with a simulated session, in which case it outputs the computed keys. The proof of the property 3) of section 5.2.4 is thus a real-or-random proof of a honestly produced key.

The previous security proof can be performed under oracle $O_{P,Q}$ that allows to simulate the continuation (hypothesis 1 of section 5.2). The continuation should be proven secure when using an ideal key (hypothesis 3 of section 5.2). In some cases, this step is trivial. Indeed, let us consider a record protocol $L := L^I(x^I) \| L^R(x^R)$, that exchanges encrypted messages using the exchanged key, and does not share any long term secret. Without any shared secret, we do not need any oracle to simulate $in(k); L^I(k) \| in(k); L^R(k)$, we can choose a trivial $O_{P,Q}$ that does nothing.

7 EXTENSION TO KEY CONFIRMATIONS

We consider key exchanges where the key is derived in a first part of the protocol and then used in the second part. We proceed as in section 5, outlining how we may split the security proof into smaller proofs, using the same composition theorems at each step. Compared to [11], our method allows in addition long term secret sharing.

Consider a key exchange $I_i(lsid^I_i, id^I) \| R_i(lsid^R_i, id^R)$. We further split I and R into $I_i := I^0_i; I^1_i$ and $R_i := R^0_i; R^1_i$, where I^0_i and R^0_i correspond to the key exchange up to, but not including, the first use of the secret key, and I^1_i and R^1_i are the remaining parts of the protocol. The intuition behind the proof of security is that at the end of I^0_i and R^0_i, i.e. just before the key confirmation, either the sessions are partnered together and the derived key satisfies the real-or-random, or they are not and the key confirmation will fail. The key point for applying the composition theorems is that any protocol using the derived key must then be secure in the presence of an oracle O such that I^1_i, R^1_i is O simulatable (in particular the derived key is in the support of O). For typical cases, this is rather simply an oracle providing a hash of the key or some specific encryption of the confirmation message with the key. Specific hypotheses and details of the technique are provided in appendix D.

7.1 Application to SSH

SSH [32] is a protocol that allows a user to login onto a server from its platform. It is widely used in the version where signatures are used for authentication. An interesting feature is agent forwarding: once a user u is logged on a server S, he may, from S, perform another login on another server T. As S does not have access to the signing key of u, it forwards a signature request to the u's platform using the secure SSH channel between u and S. This represents a challenge: we compose a first key exchange with another one, the second one using a signature key already used in the first.

There is a known weakness in this protocol: any privileged user on S could use the agent as a signing oracle. Thus, in order to be able to prove the security of the protocol, we only consider the case where there is no such privileged user. Figure 6, in Appendix, presents an example of a login followed by a login using the agent

forwarding. For simplicity, we abstract away some messages that are not relevant to the security of the protocol.

In the current specification of the forwarding agent, it is impossible for a server to know if the received signature was completed locally by the user's platform, or remotely through the agent forwarding. As the two behaviors are different in term of trust assumptions, we claim that they should be distinguishable by a server. For instance, a server should be able to reject signatures performed by a forwarded agent, because intermediate servers are not trusted. To this end, we assume that the signatures performed by the agent are (possibly implicitly) tagged in a way that distinguishes between their use in different parts of the protocol.

We consider a scenario, in which there is an unbounded number of sessions of SSH, each with one (modified) agent forwarding, used to provide a secure channel for a protocol P. Thanks to multiple applications of theorems 4.3 and 4.5, we are able to break the proof of this SSH scenario into small ones, that are very close to the proof of a simple Diffie-Hellman key exchange. This assumes the *decisional Diffie-Hellman* (DDH) hypothesis for the group, EUF-CMA for the signature scheme and that the encryption is authenticated. P also has to satisfy the assumptions of appendix D.3. In particular, it must be secure w.r.t. an attacker that has access to a hash that includes the exchanged secret key, since SSH produces such a hash. Details can be found in [17]. Note that the scenario includes multiple sessions, but only one forwarding. The extension would require an induction to prove in our framework the security for any number of chained forwardings.

8 FORMAL PROOFS IN THE CCSA MODEL

All the above proofs are cast in the CCSA model, which is well-suited for our purposes. In this model, formulas are first-order formulas built over a single predicate symbol $\sim$. Hypotheses on the cryptographic libraries are given as a (recursive) set of formulas A, called axioms. The protocols P, Q are folded into terms t_P, t_Q, in such a way that P is computationally indistinguishable from Q if $t_P \sim t_Q$ is derivable from A.

This result (computational soundness in [6]) also holds when the attacker has access to an oracle O, provided that the formulas in A are sound with respect to such an attacker (in such a case, we say that the formulas are O-sound). Details can be found in [17]. Therefore, we have to design axioms that are O-sound. Let us give one such example, used in our proofs. Our definition of the tagging predicate may (here) depend on a possibly infinite list of secrets, as required in section 5 to verify that a session identifier belongs to the set of honest session identifiers.

Definition 8.1. Given a name sk, a sequence of names $\overline{s}$ and a function symbol T, we define the generic axiom scheme EUF-CMA$_{T,sk,\overline{s}}$ as, for any term t such that sk is only in key position:

$$\mathtt{checksign}(t, pk(sk)) \Rightarrow T(\mathtt{getmess}(t), \overline{s}) \bigvee_{\mathtt{sign}(x,sk)\in \mathtt{St}(t)} (t = \mathtt{sign}(x, sk))$$

The above formula is written in a simplified form (not using equivalences) and we do not define formally all its components. Its intuitive meaning is: if t is a valid signature with sk, then either the signed message satisfies the predicate T (the intention is "the message is correctly tagged") or it is a message that has been honestly produced in the past. The tagged signing oracle is defined as previously, only adding the extra argument to the tagging function.

Definition 8.2. Given a name sk, a sequence of names $\overline{s}$, and a predicate T, we define the generic signing oracle $O^{sign}_{T,sk,\overline{s}}$ as follows:

$$O^{\mathrm{sign}}_{T,sk,\overline{s}}(m) := \text{if } T(m, \overline{s}) \text{ then } \ \mathrm{output}(\mathrm{sign}(m, sk)))$$

Proposition 8.3. *For any computational model in which the interpretation of sign is EUF-CMA, any name sk, any sequence of names $\overline{s}$ such that $sk \notin \overline{s}$, and any polynomially computable interpretation of T, EUF-CMA$_{T,sk,\overline{s}}$ is $O^{sign}_{T,sk,\overline{s}}$-sound.*

9 CONCLUSION

In summary, we designed a method that allows to decompose a security property of a compound protocol into security properties of its components. This works for parallel composition, but also sequential composition and replication: we designed a reduction from the security of multiples copies of a protocol to a security property of a single copy. Our method works even if the various components share secrets and (in case of sequential composition) when a state is passed to the other component. However, designing oracles can be a technical challenge of our model.

We illustrated the results with the applications to key exchanges and the SSH protocol with agent forwarding. Up to our knowledge these applications, in the general setting that we consider, cannot be covered by other methods. As future work, we plan to explore more complex properties of key exchanges and study other applications (MPC, e-voting), designing new applications of our theorems and new axioms when needed.

This approach is also well-suited for the formal proofs in the CCSA model. This model is designed for formal proofs that are computationally sound, but also works for stronger attackers, who typically can access some oracles. Though the O-soundness proofs of the axioms, with standard cryptographic assumptions, have to be completed by hand, the core proofs can be completely formal and mechanizable. We plan to develop this mechanization.

In parallel, it could also be used to help proving complex protocols in EasyCrypt [7] for example, as security w.r.t. an attacker accessing an oracle can be formalized in this tool.

Acknowledgements. We wish to thank the anonymous reviewers for their useful comments, as well as Bogdan Warinschi and Ralf Küsters for interesting discussions. We are grateful for the support by the ANR under the TECAP grant (ANR-17-CE39-0004-01) and by the Institut Universitaire Français.

REFERENCES

[1] [n. d.]. ISO/IEC 9798-3:2019, IT Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques. https://www.iso.org/standard/67115.html

[2] M. Arapinis, V. Cheval, and S. Delaune. 2012. Verifying Privacy-Type Properties in a Modular Way. In *2012 IEEE 25th Computer Security Foundations Symposium*. 95–109. https://doi.org/10.1109/CSF.2012.16

[3] Michael Backes, Markus Dürmuth, Dennis Hofheinz, and Ralf Küsters. 2008. Conditional reactive simulatability. *Int. J. Inf. Sec.* 7, 2 (2008), 155–169.

[4] Michael Backes, Birgit Pfitzmann, and Michael Waidner. 2007. The Reactive Simulatability (RSIM) Framework for Asynchronous Systems. *Inf. Comput.* 205, 12 (Dec. 2007), 1685–1720. https://doi.org/10.1016/j.ic.2007.05.002

[5] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. 2018. Formal Analysis of Vote Privacy Using Computationally Complete Symbolic Attacker. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*. 350–372. https://doi.org/10.1007/978-3-319-98989-1_18

[6] Gergei Bana and Hubert Comon-Lundh. 2014. A Computationally Complete Symbolic Attacker for Equivalence Properties. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS'14)*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, Scottsdale, Arizona, USA, 609–620. https://doi.org/10.1145/2660267.2660276

[7] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *Advances in Cryptology – CRYPTO 2011 (Lecture Notes in Computer Science)*, Vol. 6841. Springer, Heidelberg, 71–90.

[8] Bruno Blanchet. 2007. CryptoVerif: A Computationally Sound Mechanized Prover for Cryptographic Protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied"*.

[9] Bruno Blanchet. 2018. Composition Theorems for CryptoVerif and Application to TLS 1.3. In *31st IEEE Computer Security Foundations Symposium (CSF'18)*. IEEE Computer Society, Oxford, UK, 16–30.

[10] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. 2018. State Separation for Code-Based Game-Playing Proofs. In *ASIACRYPT (3) (Lecture Notes in Computer Science)*, Vol. 11274. Springer, 222–249.

[11] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. 2013. Less is more: relaxed yet composable security notions for key exchange. *International Journal of Information Security* 12, 4 (Aug. 2013), 267–297. https://doi.org/10.1007/s10207-013-0192-y

[12] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. 2011. Composability of Bellare-rogaway Key Exchange Protocols. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, New York, NY, USA, 51–62. https://doi.org/10.1145/2046707.2046716

[13] David Cadé and Bruno Blanchet. 2013. From Computationally-Proved Protocol Specifications to Implementations and Application to SSH. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)* 4, 1 (March 2013), 4–31.

[14] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. 2019. *iUC: Flexible Universal Composability Made Simple.* Technical Report.

[15] Ran Canetti. 2000. *Universally Composable Security: A New Paradigm for Cryptographic Protocols.* http://eprint.iacr.org/2000/067

[16] Ran Canetti and Tal Rabin. 2003. Universal Composition with Joint State. In *Advances in Cryptology - CRYPTO 2003 (Lecture Notes in Computer Science)*, Dan Boneh (Ed.). Springer Berlin Heidelberg, 265–281.

[17] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. 2020. Oracle Simulation: a Technique for Protocol Composition with Long Term Shared Secrets - long version. https://hal.inria.fr/hal-02913866

[18] Hubert Comon and Adrien Koutsos. 2017. Formal Computational Unlinkability Proofs of RFID Protocols. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, Boris Köpf and Steve Chong (Eds.). IEEE Computer Society Press, Santa Barbara, California, USA, 100–114. https://doi.org/10.1109/CSF.2017.9

[19] Véronique Cortier and Stéphanie Delaune. 2009. A method for proving observational equivalence. In *2009 22nd IEEE Computer Security Foundations Symposium*. IEEE, 266–276.

[20] Cas Cremers. 2008. On the Protocol Composition Logic PCL. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security (ASIACCS '08)*. ACM, New York, NY, USA, 66–76. https://doi.org/10.1145/1368310.1368324 event-place: Tokyo, Japan.

[21] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. 2005. Probabilistic Polynomial-Time Semantics for a Protocol Security Logic. In *Automata, Languages and Programming (Lecture Notes in Computer Science)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer Berlin Heidelberg, 16–29.

[22] Nancy Durgin, John Mitchell, and Dusko Pavlovic. 2003. A Compositional Logic for Proving Security Properties of Protocols. *J. Comput. Secur.* 11, 4 (July 2003), 677–721. http://dl.acm.org/citation.cfm?id=959088.959095

[23] Marc Fischlin and Felix Günther. 2014. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1193–1204. https://doi.org/10.1145/2660267.2660308 event-place: Scottsdale, Arizona, USA.

[24] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.* 17, 2 (1988), 281–308.

[25] Dennis Hofheinz and Victor Shoup. 2015. GNUC: A New Universal Composability Framework. *Journal of Cryptology* 28, 3 (July 2015), 423–508. https://doi.org/10.1007/s00145-013-9160-y

[26] Ralf Kusters and Daniel Rausch. 2017. A Framework for Universally Composable Diffie-Hellman Key Exchange. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Jose, CA, USA, 881–900. https://doi.org/10.1109/SP.2017.63

[27] Ralf Küsters and Max Tuengerthal. 2011. Composition Theorems Without Pre-established Session Identifiers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/2046707.2046715 event-place: Chicago, Illinois, USA.

[28] Ueli Maurer. 2011. Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In *TOSCA (Lecture Notes in Computer Science)*, Vol. 6993. Springer, 33–56.

[29] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam, and Susan Thomson. 2011. On the Joint Security of Encryption and Signature, Revisited. In *Advances in Cryptology – ASIACRYPT 2011 (Lecture Notes in Computer Science)*, Dong Hoon Lee and Xiaoyun Wang (Eds.). Springer Berlin Heidelberg, 161–178.

[30] Guillaume Scerri and Stanley-Oakes Ryan. 2016. Analysis of Key Wrapping APIs: Generic Policies, Computational Security. IEEE Computer Society, 281–295. https://doi.org/10.1109/CSF.2016.27

[31] Stephen C. Williams. 2011. Analysis of the SSH Key Exchange Protocol. In *Cryptography and Coding (Lecture Notes in Computer Science)*, Liqun Chen (Ed.). Springer Berlin Heidelberg, 356–374.

[32] Tatu Ylonen and Chris Lonvick. [n. d.]. The Secure Shell (SSH) Transport Layer Protocol. https://tools.ietf.org/html/rfc4253

A STATELESS ORACLES

Definition A.1 ((Stateless) Oracle). An *oracle* O is a triple of functions that have the following inputs

- a sequence of bit-strings $\overline{w} \in (\{0,1\}^*)^n$ and two bit-strings r, s: the query, consisting of an *input query* $\overline{w}$, an *input tag* r, an *input key* s.
- a random tape ρ_s for the (secret) random values
- the security parameter η
- a random tape ρ_O for the oracle's coins.

The first function assigns to each $\overline{w}, s, r$ an integer $n(\overline{w}, s, r) \in \mathbb{N}$ and is assumed injective. $n(\overline{w}, s, r)$ is used to extract a substring $e_1(n(\overline{w}, s, r), \eta, \rho_O)$ from ρ_O, which is uniquely determined by the input. We assume that the length of the substring extracted by e_1 only depends on η, and substrings extracted with e_1 are disjoint for different values of n.

The second function e_2 assigns to each s a sequence $\overline{p}(s)$ of natural numbers, that are used to extract secret values from ρ_s: $e_2(\overline{p}(s), \eta, \rho_s)$ is a sequence of bit-strings. It is also assumed to be injective.

The third function takes $\eta, \overline{w}, r, s, e_1(n(\overline{w}, r), \eta, \rho_O), e_2(\overline{p}(s), \eta, \rho_s)$ as input and returns a result (a bit-string) or a failure message.

B SIMULATABILITY

For technical reasons, we need to ensure that simulator's oracle calls and attacker's oracle calls do not use shared randomness. We thus assume, w.l.o.g., that the random handles r of simulator's queries are prefixed by 1. This ensures that, as long as adversaries only make oracle calls prefixed by 0 (this can be assumed w.l.o.g. since it only constrains the part of the oracle's random tape where the randomness is drawn.) the oracle randomness used by the simulator is not used by the adversary. This is illustrated in example B.3.

From now on, we replace the definition of simulatability by the following one:

Definition B.1. Given a cryptographic library $\mathcal{M}_f$, a sequence of names $\overline{n}$, an oracle O and a protocol P, we say that $\nu\overline{n}.P$ is O-*simulatable* if the support of O is $\overline{n}$ and there is a PPTOM $\mathcal{A}^O$ (using queries prefixed by 0) such that, for every $c \in \{0,1\}^\star$, for

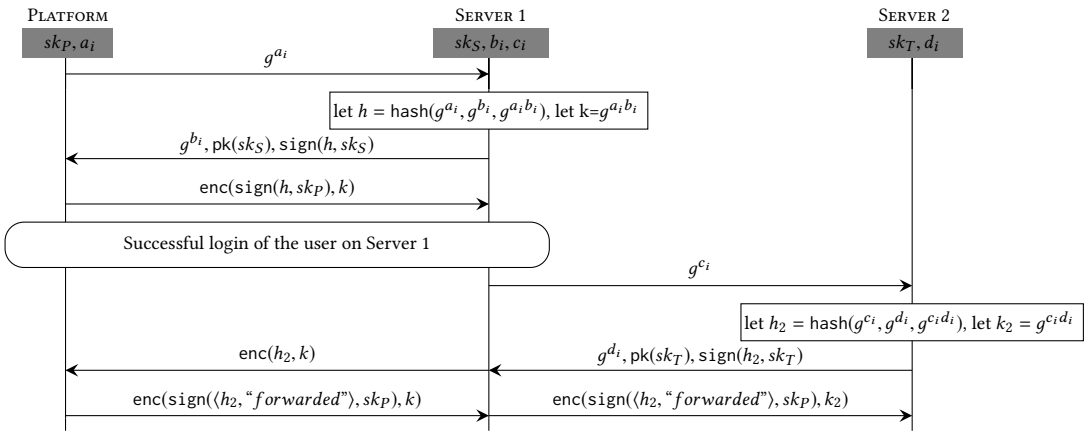

Figure 6: SSH with agent forwarding

every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}$, for every $m \geq 1$, for every PPTOM $\mathcal{B}^O$ (using random handles prefixed by 1),

$$\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{\mathcal{A}^{O(\rho_s,\rho_O)}(\rho_{r_1},\theta^1_m,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{O_P(\rho_s,\theta^2_m) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

where

$$\phi^2_{k+1} = \phi^2_k, O_P(\rho_s,\theta^2_k)$$
$$\phi^1_{k+1} = \phi^1_k, \mathcal{A}^{O(\rho_s,\rho_O)}(\rho_{r_1},\theta^1_k,\eta)$$
$$\theta^i_{k+1} = \theta^i_k, \mathcal{B}^{O(\rho_s,\rho_O)}(\rho_{r_2},\eta,\phi^i_{k+1})$$

for $0 \leq k < m$ and $\phi_0 = \emptyset$, $\theta_0 = \mathcal{B}^{O(\rho_s,\rho_O)}(\rho_{r_2},\eta,\emptyset)$.

The machine $\mathcal{A}$ can be seen as the simulator, while $\mathcal{B}$ is an adversary that computes the inputs: the definition states that there is a simulator, independently of the adversary.

This new definition of simulatability is equivalent to the one based on definition 3.1, as shown in proposition B.2. Alternative definitions for simulatability are discussed in appendix B.3.

Proposition B.2. *Given an oracle O with support $\overline{n}$, a cryptographic library $\mathcal{M}^f$, protocols P, Q such that $\mathcal{N}(P) \cap \mathcal{N}(Q) \subseteq \overline{n}$, then, for any PPTOM $\mathcal{A}^O_P$, $\nu\overline{n}.P$ is O-simulatable with $\mathcal{A}^O_P$ if and only if for every PPTOM $\mathcal{D}^{O,O_P,O_Q}$, for every η, every $\overline{v} \in (\{0,1\}^\eta)^{|\overline{n}|}$, $c \in \{0,1\}^\star$,*

$$\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P,O_Q}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^{O,O_Q}(\rho_r,1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$$

The above characterization is the one we will use inside the proofs, where the extra protocol Q will be the context. The definition 3.1 was implicitly extended to support a distinguisher with an additional protocol oracle.

B.1 A detailed example of protocol simulation

The purpose of the following example is to illustrate the definitions and also to show the need for disjointness of the oracle tags of the attacker from the oracle tags of the simulator.

Example B.3. We take a more formal view on the example from example 3.3.

We let O be the encryption-decryption oracle: it expects an input ⟨"dec", m⟩ or ⟨"enc", m⟩, a key $s = 1$ (only one encryption key is considered), an input tag t and a security parameter η and returns

- enc(m, r, k) if the query is prefixed by "enc", k is the secret value extracted from ρ_s corresponding to the key 1, r is drawn from ρ_O and associated with the tag t (via e_1).
- dec(m, k) if the query is prefixed by "dec", k is the secret value extracted from ρ_s corresponding to the key 1
- an error message otherwise (either the primitives fail or the query does not have the expected format).

The goal is to show that $\nu k.A$ is O-simulatable. (So, here, B is useless, and we let P be A).

O_P is then defined as follows (according to the section 2.5):

- On input w_1, with an empty history, it outputs $[\![\text{enc}(n,r,k)]\!]^\eta_{\rho_s}$ and writes w_1 on the history tape.
- On input w_2 with a non empty history tape, it outputs ok if $[\![\text{dec}(x,k)]\!]^{\eta,x\mapsto w_2}_{\rho_s} = [\![\langle n,1\rangle]\!]^\eta_{\rho_s}$ and an error otherwise.

The machine $\mathcal{A}^O(\rho_{r_1},\theta,\eta)$ is then defined as follows:

- If $\theta = \{m_1\}$
  (1) $\mathcal{A}$ draws α (for the value of n) from ρ_{r_1} and draws t from ρ_{r_1}
  (2) calls O with (⟨"enc", α⟩, 1, t) and gets back the bitstring $[\![\text{enc}(n,r,z)]\!]^{\eta,z\mapsto[\![k]\!]^\eta_{\rho_s}}_{\rho_{r_1},\rho_O}$. The interpretation of k is indeed fixed at once since it belongs to the "shared" names bounded by ν.
  (3) outputs $[\![\text{enc}(x,r,z)]\!]^{\eta,x\mapsto\alpha,z\mapsto[\![k]\!]^\eta_{\rho_s}}_{\rho_{r_1}}$
- If $\theta = (m_1, m_2)$,
  (1) calls O with (⟨"dec", m_2⟩, 1, −) and gets back the bitstring $w = [\![\text{dec}(y,z)]\!]^{y\mapsto m_2,z\mapsto[\![k]\!]^\eta_{\rho_s}}$ or an error message.
  (2) checks whether $w = [\![\langle n,1\rangle]\!]^\eta_{\rho_{r_1}}$. If it is the case, then outputs ok.

Now, consider an arbitrary PPTOM $\mathcal{B}^O$.

- $\phi^1_1 = [\![\text{enc}(n,x,z)]\!]^{\eta,x\mapsto s_1,z\mapsto[\![k]\!]^\eta_{\rho_s}}_{\rho_{r_1}}$ where s_1 is the randomness used by O when queried with $[\![t]\!]_{\rho_{r_1}}$ (note: we will see

that it does matter to be very precise here; we cannot simply claim that the value of x is just a randomness drawn by O).

- $\phi_1^2 = [\![\mathrm{enc}(n, r, k)]\!]^\eta_{\rho_s}$
- $\theta_i^1 = w_i$, an arbitrary bit-string, computed by $\mathcal{B}^O$ using the oracle O, ϕ_i^1 and the random tape ρ_{r_2}.
- $\phi_2^1 = \phi_1^1$, ok if $[\![\mathrm{dec}(y, z)]\!]^{y\mapsto w_1, z\mapsto [\![k]\!]^\eta_{\rho_s}}_{\rho_{r_1}} = [\![\langle n, 1\rangle]\!]^\eta_{\rho_{r_1}}$ and an error otherwise
- $\phi_2^2 = \phi_1^2$, ok if $[\![\mathrm{dec}(x, k)]\!]^{\eta, x\mapsto w_2}_{\rho_s} = [\![\langle n, 1\rangle]\!]^\eta_{\rho_s}$ and an error otherwise

$\mathcal{A}$ O-simulates $vk.P$ iff, for every $v = [\![k]\!]_{\rho_s}$,

$$\mathbb{P}_{\rho_s,\rho_{r1},\rho_{r_2},\rho_O}\{[\![\mathrm{dec}(y, z)]\!]^{y\mapsto w_1, z\mapsto v} = [\![\langle n, 1\rangle]\!]^\eta_{\rho_{r_1}}\}$$
$$= \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{[\![\mathrm{dec}(x, k)]\!]^{\eta, x\mapsto w_2}_{\rho_s} = [\![\langle n, 1\rangle]\!]^\eta_{\rho_s}\}$$

First, the distributions of ϕ_1^1 and ϕ_1^2 are identical. ϕ_1^1 depends on ρ_{r_1} and ρ_O, while ϕ_1^2 depends on ρ_s only. The distributions of $\phi_1^1, [\![\langle n, 1\rangle]\!]_{\rho_{r_1}}$ and $\phi_1^2, [\![\langle n, 1\rangle]\!]_{\rho_s}$ are also identical.

Now the distributions $w_1 = \mathcal{B}^O(\phi_1^1, \rho_{r_2}), [\![\langle n, 1\rangle]\!]_{\rho_{r_1}}$ and $w_2 = \mathcal{B}^O(\phi_2^1, \rho_{r_2}), [\![\langle n, 1\rangle]\!]_{\rho_s}$ are equal *if the randomness used by $\mathcal{B}$ are disjoint from the randomnesses used in ϕ_1^1, ϕ_1^2*. This is why there is an assumption that ρ_{r_1} and ρ_{r_2} are disjoint and why *it should be the case* that the randomnesses used in the oracle queries of $\mathcal{B}$ are distinct from the randomnesses used in the oracle queries of $\mathcal{A}$. This can be ensured by the disjointness of tags used by $\mathcal{A}$ and $\mathcal{B}$ respectively.

With these assumptions, we get the identity of the distributions of $\mathrm{dec}(w_1, v), [\![\langle n, 1\rangle]\!]_{\rho_s}$ and $\mathrm{dec}(w_2, v), [\![\langle n, 1\rangle]\!]_{\rho_s}$, hence the desired result.

Without these assumptions (for instance non-disjointness of tags used by $\mathcal{B}$, $\mathcal{A}$), $\mathcal{B}$ can query O with a random input and a random tag, say n', t'. As above, we let s_1 be the random value drawn by O corresponding to the tag t'. Then $\mathbb{P}\{[\![n]\!]_{\rho_s} = n' \wedge [\![r]\!]_{\rho_s} = s_1\} = \frac{1}{2^{2\eta}}$ while

$$\mathbb{P}\{[\![n]\!]_{\rho_{r_1}} = n' \wedge [\![r]\!]_{\rho_{r_1}} = s_1\} =$$
$$\tfrac{1}{2^\eta}\mathbb{P}\{[\![t]\!]_{\rho_{r_1}} = [\![t']\!]_{\rho_{r_2}} \vee ([\![t]\!]_{\rho_{r_1}} \neq [\![t']\!]_{\rho_{r_2}} \wedge [\![r]\!]_{\rho_{r_1}} = [\![r']\!]_{\rho_O})\}$$
$$= \tfrac{1}{2^\eta} \times (\tfrac{1}{2^\eta} + \tfrac{2^\eta - 1}{2^\eta} \times \tfrac{1}{2^\eta})$$
$$= \tfrac{1}{2^{2\eta}}(2 - \tfrac{1}{2^\eta})$$

In other words, the collision is more likely to occur since it can result from either a collision in the tags or a collision in the randomness corresponding to different tags.

B.2 Prefixed model

As demonstrated in ther previous example, it is necessary to assume that oracle randomness used by the simulator queries and the attacker queries are disjoint. The simplest way of ensuring this is to force all tags of oracle calls to be prefixed. We show here that this assumption can be made without loss of generality.

Definition B.4. Given a PPTOM $\mathcal{A}^O$ and a constant c. We define $\mathcal{A}^O_{pref-c}$ as a copy of $\mathcal{A}$, except that all calls to the oracle of the form $\overline{w}, r, s$ are replaced with calls of the form $\overline{w}, c \cdot r, s$, where the $\cdot$ denotes the concatenation of bit-strings.

The following lemma shows that we can, w.l.o.g., consider models, in which the tags are prefixed. It uses the formal definition of a stateless oracle of Appendix A.

Lemma B.5. *For any non-empty constant c and any PPTOM $\mathcal{A}^O$, we haves*

$$\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{A}^{O(\rho_s,\rho_O)}(\rho_r, 1^\eta) = 1\}$$
$$= \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{A}^{O(\rho_s,\rho_O)}_{pref-c}(\rho_r, 1^\eta) = 1\}|$$

Proof. We fix a constant c, for any oracle O (with functions n, e_1, e_2), we define O_{pref-c} (with mapping function n', e_1', e_2') the copy of O such that:

$$n'(w, s, r) = n(w, s, c|r)$$

n is injective by definition, so n' is injective too. For any $v \in \{0, 1\}^\eta$, as all extractions of e_1 are unique for each value of n and their length only depends on η, we have for any w, r, s

$$\mathbb{P}_{\rho_O}\{e_1(n(w, s, r), \eta, \rho_O) = v\}$$
$$= \mathbb{P}_{\rho_O}\{e_1'(n'(w, s, r), \eta, \rho_O) = v\}$$

This implies that for any input, O and O_{pref-c} will produce the same output distribution. So $\mathcal{A}^O$ and $\mathcal{A}^{O_{pref-c}}$ will produce the same distributions for any input. We conclude by remarking that $\mathcal{A}^{O_{pref-c}}$ and $\mathcal{A}^O_{pref-c}$ behaves the same by construction. □

An immediate consequence of this Lemma is that for all indistinguishability results, we can, without loss of generality, constrain attackers to only use prefixed oracle calls.

In particular it implies equivalence between indistinguishability in a computational model and indistinguishability for prefixed distinguishers in the prefixed computational model.

B.3 Alternative notions of simulatability

First, let us note that our notion of simulatability assumes that models are prefixed. As demonstrated previously this is necessary in order to get an achievable notion of simulatability. We will therefore not consider models that are not prefixed.

We may consider variants of simulatability, depending on the order of the quantifiers and sharing of randomness between simulator and distinguisher. We define simulatability as the existence of a simulator that works for all distinguishers. In other words our ordering of quantifier is:

$$\exists \mathcal{A}^O(\rho_{r_1}) \forall \mathcal{D}(\rho_{r_2})$$

In a prefixed model, we believe that switching the quantifiers lead to the same notion:

$$\exists \mathcal{A}^O(\rho_{r_1}) \forall \mathcal{D}(\rho_{r_2}) \Leftrightarrow \forall \mathcal{D}(\rho_{r_2}) \exists \mathcal{A}^O(\rho_{r_1})$$

We do not provide the proof, but the intuition is that there exists a "universal" distinguisher, namely the PPTOM $\mathcal{D}$, which performs any possible queries with uniform probability. Now, considering any other distinguisher $\mathcal{D}'$, as the simulator $\mathcal{A}^O$ for $\mathcal{D}$ has to provide the exact same distribution as the protocol for each query of $\mathcal{D}$, as $\mathcal{D}$ performs all possible queries (with very small probability), $\mathcal{A}^O$ will also be a correct simulator for $\mathcal{D}'$.

Another alternative is to allow the simulator and the distinguisher to share the same randomness. Then, $\exists \mathcal{A}^O(\rho_r) \forall \mathcal{D}(\rho_r)$ seems to provide an unachievable definition. Indeed, if the simulator is not allowed to use private randomness while the protocol is, the simulator cannot mimick the probabilistic behavior of the protocol.

The last possibility however seems to offer an alternative definition for simulatability:

$$\forall \mathcal{D}(\rho_r)\exists \mathcal{A}^O(\rho_r)$$

This seems to be a weaker definition than ours as the choices of the simulator can depend on the ones of the distinguisher. It may simplify (slightly) the proofs for the main theorem, but it would create issues for the unbounded replication as it would break uniformity of reductions (since the runtime of the simulator may now depend on the environment it is running in).

C PROOF OF THEOREM 4.3

We first generalize theorem 4.3, so that it is more easily usable. Notably, rather than restricting the channels of the protocols and the context to be disjoint, we allow to define a renaming between channels.

Theorem C.1. *Let $C[\__1, \ldots, \__n]$ be a context. Let $P_1, \ldots, P_n$, $Q_1, \ldots, Q_n$ be protocols, and let $\sigma : C(P_1, \ldots, P_n) \mapsto C$ such that $\overline{C}\|P_1\| \ldots \|P_n, \overline{C}\|Q_1\| \ldots \|Q_n, C[P_1\sigma, \ldots, P_n\sigma], C[Q_1\sigma, \ldots, Q_n\sigma]$ are protocols. Given a cryptographic library $\mathcal{M}^f$, an oracle O, with $\overline{n} \supseteq \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n, Q_1, \ldots, Q_n)$, if $\nu \overline{n}.\overline{C}$ is O-simulatable and $P_1\| \ldots \|P_n \cong_O Q_1\| \ldots \|Q_n$, then*

$$C[P_1\sigma, \ldots, P_n\sigma] \cong_O C[Q_1\sigma, \ldots, Q_n\sigma]$$

C.1 Oracle simulation

We first show that O-simulation, whose definition implies the identical distributions of two messages produced either by te simulator of by the oracle, implies the equality of distributions of message sequences produced by either the oracle or the simulator.

For any sequence of names $\overline{n}$ and parameter η, we denote $D^\eta_{\overline{n}} = \{[\![\overline{n}]\!]^\eta_{\rho_s} | \rho_s \in \{0,1\}^\omega\}$ the set of possible interpretations of $\overline{n}$. We reuse the notations of definition B.1.

Lemma C.2. *Given a cryptographic library $\mathcal{M}_f$, a sequence of names $\overline{n}$, an oracle O with support $\overline{n}$ and a protocol P, that is O-simulatable with $\mathcal{A}^O$, we have, for every $\overline{x}, \overline{y}, c, r_2, r_{\mathcal{B}} \in \{0,1\}^\star$, every $\overline{v} \in D^\eta_{\overline{n}}$, for every $m \geq 1$, for every PPTOM $\mathcal{B}^O$ (using tags prefixed by 1):*

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{\theta^1_m = \overline{x}, \phi^1_m = \overline{y} \\
&\qquad\qquad | [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}, \rho^{\mathcal{B}}_O = r_{\mathcal{B}}, \rho_{r_2} = r_2\} \\
&= \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{\theta^2_m = \overline{x}, \phi^2_m = \overline{y} \\
&\qquad\qquad | [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}, \rho^{\mathcal{B}}_O = r_{\mathcal{B}}, \rho_{r_2} = r_2\}
\end{aligned}$$

where we split ρ_O into $\rho^{\mathcal{A}}_O \uplus \rho^{\mathcal{B}}_O$ such that O called by $\mathcal{B}$ only accesses $\rho^{\mathcal{B}}_O$ and O called by $\mathcal{A}$ only accesses $\rho^{\mathcal{A}}_O$ (which is possible thanks to the distinct prefixes).

The proof of this lemma can be found in [17].

We now prove that definition B.1 implies definition 3.1, i.e that the simulatability implies that we can replace a protocol oracle by its simulator.

Lemma C.3. *Given an oracle O (with support $\overline{n}$), a cryptographic library $\mathcal{M}_f$, a sequence of names $\overline{n}$, P, Q protocols, s.t $\nu \overline{n}.P$ is O-simulatable with $\mathcal{A}^O_P$ and $\mathcal{N}(P) \cap \mathcal{N}(Q) \subseteq \overline{n}$ then, for every PPTOM D^{O,O_P,O_Q} (prefixed by 1), every η, every $\overline{v} \in D^\eta_{\overline{n}}$ and every $c \in \{0,1\}^\star$,*

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P,O_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} \\
&\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^{O,O_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}
\end{aligned}$$

The idea is to use the definition of O-simulatablity, using a PPTOM $\mathcal{B}^O$ that behaves exactly as $\mathcal{D}$ when it computes the next oracle queries from the previous answers. The difficulty is that $\mathcal{D}$ may call the oracle O_Q, while $\mathcal{B}$ has no access to this oracle. We know however that shared names are included in $\overline{n}$, whose sampling can be fixed at once (thanks to the definition of O-simulation). The other randomness in Q can be drawn by $\mathcal{B}$ from ρ_r, without changing the distribution of O_Q's replies.

Note that the Lemma also implies that:

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P,O_Q}(\rho_r, 1^\eta) = c\} \\
&\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^{O,O_Q}(\rho_r, 1^\eta) = c\}
\end{aligned}$$

Proof. Fix η and the interpretation $[\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}$.

Given $\mathcal{D}$, we let $\mathcal{D}_m$ be the machine that behaves as $\mathcal{D}$, however halting after m calls to O_P (or when $\mathcal{D}$ halts if this occurs before the mth call) and returning the last query to O_P.

We have that $\mathcal{D}_m$ first executes $\mathcal{D}_{m-1}$, then performs the oracle call $O_P(\rho_s, \theta_{m-1})$, getting u_{m-1} and performs the computation of the next oracle call v_m (if $\mathcal{D}$ makes another oracle call), updates the history $\theta_m := (v_1, \ldots, v_m)$ and returns v_m if there is one or the output of $\mathcal{D}$ otherwise. $\mathcal{D}_m[\mathcal{A}^O_P]$ first executes $\mathcal{D}_{m-1}[\mathcal{A}^O_P]$, then performs the computation $\mathcal{A}^O_P(\mathcal{M}_f, \rho_{r_1}, \theta'_{m-1})$ of u'_m, computes the next oracle call v'_m (if one is performed), updates $\theta'_m := (v'_1, \ldots, v'_m)$ and outputs either v_m of the output of $\mathcal{D}$.

We wish to use the definition of O-simulation in order to conclude. However, we cannot directly use the O-simulation, as $\mathcal{D}$ has access to an extra oracle O_Q.

Part 1

We first prove that, assuming $\mathcal{A}^O_P$ is a simulator of O_P:

$$\begin{aligned}
&\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c\} \\
&\quad = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^O(\rho_r, 1^\eta) = c\}
\end{aligned}$$

This is a straightforward consequence of lemma C.2. Writing respectively $p^1_1(c) = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c\}$ and $p^2_1(c) = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^O(\rho_r, 1^\eta) = c\}$, Using ρ_{r_1}, ρ_{r_2} as in definition 3.1, we have

$$\begin{aligned}
p^1_1(c) \quad = \quad & \sum_{r_{\mathcal{B}}, r_2} \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c \\
& \qquad\qquad | ([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\
& \qquad \times \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\
p^2_1(c) \quad = \quad & \sum_{r_{\mathcal{B}}, r_2} \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}^O_P]^O(\rho_r, 1^\eta) = c \\
& \qquad\qquad | ([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\} \\
& \qquad \times \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{[\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}, \rho^{\mathcal{B}}_O = r_{\mathcal{B}}, \rho_{r_2} = r_2\}
\end{aligned}$$

We let

$$\begin{aligned}
p^1_2(r_{\mathcal{B}}, r_2, \bar{v}, c) = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{&\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c \\
&| ([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}
\end{aligned}$$

and

$$p_2^2(r_{\mathcal{B}}, r_2, \bar{v}, c) = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}_P^O]^O(\rho_r, 1^\eta) = c \mid ([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}$$

We use definition B.1 with $\mathcal{B}^O(\rho_{r_2}, \eta, \phi)$ as the machine that simulates $\mathcal{D}_m$ for $m = |\phi|$ and using ϕ instead of querying the oracle. Let us define ϕ^i_m, θ^i_m for $i = 1, 2$ as in definition B.1. Note that with the definition of $\mathcal{D}$, $\mathcal{B}$ uses prefixes for oracle calls, disjoint from those used in $\mathcal{A}_P$, hence randomness used for oracle calls in $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Let v^i_m be the last message of θ^i_m. By definition of $\mathcal{D}$ and $\mathcal{B}$ we have $v^1_m = v_m$ and $v^2_m = v'_m$. Choosing m such that $\mathcal{D}$ makes less than m oracle calls, we have

$$p_2^i(r_{\mathcal{B}}, r_2, \bar{v}, c) = \sum_{\bar{x} \text{ s.t. } x_m = c, \bar{y}} \mathbb{P}_{\rho_s,\rho_{r_1},\rho_{r_2},\rho_O}\{\theta^i_m = \overline{x}, \phi^i_m = \overline{y} \mid ([\![\overline{n}]\!]^\eta_{\rho_s}, \rho^{\mathcal{B}}_O, \rho_{r_2}) = (\overline{v}, r_{\mathcal{B}}, r_2)\}.$$

lemma C.2 yields for all $r_{\mathcal{B}}, r_2, c$ that $p_2^2(r_{\mathcal{B}}, r_2, c) = p_2^1(r_{\mathcal{B}}, r_2, c)$, which concludes part 1.

Part 2

We now prove that:

$$\begin{aligned} &\forall \mathcal{D}.\ \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P}(\rho_r, 1^\eta) = c\} \\ &\qquad = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}_P^O]^O(\rho_r, 1^\eta) = c\} \\ &\qquad\qquad \Rightarrow \\ &\forall \mathcal{D}.\ \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}^{O,O_P,O_Q}(\rho_r, 1^\eta) = c\} \\ &\qquad = \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}_P^O]^{O,O_Q}(\rho_r, 1^\eta) = c\} \end{aligned} \tag{1}$$

We are thus going to show that, with the interpretation of $\overline{n}$ fixed, we can simulate O_Q inside some $\mathcal{D}'$ by sampling inside ρ_r instead of ρ_s. However, both computations of O_P and O_Q depend on ρ_s. This is where we need the assumptions that $\overline{n}$ contains the shared secrets between P and Q, as well as the splitting of ρ_r.

For any machine $\mathcal{M}^{O,O_Q}$, we let $[\mathcal{M}]^O_{\overline{n}}$ be the machine that executes $\mathcal{M}$, simulating O_Q for a fixed value $\overline{v}$ of $\overline{n}$. The machine samples the names appearing in Q and not in $\overline{n}$ and hard codes the interpretation of $\overline{n}$.

More precisely, we write $O_Q(\rho_s, \theta) := O_Q((\rho_{s_0}, \rho_{s_1}, \rho_{s_2}), \theta)$ where ρ_{s_0} is used for the sampling of $\overline{n}$, ρ_{s_1} for the sampling of other names in Q, and ρ_{s_2} for the reminder.

Then $[\mathcal{M}]^O_{\overline{n}}(\rho_r, 1^\eta)$ is the machine that:

- Splits ρ_r into two infinite and disjoints ρ_{sQ}, ρ_{rM} and initializes an extra tape θ to zero.
- Simulates $\mathcal{M}(\rho_{rM}, 1^\eta)$ but every time $\mathcal{M}$ calls O_Q with input u, the machine adds u to θ, and produces the output of $O_Q((\overline{v}, \rho_{rQ}, 0), \theta)$.

Such a machine runs in deterministic polynomial time (w.r.t η). For any machine $\mathcal{M}^{O,O_Q,O_P}$, we similarly define $[\mathcal{M}]^{O,O_P}_{\overline{n}}$. Now, we have that, for any c, by letting, for any X and U, $\mathbb{P}^{c,\overline{v}}_X(U) := \mathbb{P}_X\{U = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\}$:

$$\begin{aligned} &\mathbb{P}^{c,\overline{v}}_{\rho_s,\rho_r,\rho_{sQ},\rho_O}(\mathcal{D}^{O,O_P(\rho_{s_0},\rho_{s_1},\rho_{s_2}),O_Q(\rho_{s_0},\rho_{s_1},\rho_{s_2})}(\rho_r, 1^\eta)) \\ &=^1 \mathbb{P}^{c,\overline{v}}_{\rho_{s_1},\rho_{s_2},\rho_r,\rho_O}(\mathcal{D}^{O,O_P(\rho_{s_0},0,\rho_{s_2}),O_Q(\rho_{s_0},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ &=^2 \mathbb{P}^{c,\overline{v}}_{\rho_{s_1},\rho_{s_2},\rho_r,\rho_O}(\mathcal{D}^{O,O_P(\rho_{s_0},0,\rho_{s_2}),O_Q(\rho_{s_0},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ &=^3 \mathbb{P}^{c,\overline{v}}_{\rho_{s_1},\rho_{s_2},\rho_r,\rho_O}(\mathcal{D}^{O,O_P(\overline{v},0,\rho_{s_2}),O_Q(\overline{v},\rho_{s_1},0)}(\rho_r, 1^\eta)) \\ &=^4 \mathbb{P}^{c,\overline{v}}_{\rho_{sQ},\rho_s,\rho_{rD},\rho_O}(\mathcal{D}^{O,O_P(\overline{v},0,\rho_s),O_Q(\overline{v},\rho_{sQ},0)}(\rho_r, 1^\eta)) \\ &=^5 \mathbb{P}^{c,\overline{v}}_{\rho_s,\rho_r,\rho_O}([\mathcal{D}]^{O,O_P(\rho_s)}_{\overline{n}}(\rho_r, 1^\eta)) \text{ (ii)} \end{aligned}$$

Since

(1) O_Q does not access ρ_{s_2}
(2) O_P does not access ρ_{s_1}
(3) We are sampling under the assumption that $[\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}$, i.e, ρ_{s_0} is equal to $\overline{v}$.
(4) Renaming of tapes
(5) By construction

And we also have similarly that, for any c:

$$\begin{aligned} &\mathbb{P}_{\rho_s,\rho_r,\rho_O}\{\mathcal{D}[\mathcal{A}_P^O]^{O,O_Q}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} \\ &= \mathbb{P}_{\rho_s,\rho_r,\rho_O}\{[\mathcal{D}[\mathcal{A}_P^O]]^O_{\overline{v}}(\rho_r, 1^\eta) = c \mid [\![\overline{n}]\!]^\eta_{\rho_s} = \overline{v}\} \text{ (iii)} \end{aligned}$$

By applying the left-handside of (1) to $[\mathcal{D}]^{O,O_P(\rho_s)}_{\overline{n}}(\rho_r, 1^\eta)$ and $[\mathcal{D}[\mathcal{A}_P^O]_j]^O_{\overline{v}}(\rho_r, 1^\eta)$, and using (ii) and (iii), we can conclude by transitivity. We conclude the proof of the lemma by putting Part 1 and Part 2 together.

□

C.2 Proof of the Theorem

Recall that we denote $\overline{C}$ the context C, in which each $\_i$ is replaced with $\mathtt{out}(c_i, 0).\mathbf{0}$, where c_i is a channel name and 0 is a public value.

For any protocols P, Q, we denote $\mathrm{Adv}^{P \cong Q}_{\mathcal{A}}(t)$ the advantage (c.f. definition 2.4) for the PPTOM $\mathcal{A}$ (with potential access to an oracle) with execution time bounded by t

Theorem C.1. *Let $C[\_1, \ldots, \_n]$ be a context. Let $P_1, \ldots, P_n$, $Q_1, \ldots, Q_n$ be protocols, and let $\sigma : C(P_1, \ldots, P_n) \mapsto C$ such that $\overline{C}\|P_1\| \ldots \|P_n$, $\overline{C}\|Q_1\| \ldots \|Q_n$, $C[P_1\sigma, \ldots, P_n\sigma]$, $C[Q_1\sigma, \ldots, Q_n\sigma]$ are protocols. Given a cryptographic library $\mathcal{M}^f$, an oracle O, with $\overline{n} \supseteq \mathcal{N}(C) \cap \mathcal{N}(P_1, \ldots, P_n, Q_1, \ldots, Q_n)$, if $\nu\overline{n}.\overline{C}$ is O-simulatable and $P_1\| \ldots \|P_n \cong_O Q_1\| \ldots \|Q_n$, then*

$$C[P_1\sigma, \ldots, P_n\sigma] \cong_O C[Q_1\sigma, \ldots, Q_n\sigma]$$

Proof. Let $\mathcal{A}$ be an attacker against

$$C[P_1\sigma, \ldots, P_n\sigma] \cong_O C[Q_1\sigma, \ldots, Q_n\sigma].$$

We first build an attacker against

$$\overline{C}\|P_1\| \ldots \|P_n \cong_O \overline{C}\|Q_1\| \ldots \|Q_n.$$

Let us construct $\mathcal{B}^{O,O_{\overline{C}},O_{R_1},\ldots,O_{R_n}}$ with either for every i, $R_i = P_i$, or, for every i, $R_i = Q_i$. $\mathcal{B}^{O,O_{\overline{C}},O_{R_1},\ldots,O_{R_n}}$ initially sets variables $c_1, \ldots, c_n$ to 0 (intuitively, c_i records which processes have been triggered) and sets $\overline{x}$ to the empty list. It then simulates $\mathcal{A}^{O,O_{C[R_1\sigma,\ldots,R_n\sigma]}}$ but, each interaction with $O_{C[R_1\sigma,\ldots,R_n\sigma]}$ and the corresponding request (c, m) is replaced with:

- if there exist i such that $c_i = 1$ and $c \in C(R_i\sigma)$ then
  - query O_{R_i} with $(c\sigma^{-1}, m)$

- if O_{R_i} returns $\bot$, then, if contexts C_1 and C_2 are such that $C[\__1, \ldots, \__n] = C_1[\__i; C_2]$, it adds to $\overline{x}$ the channels $C(C_2)$. (This corresponds to the semantics of sequential composition: an error message disables the continuation).
- else the answer (c', m') is changed $(c'\sigma, m')$ (and the simulation goes on)

• else if $c \in C(\overline{C})$ and $c \notin \overline{x}$ then
- query $O_{\overline{C}}$ with (c, m)
- if $O_{\overline{C}}$ answers $\top$ on channel γ_i, set $c_i = 1$
- else continue with the reply of $O_{\overline{C}}$

This new attacker is basically simply handling the scheduling of the protocols, using the signals raised in the context to synchronize everything. The condition that there exist i such that $c_i = 1$ and $c \in C(R_i)$ is always satisfied by a unique i, otherwise $C[P_1\sigma, \ldots, P_n\sigma]$ or $C[Q_1\sigma, \ldots, Q_n\sigma]$ would not be well formed.

The execution time of $\mathcal{B}$ then only depends on the number of channels in C, the size of the channel substitution σ, the number of protocols n in addition to the cost of simulating $\mathcal{A}$. Hence if t is the runtime of $\mathcal{A}$, there exists p_S such that the runtime of $\mathcal{B}$ is bounded (uniformly in $C, P_1, \ldots, P_n, Q_1, \ldots Q_n$) by $p_S(n, t, |C|, |\sigma|)$:

$$\mathrm{Adv}^{C[P_1,\ldots,P_n]\cong C[Q_1,\ldots,Q_n]}_{\mathcal{A}^O}(t) \leq \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}^{O,O_{\overline{C}}}}(p_S(n,t,|C|,|\sigma|))$$

Now, with the fact that $\nu\overline{n}.\overline{C}$ is O simulatable, we have a simulator $\mathcal{A}^O_{\overline{C}}$ such that, thanks to lemma C.3, $\mathcal{B}[\mathcal{A}^O_{\overline{C}}]^{O,O_R}$ behaves exactly as $\mathcal{B}^{O,O_{\overline{C}},O_R}$.

We have, for p_C the polynomial bound on the runtime of $\mathcal{A}_{\overline{C}}$, by definition 3.1,

$$\mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}^{O,O_{\overline{C}}}}(t) \leq \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}[\mathcal{A}^O_{\overline{C}}]^O}(q(p_C(t)+t))$$

and finally,

$$\mathrm{Adv}^{C[P_1\sigma,\ldots,P_n\sigma]\cong C[Q_1\sigma,\ldots,Q_n\sigma]}_{\mathcal{A}^O}(t) \leq \mathrm{Adv}^{P_1\|\ldots\|P_n\cong Q_1\|\ldots\|Q_n}_{\mathcal{B}[\mathcal{A}^O_{\overline{C}}]^O}(q(p_C \circ p_S(n,t,|C|,|\sigma|) + p_S(n,t,|C|,|\sigma|)))$$

□

D PROOFS WITH KEY CONFIRMATIONS

D.1 Key exchange and protocol simulatability

We modify slightly the hypotheses of sections 5.2.1 and 5.2.2 to reflect the fact that we now consider the key confirmation to be part of the continuation:

(1) $\nu\overline{p}.in(x).I^1(x); P^I(x)$, $in(x).R^1(x); P^R(x)$, $in(x).I^1(x); Q^I(x)$, $in(x).R^1(x); Q^R(x)$ are $O_{P,Q}$ simulatable.

(2) $\nu\overline{p}.\ \|^{i\leq N} I^0_i(lsid^I_i, id^I);$ $\|^{i\leq N} R^0_i(lsid^R_i, id^R);$
$$\begin{array}{l}\text{if } \underset{1\leq i\leq N}{} x^I_{lsid} = lsid^R_j \text{ then}\\ \quad \mathsf{out}(\langle i,j\rangle)\\ \text{else } I^1_i(x^I); \bot\end{array}$$
$$\begin{array}{l}\text{if } \underset{1\leq i\leq N}{} x^R_{lsid} = lsid^I_j \text{ then}\\ \quad \mathsf{out}(\langle i,j\rangle)\\ \text{else } R^1_i(x^R); \bot\end{array}$$
is O_{ke} simulatable.

D.2 Security of the protocol

Compared to section 5.2.3, the continuation must be secure even in the presence of the messages produced during the key confirmation:

$$\|^{i\leq N} I^1_i(x^I); P^I_i(x^I) \| R^1_i(x^R); P^R_i(x^R) \cong_{O_r, O_k} \|^{i\leq N} I^1_i(x^I); Q^I_i(x^I) \| R^1_i(x^R); Q^R_i(x^R)$$

We can once again split this goal into a single session proof using theorem 4.7. We remark that to prove the security of the single session, we can further reduce the proof by using an oracle that may simulate I^1 and R^1, as the security of P should not depend on the messages of the key confirmation.

D.3 Security of the key exchange

We proceed in a similar way as in section 5.2.4 and we use the same notations. Let us define

$$KE^0_i[\__1, \__2] := I^0_i(lsid^I_i, id^I); \__1 | R^0_i(lsid^R_j, id^R); \__2$$

The following Hypothesis are then suitable:

(1) $\forall i \leq N, \nu lsid^I_i, id^I, lsid^R_i, id^R.$
$KE^0_i[\mathsf{out}(x^I), \mathsf{out}(x^R)] \| \mathsf{out}(\langle lsid^R_i, lsid^I_i\rangle)$ is O_T-simulatable
(2) $\overline{s}$ is disjoint of the support of $O_{P,Q}$.
(3) $KE^0[D_I, D_R] \cong_{O_{KE}, O_{P,Q}} KE^0[C_{I,R}, C_{R,I}]$

where
$$D_X := \begin{array}{l}\text{if } x^X_{lsid} \notin \overline{s}^Y \text{ then } X^1(x^X)\\ \text{else } \mathsf{out}(\langle x^X, lsid^X_0, x^X_{lsid}\rangle)\end{array}$$
$$C_{X,Y} := \begin{array}{l}\text{if } x^X_{lsid} = lsid^Y_0 \text{ then } \mathsf{out}(\langle k, lsid^X_0, x^X_{lsid}\rangle)\\ \text{else if } x^X_{lsid} \notin \overline{s}^Y \text{ then } X^1(x^X); \mathsf{out}(\bot)\\ \text{else } \mathsf{out}(\langle x^X, lsid^X_0, x^X_{lsid}\rangle)\end{array}$$

The indistinguishability expresses that, if the two singled out parties are partnered, i.e. $x^I_{lsid} = lsid^R_0$ or $x^R_{lsid} = lsid^I_0$, then we test the real-or-random of the key. Else, a party must always be partnered with some honest session, i.e $x^X_{lsid} \notin \overline{s}^Y$ never occurs. When two honest parties are partnered, but are not the singled out parties, they leak their states.

An extensive formal analysis of multi-factor authentication protocols

Charlie Jacomme, Steve Kremer

An extensive formal analysis of multi-factor authentication protocols

Charlie Jacomme
LSV & CNRS & ENS Paris-Saclay
& Inria & Université Paris-Saclay

Steve Kremer
LORIA, Inria Nancy-Grand Est
& CNRS & Université de Lorraine

***Abstract*—Passwords are still the most widespread means for authenticating users, even though they have been shown to create huge security problems. This motivated the use of additional authentication mechanisms used in so-called multi-factor authentication protocols. In this paper we define a detailed threat model for this kind of protocols: while in classical protocol analysis attackers control the communication network, we take into account that many communications are performed over TLS channels, that computers may be infected by different kinds of malwares, that attackers could perform phishing, and that humans may omit some actions. We formalize this model in the applied pi calculus and perform an extensive analysis and comparison of several widely used protocols — variants of *Google 2-step* and *FIDO's U2F*. The analysis is completely automated, generating systematically all combinations of threat scenarios for each of the protocols and using the PROVERIF tool for automated protocol analysis. Our analysis highlights weaknesses and strengths of the different protocols, and allows us to suggest several small modifications of the existing protocols which are easy to implement, yet improve their security in several threat scenarios.**

1. Introduction

Users need to authenticate to an increasing number of electronic services in everyday life: emails, agenda, bank accounts, e-commerce sites, etc. Authentication generally requires a user to present an *authenticator*, that is *"something the claimant possesses and controls (typically a cryptographic module or password) that is used to authenticate the claimant's identity"* [1]. Authenticators are often classified according to their *authentication factor*:

- what you know, e.g. a password, or a pin code;
- what you have, e.g. an access card or physical token;
- what you are, e.g. a biometric measurement.

Although these different mechanisms exist, passwords are still by far the most widely used mechanism, despite the fact that many problems with passwords were already identified in the late '70s when they were mainly used to grant login into a computer [2]. Since then, things have become worse: many people choose the same weak passwords for many purposes, and large password databases have been leaked. Studies have shown that the requirement to add special characters does not solve these problems, and the latest recommendations by NIST [3] even discourage this practice.

To palliate password weaknesses, multi-factor authentication protocols combine several authentication factors. Typically, instead of using only a login and password, the user proves possession of an additional device, such as his mobile phone, or a dedicated authentication token. Two popular protocols are *Google 2-step* [4] (which actually regroups several mechanisms) and *FIDO's U2F* [5], which is supported by many websites, including Google, Facebook, and GitHub. In (one version of) *Google 2-step*, the user receives a verification code on his phone that he must copy onto his computer, while *FIDO's U2F* requires the use of a specific USB token that must be plugged into the computer.

Our contributions. In classical protocol analysis, the attacker is supposed to control the communication network. However, the protocols we study in this paper make extensive use of TLS communications and are supposed to provide security even if some devices are infected by malware. We therefore propose a detailed threat model for multi-factor authentication protocols which takes into account many additional threats.

- Compromised passwords: our basic assumption is that the user's password has been compromised. Otherwise multi-factor authentication would not be required.
- Network control: we define a *high-level model of TLS channels* that guarantees confidentiality and authentication of messages and additionally ensures, through inclusion of session ids, that messages of different TLS sessions cannot be mixed. Nevertheless, we allow the attacker to delay or block messages. Our model also contains a notion of *fingerprint* that is used in some protocols to identify machines, and we may give the adversary the power to spoof such fingerprints.
- Compromised platforms: we give a structured and fine-grained *model for malwares*. We take an abstract view of a system as a set of input and output interfaces, on which an adversary may have read or write access, depending on the particular malware.
- Human aspects: we take into account that most of these protocols require some interaction with the *human user*. We model that humans may not correctly perform these steps. Moreover, we model that a human may be victim

of *phishing*, or *pharming*, and hence willing to connect to and enter his credentials on a malicious website.
- Trust this computer mechanism: to increase usability, several websites, including Google and Facebook, offer the possibility to *trust* a given machine, so that the use of a second factor becomes unnecessary on these machines. We add this trust mechanism to our model.

We model these threat scenarios in the applied pi calculus and use the PROVERIF tool to analyse several variants of the *Google 2-step* and *FIDO's U2F* protocols. The analysis is completely automated, generating systematically all combinations of threat scenarios for each of the protocols and using the PROVERIF tool for automated protocol analysis. Even though we eliminate threat scenarios as soon as results are implied by weaker scenarios, the analysis required over 6 000 calls to PROVERIF. Our analysis results in a detailed comparison of the protocols which highlights their respective weaknesses and strengths. It also allows us to suggest several small modifications of the existing protocols which are easy to implement, yet improve their security in several threat scenarios. In particular, the existing mechanisms do not authenticate the *action* that is performed, e.g., a simple login may be substituted by a login enabling the *trust this computer* mechanism, or a password reset.

Related work. Bonneau et al. [6] propose a detailed framework to classify and compare web authentication protocols. They use it for an extensive analysis and compare many solutions for authentication. While the scope of their work is much broader, taking into account more protocols, as well as usability issues, our security analysis of a more specific set of protocols is more fine-grained. Moreover, our security analysis is grounded in a formal model using automated analysis techniques.

Some other attempts to automatically analyse multi-factor authentication protocols were made, including for instance the analysis of *FIDO's U2F* [7], the *Yubikey One Time Password* [8], [9] and the *Secure Call Authorization* protocols [10]. However, those analyses do not study resistance to malware, nor do they capture precisely TLS channel behaviour or fingerprints. Basin et al. [11] studied how human errors could decrease security. Their model is more evolved than ours on this aspect. However, we consider more elaborate malwares and also check for a stronger authentication property: an attack where both a honest user and an attacker try to log into the honest user's account but only the attacker succeeds is not captured in [11], as they simply check that every successful login was proceeded by an attempt from the corresponding user to login. In the same vein, [12] studies minimal topologies to establish secure channels between humans and servers.

2. Multi-factor authentication protocols

In this section we briefly present the two, widely used, multi-factor authentication protocols that we study in this paper: (several variants of) *Google 2-step* and *FIDO's U2F*.

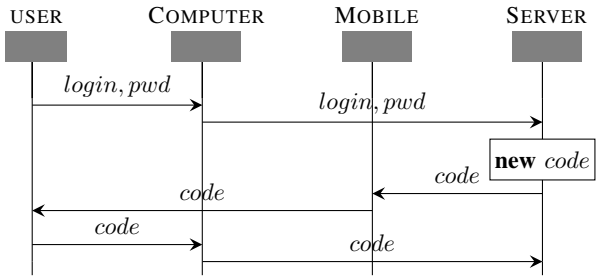

Figure 1: G2V protocol

2.1. *Google 2-step*

To improve security of user logins, Google proposes a two factor authentication mechanism called *Google 2-step* [4]. If enabled, a user may use his phone to confirm the login. On their website Google recalls several reasons why password-only authentication is not sufficient and states that *"2-Step Verification can help keep bad guys out, even if they have your password"*.

Google 2-step proposes several variants. The default mechanism sends to the user, by SMS, a verification code to be entered into his computer. An alternative is the "One-Tap" version, where the user simply presses a Yes button in a pop up on his phone. The second version avoids to copy a code and is expected to improve the usability of the mechanism. This raises an interesting question about the trade-off between security and ease of use. We also present a newer version of "One-Tap" that we dubbed "Double-Tap".

2.1.1. *Google 2-step* with verification codes - G2V. In Figure 1 we depict the different steps of the protocol. All communications between the user's computer and the server are protected by TLS. The user enters his login and password into his computer, which forwards the information to the server. Upon receiving login and password, the server checks them. In case of success, the server generates a fresh 6 digits code, and sends an SMS of the form *"G-****** is your Google verification code"* to the user's mobile phone. The user then copies the code to his computer, which sends it to the server. If the correct code is received login is granted.

2.1.2. *Google 2-step* with One-Tap - G2OT. In Figure 2 we present the One-Tap version of *Google 2-step*. The protocol starts as the verification code variant with the transmission of the login credentials to the server. The server then creates a fresh random *token* that is sent to the user's mobile phone. Unlike in the previous version, the communication between the server and the phone is over a TLS channel rather than by SMS. The phone displays a pop up to the user who can then confirm the action or abort it, by choosing "Yes" or "No" respectively. In case of confirmation the phone returns the token and login is granted. Note that in its most basic version, the user only answers a yes/no question. Google announced in February 2017 [13] that the pop up would also contain in the future a fingerprint of the

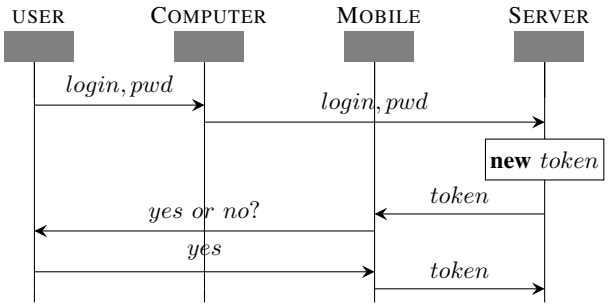

Figure 2: G2OT protocol

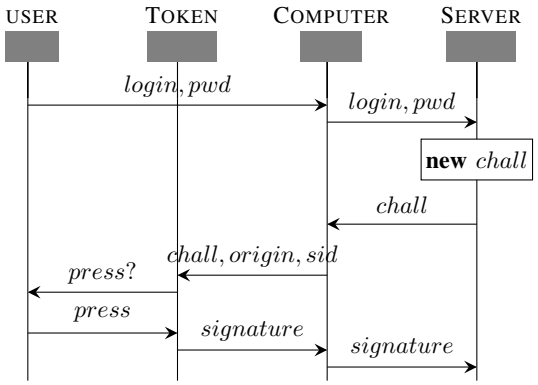

Figure 3: U2F protocol

computer, including information such as IP address, location and computer model. However this new version has yet to be implemented on some of the smartphones we used for tests. In the following we will analyse both versions, with (G2OT$^{\text{fpr}}$) and without (G2OT) the fingerprint.

2.1.3. *Google 2-step* with Double-Tap - G2DT$^{\text{fpr}}$. The issue with One-Tap compared to the code version is that the user is likely to simply press "Yes" without reading any displayed information. To mitigate this issue, Google sometimes uses a version which we call Double-Tap. It is not documented, but we saw it at work in practice. The first step is the One-Tap protocol previously presented, with the display of the fingerprint. It is then followed by a second step, where a two digit number is displayed on the user's computer screen, and the same number is displayed on the user phone along two other random numbers. The user is then asked to select on his phone the number displayed on his computer. This mimics the behaviour of a verification code displayed on the computer and that the user should enter on his phone, but with the benefits of greater simplicity and ease of use.

2.2. *FIDO's Universal 2nd Factor* - U2F

FIDO is an alliance which aims at providing standards for secure authentication. Among their proposed solutions is the Universal 2nd Factor (U2F) protocol [5]. We here concentrate on the version using a USB token as the second factor. The U2F protocol relies on a token able to securely generate and store secret and public keys, and perform cryptographic operations using these keys. Moreover, the token has a button that a user must press to confirm a transaction. To enable second-factor authentication for a website, the token generates a key pair and the public key is registered on the server. This operation is performed once, and the token can then be used for authenticating; the steps of the authentication protocol are presented in Figure 3. First, the computer forwards the user's login and password to the server. Then, the server generates a challenge which is sent to the user's computer. Upon reception, the browser generates a payload containing the url of the server, the challenge and the identifier of the current TLS session to be signed by the token. The user confirms the transaction by pressing the token button. The signature is then forwarded to the server for verification.

2.3. Disabling the second factor on trusted devices

When designing an authentication protocol, as also emphasized in [6], a key requirement should be usability. On a user's main computer, used on a daily basis, it may not be necessary to use a second factor: for instance, using a second factor each time a user pops his emails on his main laptop would be very cumbersome. This is why several providers, including Google and Facebook, propose to *trust* specific computers and disable the second factor authentication on these particular machines. This is done by checking a *"Trust this computer"* option when initiating a two-factor authenticated login. Technically, the computer will be identified by a cookie and its *fingerprint*. A fingerprint typically includes information about the user's IP address, inferred location, OS or browsers version, etc. As these elements will obviously change over time, in practice, a distance between fingerprints is evaluated, and if the fingerprint is too far from the expected one, the second factor authentication will be required. To the best of our knowledge, this feature is not documented and the full mechanism has not been studied previously even though it may lead to security issues. To capture such security issues we will include the *"Trust this computer"* mechanism in our analysis.

3. Threat model

In order to conduct an in depth analysis of multi-factor authentication protocols, we consider different threat models, types of attacks and corresponding attacker capabilities. We will consider a Dolev-Yao attacker [14] that controls any compromised parts and, classically, the network. However, many of the protocols we study use channels protected by TLS. The attacker may block a message, even if he cannot read or write on such channels. Moreover, as we are studying multi-factor authentication protocols, in order to assess additional protection offered by these protocols, we are interested in the case where the user's password has been

compromised. Therefore, the most basic threat scenario we consider is the one where the attacker has (partial) control over the network, and knows the users' passwords.

There are however several ways the attacker can gain more power. Our aim is to present a detailed threat model, reflecting different attacker levels that may have more or less control over the user's computer, the network, or even over the user itself. Those levels aim at capturing the attacker capabilities that are necessary for a given attack.

3.1. Malware based scenarios

The first range of scenarios covers malwares that give an attacker control over parts of a user's device.

3.1.1. Systems as interfaces. To give a principled model of malwares and what parts of a system a malware may control, we take an abstract view of a system as a set of interfaces on which the system receives inputs and sends outputs. Some interfaces may only be used for inputs, while other interfaces may be used for outputs, or both. For example the keyboard is an input interface, the display is an output interface, and the network is an input and output interface. Compromise of part of the system can then be formalized by giving an attacker read or write access to a given interface. On a secure system, the attacker has neither read nor write access on any interface. Conversely, on a fully compromised system the attacker has read-write access on all interfaces.

More formally we consider that for each interface the attacker may have

- no access (NA),
- read-only access (RO),
- write-only access (WO), or
- read-write access (RW).

We may specify many different levels of malware by specifying for every interface two access levels, one for inputs and one for outputs on the interface. Obviously, for a given interface not all combinations need to be considered: a read-write access will yield a stronger threat model than read-only access, write-only or no access.

We will suppose in this paper that it is harder to control the outputs of an interface than its inputs: therefore a given access level to the outputs will imply the same access level on the interface inputs. Although not a limitation of our model, this choice is motivated by practical considerations. Running for instance a keylogger does not require specific rights, because the keyboard data is completely unprotected in the OS. FIDO devices are identified by the OS as a keyboard (at least on linux). However, reading data sent by an application to some USB device, may require to corrupt the driver (or in the case of linux enable the "usbmon" module) which requires specific privileges. Similarly, we suppose that having write access implies having read access. This yields for each interface 5 levels, that can be organized as a lattice depicted in Figure 4.

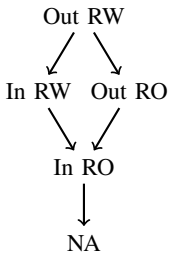

Figure 4: Access

3.1.2. Malware on a computer. For a computer, we will consider three interfaces:

- the USB interface, capturing for instance the keyboard, or a U2F USB key, with all possible types of access;
- the display, the computer screen, with only outputs interfaces;
- the TLS interface, capturing the network communications, but by always assuming that the attacker as the same level of control over inputs and outputs.

We can succinctly describe a malware on a computer by giving for each interface inputs and output the attacker access, and we will the use the notation : $\mathcal{M}^{\mathsf{interface}}_{\mathsf{in}:acc1,out:acc2}$, where $\mathsf{interface}$ might be tls, usb or dis, and $acc1$ and $acc2$ might be $\mathcal{RO}$ or $\mathcal{RW}$.

By convention, if we do not specify any access level, it means that the attacker has no access. A key logger is for instance denoted with $\mathcal{M}^{\mathsf{usb}}_{\mathsf{in}:\mathcal{RO}}$. If the access level is the same both for the inputs and the outputs, as we always assume for TLS, we may write $\mathcal{M}^{\mathsf{tls}}_{\mathsf{io}:\mathcal{RW}}$, thus capturing the fact that the attacker may have full control over the user browser, or that he might have exploited a TLS vulnerability.

3.1.3. Malware on a phone. For a mobile phone, the type of interface may vary from protocols, with for instance SMS inputs or TLS inputs. To simplify, we will only consider a phone to have one input and one output interface. We thus only consider a generic device interface called dev, with all possible access level. $\mathcal{M}^{\mathsf{dev}}_{\mathsf{in}:\mathcal{RO}}$ then corresponds for instance to the attacker having broken the SMS encryption, or to some malware on the phone listening to inputs.

3.2. Fingerprint Spoofing

Whenever a user browses the Internet, he provides information about himself, called his *fingerprint*. Those elements will be very useful later on for additional checks in our protocols, and as we mentioned Google is adding this kind of details to their One-Tap protocol. However, in some cases the attacker might be able to obtain the same fingerprint as a given user. While some elements, such as the OS version, are rather easy to spoof, it is more complicated to spoof the IP address and inferred location. It is nevertheless possible if an attacker either completely controls the network the user connects on, or because he is connected to the same Wifi, or works in the same office.

3.3. Human errors

The attacker may also exploit vulnerabilities that rely on the user not or wrongly performing some actions, or preferring to ignore security warnings. The assumption that users may not behave in the expected way seems reasonable given that most users are not trained in computer security, and their goal is generally to access a service rather than performing security related actions.

3.3.1. Phishing. In our model, we capture that users may be victim of *phishing* attempts, i.e. willing to authenticate on a malicious website. For instance, an untrained, naive user may be willing to click on a link in an email which redirects to a fake web site. While a phishing attack through an e-mail may not fool a trained user, even an experienced user may be a victim, for instance if he connects to an attacker Wifi hotspot which asks to login to a website in order to obtain free Wifi. Therefore, when we consider the *phishing threat scenario* we allow the attacker to choose with whom the user will initiate the protocol. We consider phishing as one of the simplest attacks to mount, and protocols should effectively protect users against it.

However, even though we consider that users might be victim of phishing, we suppose that they are careful enough to avoid it when performing the most sensitive operations: these operations include the registration of the U2F key, and logging for the first time on a computer they wish to trust later on. Indeed, if we were to allow phishing to be performed during those steps, no security guarantees could ever be achieved as the use of a second factor authentication requires a trusted setup.

3.3.2. No compare. A protocol may submit to the user a fingerprint and expect him to continue the protocol only if the fingerprint corresponds to his own. When given a fingerprint and a confirmation button, some users may confirm without reading the displayed information. Thus, when considering the *no compare* scenario, we assume that the user does not compare any value given to him and always answers yes.

3.4. Threat scenarios considered

In our analysis we consider all the possible combinations of the previously presented scenarios. This yields a fine-grained threat model allowing for a detailed comparison of the different protocols, allowing us to identify the strengths and weaknesses of each protocol, by showing which threats are mitigated by which mechanisms.

By considering those possibilities, we capture many real life scenarios. For instance, when connecting to a Wifi hotspot in an hotel or a train station, the Wifi might be controlled by the attacker, and we would have fingerprint spoof and phishing, because the attacker can have full control over the network, and thus use the provided IP address or redirect a user to a fake website.

If we try to connect on some untrusted computer, for instance the computer of a coworker, it may contain a rather basic malware, for instance a keylogger ($\mathcal{M}^{\mathsf{usb}}_{\mathsf{in}:\mathcal{RO}}$). However, if we connect on a computer shared by many people at some place, for instance at a cybercaffé, there could be a very strong malware controlling the display of the computer ($\mathcal{M}^{\mathsf{dis}}_{\mathsf{out}:\mathcal{RW}}$) or controlling any TLS connection on this computer ($\mathcal{M}^{\mathsf{tls}}_{\mathsf{io}:rw}$). Moreover, the network in this unknown place might also be compromised, and we may have some other scenarios combined with the malware, such as phishing (PH) or fingerprint spoofing (FS).

Our different scenarios, provide a high level of granularity going from no attacker power at all to complete control over both the network and the platforms. Our threat model abstracts how the attacker gained this power. Thus, the scenarios we consider will contain at some point all the possible attacks, without the need to specify how they may be performed. For instance, a side channel attack such as Meltdown [15] or Spectre [16] may allow the attacker to read the memory of the user computer: this is captured by giving read-only access to all the interfaces of the computer ($\mathcal{M}^{\mathsf{usb}}_{\mathsf{in}:\mathcal{RO}}\ \mathcal{M}^{\mathsf{tls}}_{\mathsf{in}:\mathcal{RO}}\ \mathcal{M}^{\mathsf{dis}}_{\mathsf{in}:\mathcal{RO}}$). Another example is pharming, where the attacker can "lie" about the url that is displayed to the user. This may happen either because of a malware that edits the *hosts* file (on a UNIX system), or by performing DNS ID Spoofing or DNS Cache Poisoning. All of these scenarios are simply captured as $\mathcal{M}^{\mathsf{tls}}_{\mathsf{io}:\mathcal{RW}}$.

4. The formal model

For our formal analysis, we model protocols in a dialect of the applied pi calculus [17], [18] that is used as input language by the PROVERIF tool [19] which we use to automate the analysis. We will only give a brief, informal overview here, which should be sufficient to explain our modelling of TLS sessions and threat scenarios. We refer the reader to [19] for additional details about the formal semantics.

4.1. The applied-pi calculus and PROVERIF

In the applied pi-calculus, protocols are modelled as concurrent processes that communicate messages built from a typed term algebra. The attacker controls the execution of the processes and can make computations over known terms. The grammar is given in Figure 5.

Atomic terms are either names $a, b, c, n, \ldots$, or variables $x, y, z, \ldots$, each declared with a type. Pre-defined types include channel, bool and bitstring, but a user may define additional types. We note that the type system is only a convenient way to avoid errors in the specification; it does not limit the attacker, and types are basically ignored in the semantics. We suppose a set of function symbols, split into *constructors* and *destructors*. Each function symbol has an *arity*, defining the number and types of the arguments, as well as the type of the resulting term. Terms are built by applying constructors to other terms. Destructors are defined by one or several rewriting rules and model the properties of function symbols. For example, we can model digital signatures as follows. Suppose that `pkey` and `skey` are user defined types, modelling public and secret keys. Then we can define the function constructors

$$\texttt{pk}(\texttt{skey}) : \texttt{pkey} \text{ and } \texttt{sign}(\texttt{bitstring}, \texttt{skey}) : \texttt{bitstring}$$

as well as the destructor `checksign` by the following rewrite rule

$$\texttt{checksign}(\texttt{sign}(m, k), \texttt{pk}(k)) \rightarrow m$$

While constructors are used to build terms, application of destructors generalizes terms to expressions. Expressions may *fail* when a destructor is applied and the expression cannot reduce to a term by applying the rewrite rules defining the destructors. Additionally, one may declare *equations* on terms, which define a congruence relation on terms that are considered equal. Hence, an alternative way of specifying digital signatures would be to declare `checksign` as a constructor together with the equation

$$\texttt{checksign}(\texttt{sign}(m, k), \texttt{pk}(k)) = m$$

In contrast to the previous modelling, $\texttt{checksign}(t_1, t_2)$ is a valid term for any t_1, t_2 and the evaluation of this term will not fail. Moreover, one can define *private* names and function symbols that may not be used by the attacker.

The protocols themselves are modelled by processes. **0** is the terminal process that does nothing. $P \mid Q$ runs processes P and Q in parallel, and $!P$ allows to spawn an unbounded number of copies of P to be run in parallel. **new** $n : T$ declares a fresh name of type T; this construct is useful to generate fresh, secret keys and nonces. $\textbf{in}(M, x : T)$ inputs a term that will be bound to a variable of type T on channel M and $\textbf{out}(M, N)$ outputs the term N on channel M. If the channel name is known to (or can be computed by) the adversary, the channel is under the adversary's control: any input message may come from the adversary, and any output is added to the adversary's knowledge. On the contrary, if the channel is private, then the adversary can neither read from nor write on this channel. The conditional **if** $E_1 = E_2$ **then** P **else** Q checks whether two expressions successfully evaluate to equal terms and executes P, or Q if at least one of the expressions failed or the two expressions yield different terms. Finally processes can be annotated by an event $e(M)$ where e is a user defined event. Events do not influence the process execution and serve merely as an annotation for specifying properties.

As an example consider the processes defined in Figure 6. A server process S signs a freshly generated random bitstring rnd and sends it to a user process U. U raises the event $Accept(rnd)$ if the signature is valid. The main process then declares that sk is a fresh secret key and executes an unbounded number of copies of S and U in parallel.

In this paper we are interested in verifying *authentication properties*. We model them, as usual, as correspondence properties of the form

$$e_1(t_1, \ldots, t_n) \Longrightarrow e_2(u_1, \ldots, u_m)$$

Such a property holds, if for any execution, any occurrence of an instance of $e_1(t_1, \ldots, t_n)$ is preceded by the corresponding instance of $e_2(u_1, \ldots, u_m)$. Considering the example of Figure 6, we model the property that any accepted session with a given random was actually initiated with the same random as

$$Accept(x) \Longrightarrow Init(x)$$

This property is indeed satisfied. However, it may be too weak as it does not capture replay attacks. We may have twice the event $Accept(rnd)$ (for the same value rnd) while this session was only initiated once. To capture such replay attack we may use *injective* correspondence properties

$$e_1(t_1, \ldots, t_n) \Longrightarrow_{\mathsf{inj}} e_2(u_1, \ldots, u_m)$$

that require that any occurrence of e_1 is matched by a different preceding occurrence of e_2.

4.2. Modelling TLS communications

Most web protocols rely on TLS to ensure the secrecy of the data exchanged between a client and a server. In order to formally analyse online authentication protocols, we thus need to model TLS sessions and corresponding attacker capabilities. A possibility would of course be to precisely model the actual TLS protocol and use this model in our protocol analysis. This would however yield an extremely complex model, which would be difficult to analyse. Therefore, for this paper, we opt to model TLS at a higher level of abstraction.

In essence we model that TLS provides

- confidentiality of the communications between the client and the server, unless one of them has been compromised by the adversary;
- a session identifier that links all messages of a given session, avoiding mixing messages between different sessions.

We model this in the applied pi calculus as follows:

- we define a private function $\texttt{tls}(\texttt{id}, \texttt{id}) : \texttt{channel}$ where `id` is a user defined type of identities, and use the channel $\texttt{tls}(c, s)$ for communications between client c and server s;
- we define a *TLS manager* process that given as inputs two identities id_1, id_2 and outputs on a public channel the channel name $\texttt{tls}(id_1, id_2)$, if either id_1 or id_2 are compromised;
- we generate a fresh name of type `sid` for each TLS connection and use it as a session identifier, concatenating it to each message, and checking equality of this identifier at each reception in a same session.

However, even if the communication is protected by TLS, we suppose that the adversary can block or delay communications. As communications over private channels are synchronous we rewrite each process of the form $\textbf{out}(\texttt{tls}(c, s), M).P$ into a process $\textbf{out}(\texttt{tls}(c, s), M) | P$. This ensures that the communications on TLS channels are indeed *asynchronous*.

4.3. Modelling threat models

We will now present how we model the different scenarios discussed in Section 3 in the applied pi calculus.

4.3.1. Malware. As discussed in Section 3.1.1, we view a system as a set of interfaces. By default, these interfaces are defined as private channels. Let a be a public channel. A malware providing read-only access to an interface ch is

$M, N ::=$		terms	$P, Q ::=$	$\mathbf{0}$	null process
	a, b, c, k, m, n, s	name		$P \mid Q$	parallel
	x, y, z	variable		$!P$	replication
	$\mathtt{f}(M_1, \ldots, M_n)$	constructor application		**new** $n : T.\ P$	name restriction
				in$(M, x : T).P$	message input
$E ::=$		expressions		**out**$(M, N).P$	message output
	M	name		**if** $E_1 = E_2$ **then** P **else** Q	conditional
	$\mathtt{h}(E_1, \ldots, E_n)$	function application		**event** $e(M).P$	event e
	fail	failure			

Figure 5: Terms and processes

$S(sk : \mathtt{skey}) \hat{=}$
  new $rnd : \mathtt{bitstring}$.
  event $\mathtt{Init}(rnd)$.
  out$(a, (\mathtt{sign}(rnd, sk), rnd))$

$U(pk : \mathtt{pkey}) \hat{=}$
  in$(a, (sig : \mathtt{bitstring}, rnd : \mathtt{bitstring}))$.
  if $\mathtt{checksign}(sig, pk) = rnd$ **then**
    event $\mathtt{Accept}(rnd)$.

new $sk : \mathtt{skey};\ !S(sk) | !U(\mathtt{pk}(sk))$

Figure 6: Process example

modelled by rewriting processes of the form **in**$(ch, x).P$ into processes of the form **in**(ch, x).**out**$(a, x).P$, respectively **out**$(ch, M).P$ into **out**(a, M).**out**$(ch, M).P$, depending on whether inputs or outputs are compromised. Read-write access is simply modelled by revealing the channel name ch, which gives full control over this channel to the adversary.

4.3.2. Human errors - No compare. Our model contains dedicated processes that represent the expected actions of a human, e.g., initiating a login by typing on the keyboard, or copying a code that he may receive through the display interface of its computer or its phone. A user is also assumed to perform checks, such as verifying the correctness of a fingerprint or comparing two random values, one displayed on the computer and one on the phone. In the *No Compare* scenario we suppose that a human does not perform these checks and simply remove them.

4.3.3. Human errors - Phishing. In our model of TLS we simply represent a url by the server identity idS, provided by the human user. This initiates a communication between the user's computer, with identifier idC, and the server over the channel $\mathtt{tls}(idC, idS)$. This models that the server URL is provided by the user and may be that of a malicious server, which his machine is then connecting to. We let the adversary provide the server identity idA to the user in order to model a basic *phishing* mechanism. We distinguish two cases: a trained user will check that $idA = idS$, where idS is the correct server, while an untrained user will omit this check and connect to the malicious server.

4.3.4. Fingerprint and spoofing. As discussed before, when browsing, one may extract informations about a user's location, computer, browser and OS version, etc. This fingerprint may be used as an additional factor for identification, and can also be transmitted to a user for verification of its accuracy. We model this fingerprint by adding a function $\mathtt{fpr}(id) : \mathtt{fingerprint}$ which takes an identity and returns its corresponding fingerprint. Given that all network communications are performed over a TLS channel $\mathtt{tls}(c, s)$ the server s can simply extract the fingerprint $\mathtt{fpr}(c)$. However, in some cases we want to give the attacker the possibility to spoof the fingerprint, e.g., if the attacker controls the user's local network. In these cases we declare an additional function $\mathtt{spoof}_{\mathtt{fpr}}(\mathtt{fingerprint}) : \mathtt{id}$ and the equation

$$\mathtt{fpr}(\mathtt{spoof}_{\mathtt{fpr}}(\mathtt{fpr}(c))) = \mathtt{fpr}(c)$$

which allows the attacker to initiate a communication on a channel $\mathtt{tls}(\mathtt{spoof}_{\mathtt{fpr}}(\mathtt{fpr}(idC)), s)$ using a fingerprint that is identical to $\mathtt{fpr}(idC)$.

5. Analysis and Comparison

In this section we use the formal framework to analyze several multi-factor authentication protocols. The analysis is completely automated using the PROVERIF tool. All scripts and source files used for these analyses are available at [20].

5.1. Properties and methodology

5.1.1. Properties. We will concentrate on authentication properties and consider that a user may perform 3 different actions:

- an *untrusted login*: the user performs a login on an untrusted computer, i.e., without selecting the *"trust this computer"* option, using second-factor authentication;
- a *trusted login*: the user performs an initial login on a trusted computer, and selects the *"trust this computer"* option, using second-factor authentication;
- a *cookie login*: the user performs a login on a computer that he previously trusted, using his password, but no second factor, and identifying through a cookie and fingerprint.

For each of these actions we check that whenever a login happens, the corresponding login was requested by the user. We therefore define three pairs of events

$$(init_x(\texttt{id}), accept_x(\texttt{id})) \qquad x \in \{u, t, c\}$$

The $init_x(\texttt{id})$ events are added to the process modelling the human user, in order to capture the user's intention to perform the login action. The $accept_x$ events are added to the server process. The three properties are then modelled as three injective correspondence properties:

$$accept_x(id) \Longrightarrow_{\mathsf{inj}} init_x(id) \qquad x \in \{u, t, c\}$$

When the three properties hold, we have that every login of some kind accepted by the server for a given computer is matching exactly one login of the same kind initiated by the user on the same computer.

5.1.2. Methodology. For every protocol, we model the three different types of login, and then check using PROVERIF whether each security property holds for all possible (combinations of) threat scenarios presented in Section 3. As we consider trusted and untrusted login, we provide the user with two platforms: a trusted platform on which the user will try to perform trusted logins, and an untrusted platform for untrusted logins. We will thus extend the notation for malwares presented in 3.1.2 by prefixing the interface with t if the interface belongs to the trusted computer, and u if it belongs to the untrusted computer, with for instance $\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{io}:\mathcal{RO}}$ for a keylogger on the untrusted computer. A scenario is described by a list of considered threats that may contain

- phishing (PH);
- fingerprint spoofing (FS);
- no comparisons by the user (NC);
- the malwares that may be present on the trusted and untrusted platform.

For instance, "PH FS $\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{io}:\mathcal{RW}}$" corresponds to the scenario where the attacker can perform phishing, fingerprint spoofing, and has read-write access to usb devices of the trusted computer, and "NC $\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RW}}$ $\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{io}:\mathcal{RW}}$ $\mathcal{M}^{\mathsf{u-dis}}_{\mathsf{io}:\mathcal{RW}}$" corresponds to a human that does not perform comparisons and full attacker control (read-write access to TLS, USB and display interfaces) on the untrusted device.

We use a script to generate the files corresponding to all scenarios for each protocol and launch the PROVERIF tool on the generated files. In total we generated 6 172 scenarios that are analysed by PROVERIF in 8 minutes on a computing server with twelve Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 50Go of RAM. We note that we do not generate threat scenarios whenever properties are already falsified for a weaker attacker (considering less threats or weaker malware). The script generates automatically the result tables, displaying only results for minimal threat scenarios that provide attacks, and maximal threat scenarios for which properties are guaranteed. In the following sections we present partial tables with results for particular protocols. Full results for all protocols are given in Tables 7 and 8 in Appendix.

Threat Scenarios				G2V	G2OT	G2OT$^{\mathsf{fpr}}$
				✓	✗	✓
PH				✗	✗	✓
		NC		✓	✗	✗
	FS			✓	✗	✗
PH		NC		✗	✗	✗
PH	FS			✗	✗	✗
			$\mathcal{M}^{\mathsf{dev}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✓
			$\mathcal{M}^{\mathsf{t-dis}}_{\mathsf{io}:\mathcal{RO}}$	✓	✗	✓
			$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✓
			$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✓
		NC	$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✗
		NC	$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✗
			$\mathcal{M}^{\mathsf{dev}}_{\mathsf{in}:\mathcal{RW}}$	✗	✗	✗
			$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RW}}$	✗	✗	✗✓✗
			$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RW}}$	✗	✗	✓✓✗
	FS		$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✗
	FS		$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✗
			$\mathcal{M}^{\mathsf{u-dis}}_{\mathsf{io}:\mathcal{RO}}$	✓	✗	✓
			$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✓
			$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✓
		NC	$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✗
		NC	$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✗
			$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RW}}$	✗	✗	✓✗✗
			$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RW}}$	✗	✗	✓✗✓
	FS		$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$	✗	✗	✗
	FS		$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$	✗	✗	✗

Table 1: Analysis of the basic *Google 2-step* protocols

The result tables use the following notations:

- results are displayed as a triple $u\,t\,c$ where u, t, c are each ✗ (violated) or ✓ (satisfied) and give the status of the authentication property for $u$ntrusted login, $t$rusted login and $c$ookie login for the given threat scenario;
- ✖ and ✔ are shortcuts for ✗✗✗ and ✓✓✓;
- signs are greyed when they are implied by other results, i.e., the attack existed for a weaker threat model, or the property is satisfied for a stronger adversary;
- we sometimes use blue symbols to emphasize differences when comparing protocols.

Even if PROVERIF can sometimes return false attacks, we remark that any ✗ corresponds to an actual attack where PROVERIF was able to reconstruct the attack trace.

5.2. *Google 2-step*: Verification Code and One-Tap

In this section we report on the analysis of the currently available *Google 2-step* protocols: the verification code (G2V, described in Section 2.1.1), the One-Tap (G2OT, described in Section 2.1.2) with and without fingerprint, and the Double-Tap (G2DT$^{\mathsf{fpr}}$, described in Section 2.1.3). The results are summarized in Tables 1 and 2.

5.2.1. G2V. In the G2V protocol the user must copy a code received on his phone to his computer to validate the login. We first show that G2V is indeed secure when only the password of the user was revealed to the attacker: as long as the attacker cannot obtain the code, the protocol remains secure. If the attacker obtains the code, either using

a keylogger ($\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$), or by reading the SMS interface ($\mathcal{M}^{\mathsf{dev}}_{\mathsf{in}:\mathcal{RO}}$), or any other read access on an interface on which the code is transmitted, the attacker can use this code to validate his own session.

We have tested on the Google website that a code generated for a login request can indeed be used (once) for any other login, demonstrating that such attacks are indeed feasible. Interestingly, this also shows that in the actual implementation, the verification code is not linked to the TLS session. This may be useful as it allows to print in advance a set of codes, e.g., if no SMS access is available. In theory, linking the code to a session does not improve security, as the code of the attacker session will also be sent to the user's phone and could then be recovered. In practice, if the code is linked, an attack can be produced only if the attacker's code is received first, i.e., if the attacker can login just before or after the user.

We remark that the results for G2V are also valid for another protocol, *Google Authenticator*, in which the phone and the server shares a secret key, and use it to derive from the current time a one time password. In all the scenarios where the SMS channel is secure, G2V can be seen as a modelling of *Google Authenticator* where the OTP is a random value "magically" shared by the phone and the server.

5.2.2. G2OT. In the G2OT protocol a user simply confirms the login by pressing a yes/no button on his phone. We first consider the version that does not display the fingerprint, and which is still in use. Our automated analysis reports a vulnerability even if only the password has been stolen. In this protocol, the client is informed when a second, concurrent login is requested and the client aborts. However, if the attacker can block, or delay network messages, a race condition can be exploited to have the client tap yes and confirm the attacker's login. We have been able to reproduce this attack in practice and describe it in more detail in Appendix A. While the attack is in our most basic threat mode, it nevertheless requires that the attacker can detect a login attempt from the user, and can block network messages.

5.2.3. G2OT$^{\mathsf{fpr}}$. We provide in the third column of Table 1 the analysis of G2OT$^{\mathsf{fpr}}$. To highlight the benefits of the fingerprint, we color additionally satisfied properties in blue. In many read only scenarios ($\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$, $\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$,$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$, $\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$), and even in case of a phishing attempt, the user sees the attacker's fingerprint on his phone and does not confirm. However, if the user does not check the values (NC) or if the attacker can spoof the fingerprint (FS), G2OT$^{\mathsf{fpr}}$ simply degrades to G2OT and becomes insecure. Some attacks may be performed on the cookie login, for instance for scenarios $\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RW}}$ or $\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{io}:\mathcal{RW}}$, as the attacker may initiate a login from the user's computer without the user having any knowledge of it, and then use it as a kind of proxy.

Because of the verification code, in scenarios of FS or NC, G2V provides better guarantees than G2OT$^{\mathsf{fpr}}$. It is however interesting to note that G2OT$^{\mathsf{fpr}}$ resists to read only access on the device as there is no code to be leaked to the attacker. One may argue that an SMS channel provides less confidentiality than a TLS channel, i.e., the read-access on the SMS channel may be easier to obtain in practice. Indeed, SMS communications between the cellphone and the relay can be made with weaker encryption (A5/0 and A5/2) than TLS, and the SMS message will also travel over TLS between the relay and the provider's servers. While this argument is in favour of G2OT$^{\mathsf{fpr}}$, one may also argue that G2V has better resistance to user inattention, as a user needs to actively copy a code.

5.2.4. G2DT$^{\mathsf{fpr}}$. To palliate the weakness of G2OT compared to G2V, Google proposes G2DT$^{\mathsf{fpr}}$ where a comparison through a second tap is required. The additional security provided by the second tap is displayed in Table 2, where we highlight in blue the differences between G2OT$^{\mathsf{fpr}}$ and G2DT$^{\mathsf{fpr}}$. The attacker must now be able to have its code displayed to the user and then selected onto the user's device in order to successfully login, so even in case of FS or NC and some read only access, it is not enough. Interestingly, in the NC scenario, we are now as secure as G2V, while having greater usability. We note that we are still not secure in the PH FS scenario. This means that an attacker controlling the user's network or some Wifi hotspot, could in practice mount an attack against G2DT$^{\mathsf{fpr}}$.

Threat Scenarios			G2DT$^{\mathsf{fpr}}$
	NC		✔
FS			✔
	NC	$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$	✔
	NC	$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$	✔
FS		$\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RO}}$	✔✔✗
FS		$\mathcal{M}^{\mathsf{t-usb}}_{\mathsf{in}:\mathcal{RO}}$	✔
	NC	$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$	✔
	NC	$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$	✔
FS		$\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RO}}$	✔
FS		$\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RO}}$	✔

Table 2: Analysis of the *Google 2-step* Double-Taps

5.3. Additional display

In this section, we propose and analyse small modifications of the previously presented protocols. Given the benefits discussed in section 5.2.3, we first add a fingerprint to G2V.

In *Google 2-step* some attacks occur because the attacker is able to replace a trusted login by an untrusted one, e.g. under $\mathcal{M}^{\mathsf{u-usb}}_{\mathsf{in}:\mathcal{RW}}$. If this happens, the attacker can obtain a session cookie for its own computer and perform additional undetected logins later on. A user might expect that by using a second factor, he should be able to securely login once on an untrusted computer and be assured that no additional login will be possible.

Thus, we also add for all the protocols the action the user is trying to perform (trusted or untrusted login) to the display. This mechanism may create some harmless "attacks" where the attacker replaces a trusted login with an untrusted login. However, such attacks indicate that an attacker may change to other types of actions, such as password reset, or disabling second-factor authentication.

Threat Scenarios				G2V^{fpr}	G2V^{dis}	G2OT^{dis}	G2DT^{dis}
PH				✔	✔	✔	✔
PH	FS			✖	✗✓✓	✖	✗✓✓
PH	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✖	✖	✖	✗✓✗
PH	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✖	✖	✖	✗✓✓
PH	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$		✖	✗✓✓	✖	✖
		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✔	✔	✓✓✓	✓
		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✔	✔	✓✓✓	✓
		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$		✗✓✗	✓✓✗	✓✓✗	✓✓✗
		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$		✓✓✗	✓✓✗	✓✓✗	✓✓✗
		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✗	✓✓✗	✓✓✗
	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$		✖	✓✗✗	✖	✓✓✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$		✖	✓✗✗	✖	✔
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$		✔	✔	✖	✓✗✗
	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$		✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✓✗✗	✖	✓✗✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✗✗	✖	✓✗✗
		$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$		✔	✔	✓✓✓	✓
		$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$		✔	✔	✓✓✓	✓
		$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$		✓✗✗	✔	✓✓✓	✓✓✓
		$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$		✓✗✓	✔	✓✓✓	✓✓✓
	FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$		✖	✗✓✓	✖	✔
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$		✖	✗✓✓	✖	✔
	FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$		✔	✔	✖	✗✓✓
	FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$		✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$		✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✗✓✓	✖	✗✓✓
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✖	✗✓✓	✖	✗✓✓

Table 3: *Google 2-step* protocols with additional display

We call G2V^{fpr}, G2V^{dis}, G2OT^{dis} and G2DT^{dis} the protocols versions that additionally display fingerprint, respectively the action, and provide in Table 3 the results of our analysis. To highlight the benefits of our modifications, we color additionally satisfied properties in blue, when considering G2V and G2V^{fpr}, G2V^{fpr} and G2V^{dis}, G2OT^{fpr} and G2OT^{dis} and G2DT^{fpr} and G2DT^{dis}.

It appears that adding the action - and the fingerprint in the G2V case - performs as expected: the protocols become secure in all the scenarios where the only possible attack was a mixing of actions.

5.4. Conclusion regarding *Google 2-step*

Currently, Google proposes G2V, G2OT, G2OT^{fpr} and G2DT^{fpr}. Adding the action to the display would provide additional security guarantees.

Among the studied mechanisms, G2V^{dis} and G2DT^{dis} provide the best security guarantees in our model, having each advantages and disadvantages. In Table 4, we provide a comparison between these two mechanisms. We observe that G2V^{dis} performs better than G2DT^{dis} only in scenarios where we have $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$, which may be considered as a powerful malware.

G2DT^{dis} provides better guarantees in many simpler threat scenarios, with for instance read-only access to the phone. As the code is sent back to the server from the phone

Threat Scenarios			G2V^{dis}	G2DT^{dis}
PH	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✗✓✗
PH	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✗✓✓
PH	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✗✓✓	✖
		$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	✖	✔
	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✔
	NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✔
	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✔	✖
	FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✗✗	✓✓✗
	FS	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✓✗✗	✓✓✓
	FS NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗
	FS	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✓✓✓	✓✗✗
	NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✔
	NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✔
	NC	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✔	✖
	FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✗✓✓	✓✓✓
	FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✗✓✓	✓✓✓
	FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✓✓✓	✗✓✓

Table 4: Comparison of *Google 2-step* with code or tap

rather than the computer, this mechanism is more resilient to malware on the computer.

Moreover, the code is sent through a TLS channel rather than via SMS, which may arguably provide better security.

Finally, even though *Google 2-step* may significantly improve security, phishing attacks combined with fingerprint spoofing are difficult to prevent. This seems to be inherent to the kind of protocol which is *Google 2-step*, where the security is only enforced through the 2nd factor. As we will see in the next section the FIDO U2F protocol may provide better guarantees for these threat scenarios.

5.5. FIDO U2F

FIDO's U2F adds cryptographic capabilities through its registration mechanism. As explained previously, the URL of the server the user is trying to authenticate to is included in the query made to the FIDO USB token, and also in the signature returned by the token. The server will then only grant login if the signature contains its own url.

We present the results of our formal analysis in Table 5. U2F is secure under many threat scenarios, including some that combine phishing and fingerprint spoofing. However, an attack is found when the computer runs malware that controls the USB interface of the trusted computer ($\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$). Indeed, the malware can then communicate with the U2F token, and thus send a request generated for

Threat Scenarios		U2F
		✔
PH		✔
FS		✓✓✓
	$\mathcal{M}^{t-dis}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖
	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✓✓✗
	$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✗✗✗
FS	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗
	$\mathcal{M}^{u-dis}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✔
	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✓✓✓
	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖
	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✓✗✓
	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✗✗✗

Table 5: U2F results

an attacker session. Also, if the attacker can control the TLS interface ($\mathcal{M}^{\mathsf{t-tls}}_{\mathsf{io}:\mathcal{RW}}$ or $\mathcal{M}^{\mathsf{u-tls}}_{\mathsf{io}:\mathcal{RW}}$), he may change the intended action and replace an untrusted login with a trusted one, thus providing the attacker with a trusted cookie. As a consequence, a login on an untrusted computer with U2F may enable future attacker logins on this computer.

This is a problem, as Yubikeys (an implementation of U2F token) claims protection against "phishing, session hijacking, man-in-the-middle, and malware attacks." While this holds for the first ones, this is not the case for the malware attacks. Moreover, one might expect that an external hardware token should allow users to login securely on an untrusted computer. Then, it is actually easy for an attacker to submit its own request to the user's token. Of course, a user has to press a button to accept a request. As we said previously, with a malware controlling the TLS connection, one press of the user may lead to many attacker logins, but this is a failure of the trust mechanism, not of U2F.

U2F may lead to another problem that is out of the scope of our analysis: the Yubikey does not have any way to provide feedback for a successful press. When the computer submits two request in a row to the token and the user just presses once, the user may believe that the press failed, and press once more. This is similar to the problem identified during the analysis of the One-Tap mechanism: success and failure of the second factor should not be silent.

To summarize, one might expect U2F to protect against malware, as it is based on a secure hardware token providing cryptographic capabilities. Thus, even if U2F does provide a better security than most existing solutions, it does not uphold this promise. However, the U2F mechanism providing protection against phishing is very interesting. What appears to be lacking from U2F is some feedback capabilities, i.e a screen, to notify failures, success, and maybe information such as the fingerprint of the computer.

5.6. Token Binding

We previously studied the security of the protocols combined with the *"Trust this computer"* mechanism where a cookie is used to authenticate a computer on the long term. While cookies are a common mechanism, a new protocol called TOKENBINDING [21] is under development. After a successful login a public key may be bound to the user account, and the corresponding secret key will be used to sign the session identifier of the following TLS sessions. We describe the protocol in appendix in Figure 7. It may be seen as a partial U2F where the keys are directly stored on the computer.

We provide in Table 6 the results of the formal analysis of TOKENBINDING combined with U2F and G2DT$^{\mathsf{dis}}$. It provides protection against a read only access to the TLS interface, because it is not any more sufficient to steal the cookie. The attacker needs to be able to access the private key of the user which was generated on his platform but never sent over the network.

We remark that for TOKENBINDING, having full access to the computer interfaces is not enough to steal the secret key because it is generated directly on the computer and never sent over the network. We could have specified as we did for the others a memory interface and the corresponding malware. However, this malware scenario is not useful for our comparison : it would have lead to attacks for both the cookie and TOKENBINDING.

6. Pratical Considerations

As mentioned previously, there are some interesting aspects that are outside of the scope of our threat models and formal analysis. We therefore discuss below some additional thoughts and findings.

Short integer for G2DT$^{\mathsf{dis}}$. Currently, G2DT$^{\mathsf{dis}}$ uses only a 2-digit integer. Hence, an attacker has probability $1/100$ to guess the integer, which is much higher than usually accepted. To maintain usability it might be worth exploring different mechanisms, such as using images or visual hashes. The only conditions are that the domain should be large, and a human should be able to instantly pick the correct value out of the three proposed ones.

Independence of the second factor. When trying to login on a malicious computer, we saw that the U2F token might be used by the attacker if he controls the USB interface. Therefore, our phone, which remains completely independent from the untrusted computer, is not affected by malware on the untrusted computer.

On the need of feedback. An advantage of the phone over the U2F token is also that the token does not provide feedback to the user, and two consecutive button presses may remain unnoticed. On the phone, a success or failure confirmation after pressing the button is easily provided to the user.

Storing the keys on a dedicated secure token. An advantage of the U2F token is that if your computer is compromised, the number of attacker logins is limited by the number of times the button is pressed. It is then more secure to store the keys on a device that ensures that keys are not completely compromised, rather than storing the keys on a computer or a smartphone, where a malware may extract them. We however saw that U2F does not provide perfect security, and if indeed the key cannot be compromised, one should be careful of how the token is used, to ensure that no unwanted computer becomes trusted, or that the user may not press twice the button in a row. A solution to mitigate key leakage for computer or smartphones could be to consider an Isolated Execution Environment, such as Intel SGX, ARM Trustzone or a Trusted Platform Module.

Carrying additional authenticators. An important question for multi-factor protocols is of course usability. From that point of view, the need to buy and carry an additional token may be cumbersome. Nowadays, more and more people possess and constantly carry their phone, making it a natural choice for a second factor.

Threat Scenarios				U2F	$U2F_{tb}$	$G2DT^{dis}$	$G2DT^{dis}_{tb}$
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✗✓✗	✗✓✓
PH	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✓✗✗	✓✗✗
	FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✓✓✗	✓✓✓	✖	✖

Table 6: Results for the TOKENBINDING extension

Disabling the second factor. On some websites, for instance Github, disabling the second factor (and then changing the password) does not require the use of the second factor, once a login was performed. It seems advisable to require a second factor authentication to disable the mechanism.

Privacy. A concern about authentication that is often ignored is unlinkability. If the same second factor is used to login on different providers, e.g. the user's phone, it might be possible to use this second factor to link accounts. Unlinkability is obviously not provided by $G2DT^{dis}$, because one provides the same phone number. However, U2F explicitly claims to achieve unlinkability by generating a fresh key pair for every distinct provider.

7. Conclusion

In this paper we propose a detailed threat model for multi-factor authentication protocols. It takes into account communication through TLS channels in an abstract way, yet modelling interesting details such as session identifiers and TLS sessions in with compromised agents. Moreover, we consider different levels of malwares, in a systematic way by representing a system as a set of interfaces. Additionally, we allow the adversary to perform phishing and spoof fingerprints, and consider scenarios where a careless user does not perform expected checks. We formalize this model in a dialect of the applied pi calculus and use the PROVERIF tool to systematically and automatically analyse several versions of *Google 2-step* and U2F in an extensive way, considering all combinations of threats. The resulting protocol comparison highlights strengths and weaknesses of the different mechanisms, and allows us to propose some simple variants, adding actions to the displayed information, which improves security.

As a direction for future work we consider the use of *enclaves* in trusted execution environments: such environments could provide execution certification and a way to enable secure login on a completely untrusted computer, if the computer is equipped with a trusted module. One could then use a phone as a U2F token assuming that we also have an efficient way to establish a channel between the computer and the phone in order to pass the payload. The U2F keys could be stored on the phone, and the next natural step would be to merge $G2DT^{dis}$ and U2F by performing a U2F on the phone in parallel of the $G2DT^{dis}$. The user would only see the $G2DT^{dis}$ part. $G2DT^{dis}$ combined with for instance the storage of the keys using a trusted execution environment, such as TrustZone would then palliate the issue of keys being revealed due to malware on the phone.

Finally, as discussed above, U2F tokens claim to achieve unlinkability. It would be interesting to formally model and study this property.

Acknowledgements. We wish to thank Arnar Birgisson and Alexei Czeskis for interesting discussions and the anonymous reviewers for their comments. We are grateful for the support from the ERC under the EU's Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC), as well as from the French National Research Agency (ANR) under the project Sequoia.

References

[1] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Nist special publication 800-63-3 – digital identity guidelines," Jun. 2017, available at https://doi.org/10.6028/NIST.SP.800-63-3.

[2] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[3] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, K. K. Choong, Yee-Yin Greene, and M. F. Theofanos, "Nist special publication 800-63b – digital identity guidelines – authentication and lifecycle management," Jun. 2017, available at https://doi.org/10.6028/NIST.SP.800-63b.

[4] "Google 2 Step," https://www.google.com/landing/2step/.

[5] "FIDO Yubikey," https://www.yubico.com/solutions/fido-u2f/.

[6] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 553–567. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.44

[7] O. Pereira, F. Rochet, and C. Wiedling, "Formal Analysis of the Fido 1.x Protocol," in *The 10th International Symposium on Foundations & Practice of Security*, ser. Lecture Notes in Computer Science (LNCS). Springer, 10 2017.

[8] S. Kremer and R. Künnemann, "Automated analysis of security protocols with global state," *CoRR*, vol. abs/1403.1142, 2014. [Online]. Available: http://arxiv.org/abs/1403.1142

[9] R. Künnemann and G. Steel, *YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 257–272. [Online]. Available: https://doi.org/10.1007/978-3-642-38004-4_17

[10] A. Armando, R. Carbone, and L. Zanetti, *Formal Modeling and Automatic Security Analysis of Two-Factor and Two-Channel Authentication Protocols*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 728–734. [Online]. Available: https://doi.org/10.1007/978-3-642-38631-2_63

[11] D. Basin, S. Radomirovic, and L. Schmid, "Modeling human errors in security protocols," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, June 2016, pp. 325–340.

[12] D. A. Basin, S. Radomirovic, and M. Schläpfer, "A complete characterization of secure human-server communication," in *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, 2015, pp. 199–213. [Online]. Available: https://doi.org/10.1109/CSF.2015.21

[13] "G Suite updates - Improved phone prompts for 2-Step Verification," https://gsuiteupdates.googleblog.com/2017/02/improved-phone-prompts-for-2-step.html.

[14] D. Dolev and A. Yao, "On the security of public key protocols," in *Proc. 22nd Symp. on Foundations of Computer Science (FOCS'81)*. IEEE Comp. Soc. Press, 1981, pp. 350–357.

[15] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *meltdownattack.com*, 2018. [Online]. Available: https://meltdownattack.com/meltdown.pdf

[16] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *meltdownattack.com*, 2018. [Online]. Available: https://spectreattack.com/spectre.pdf

[17] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," *SIGPLAN Not.*, vol. 36, no. 3, pp. 104–115, Jan. 2001. [Online]. Available: http://doi.acm.org/10.1145/373243.360213

[18] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *Journal of the ACM*, vol. 65, no. 1, pp. 1:1–1:41, Oct. 2017.

[19] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016. [Online]. Available: http://dx.doi.org/10.1561/3300000004

[20] "Proverif source files," https://sites.google.com/site/multifactorformalverif/.

[21] A. Popov, M. Nystrom, D. Balfanz, A. Langley, N. Harper, and J. Hodges, "Token Binding over HTTP," Internet Engineering Task Force, Internet-Draft draft-ietf-tokbind-https-12, Jan. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tokbind-https-12

Appendix A. An attack on G2OT

We provide bellow the steps of an attack on G2OT when only the password is compromised:

1) The User enters its password and email.
2) Its browser tells him to press yes on his phone.
3) The attacker detects that the user contacted Google. After the first answer from Google the attacker blocks all further messages.
4) The Attacker initiates a session of its own by entering the user's email and password.
5) Depending on the timing, two things then happen:
   - The user press yes, nothing happens on its screen, and the attacker is logged in.
   - The user press yes, nothing happens on its screen, but another yes/no pops up on his phone. If he presses yes once more, the attacker is logged in.

In step 3, we block other answers from the server, because whenever two simultaneous login requests are submitted to the server, the server informs first one that there is a problem, resulting in an error message for the user. Thus to complete the attack in a completely invisible way, the attacker should be able to block the server response. If he cannot do this, depending on the timing, the user may not validate if he sees the error. Finally, in the attack description, we provide two cases, because we were not able to obtain a deterministic response from the phone. Indeed, when two login attempts where made at the same time, we sometime needed to validate twice on the screen, and sometimes only once. This may be due to the order of reception of requests, and session expiration on the server side. To be robust, the implementation should not accept this kind of situation, and reject any kind of simultaneous login from different sessions, or at least display it clearly on the phone, as it is done in the browser. Indeed, we believe that it is plausible that users, after having pressed yes on their phone without any result, would press yes once more.

It is however important to mention that the attacker must be able to detect the user login, and block messages from the server to the client.

Appendix B. TOKENBINDING

TOKENBINDING is meant to be enabled on the trusted devices of the user, once the trust was established through a previously successful Multi Factor Authentication. We present it in Figure 7.

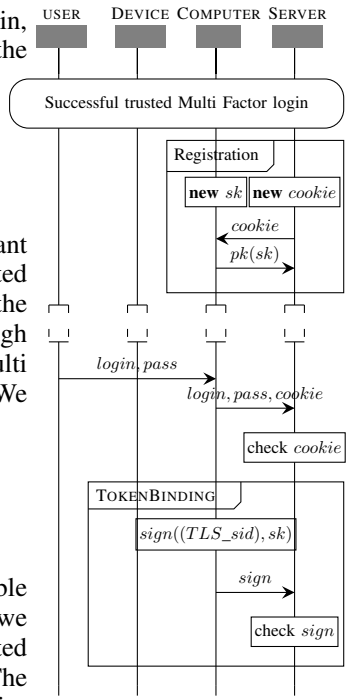

Figure 7: TOKENBINDING

Appendix C. Global results

We summarize in Table 7 and 8 all the results we computed using the automated generation of scenarios. The results were obtained in 8 minutes of computing on a server with 12 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 50Gb of RAM. During the computation, 6172 calls to PROVERIF were made. As PROVERIF may not terminate we set a timeout at 3 seconds: only two scenarios exceeded the timeout limit. For readability, we only display the minimal or maximal interesting scenarios, and results which are implied by an other scenario are greyed. The table was completely generated by an automated script, to avoid transcription mistakes.

Threat Scenarios				G2V	G2V^{fpr}	G2V^{dis}	G2C	G2OT^{fpr}	G2OT^{dis}	G2DT^{fpr}	G2DT^{dis}	U2F	U2F$_{tb}$	G2DT$^{dis}_{tb}$
				✔	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔
PH				✖	✔	✓✓✓	✖	✔	✔	✔	✓✓✓	✔	✔	✓✓✓
		NC		✔	✔	✔	✖	✖	✖	✔	✔	✔	✔	✔
	FS			✔	✔	✔	✖	✖	✖	✔	✔	✔	✔	✔
PH		NC		✖	✖	✖	✖	✖	✖	✖	✖	✓✓✓	✔	✖
PH	FS			✖	✖	✗✓✓	✖	✖	✖	✖	✗✓✓	✔	✔	✗✓✓
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✗✓-	✓✓✗	✔	✗✓✓
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✗✓✓	✔	✔	✗✓✓
PH	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✔	✖
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✗✓✓	✖	✖	✖	✖	✖	✔	✔	✖
PH	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✓✓✗	✖
PH	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✔	✖
PH	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
PH	FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
			$\mathcal{M}^{dev}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✔	✔	✔	✔	- - -	- - -	✔
			$\mathcal{M}^{t-dis}_{io:\mathcal{RO}}$	✔	✔	✔	✖	✓✓✓	✓✓✓	✓✓✓	✓✓✓	✔	✔	✓✓✓
			$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✓	✓✓✓	✖	✓✓✓	✓✓✓	✔	✔	✔	✔	✔
			$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✓✓✓	✓✓✓	✖	✓✓✓	✓✓✓	✔	✔	✔	✔	✔
		NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✓✓✓	✓✓✓	✔	✔	✔
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✔	✔	✔	✔	✔
			$\mathcal{M}^{dev}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✔	✔	✔	✔	✖	✖	✔
			$\mathcal{M}^{dev}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
			$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✗✓✗	✓✓✗	✖	✗✓✗	✓✓✗	✗✓✗	✓✓✗	✖	✗✗-	✓✓✗
		NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✔	✔	✔	✖	✖	✖	✖	✖	✔	✔	✖
			$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✓✓✗	✓✓✗	✖	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✓✓✗
		NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✗✗✗	✗✗✗	✖	✗✗-	✗✗✗
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✗✗✗	✗✗✗	✓✓✗	✓✓✗	✗✗✗
			$\mathcal{M}^{t-dis}_{io:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✗✓✗	✓✓✗	✖	✗✓✗	✓✓✗	✗✓✗	✓✓✗	✖	✗✗✗	✓✓✗
			$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✗✓✗	✓✓✗	✖	✗✓✗	✓✓✗	✗✓✗	✓✓✗	✖	✗✗✗	✓✓✗
		NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✓	✔	✖
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
			$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗	✓✓✗	✖	✓✓✗	✓✓✗	✓✓-	✓✓-	✓✓✗	✓✓✗	✓✓✗
			$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✓✓✗	✓✓✗	✖	✓✓✗	✓✓✗	✓✓✗	✓✓✗	✗✗✗	✗✗✗	✓✓✗
		NC	$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
			$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✓✓✗	✓✓✗	✖	✓✓✗	✓✓✗	✓✓-	✓✓-	✖	✖	✓✓✗
	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✓✗✗	✖	✖	✖	✓✓-	✓✓-	✓✓✗	✔	✔
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$	✖	✖	✓✗✗	✖	✖	✖	✔	✔	✔	✔	✔
	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✓✓-	✓✓-	✓✓✗	✔	✔
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✔	✔	✔	✖	✖	✖	✖	✓✗✗	✔	✔	✓✗✗
	FS		$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✓✗✗	✖	✖	✖	✗✗✗	✓✗✗	✖	✗✗✗	✓✗✗
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$	✖	✖	✓✗✗	✖	✖	✖	✗✗✗	✓✗✗	✓✓✗	✓✓✗	✓✗✗
	FS	NC	$\mathcal{M}^{t-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
	FS		$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✓✗✗	✖	✖	✖	✖	✓✗✗	✓✓✗	✔	✓✗✗
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$	✖	✖	✓✗✗	✖	✖	✖	✖	✓✗✗	✔	✔	✓✗✗
	FS	NC	$\mathcal{M}^{t-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✓✓✗	✔	✖
	FS		$\mathcal{M}^{t-usb}_{in:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✓✗✗	✖	✖	✖	✖	✓✗-	✓✓✗	✓✓✗	✓✗✗
	FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$	✖	✖	✓✗✗	✖	✖	✖	✖	✓✗✗	✖	✖	✓✗✗
	FS		$\mathcal{M}^{t-usb}_{io:\mathcal{RW}}$ $\mathcal{M}^{t-tls}_{io:\mathcal{RO}}$	✖	✖	✓✗✗	✖	✖	✖	✖	✓✗-	✖	✖	✓✗✗

Protocols

- G2V- *Google 2-step* with Verification code
- G2V^{fpr}- G2V with fingerprint display
- G2V^{dis}- G2V^{fpr} with action display
- G2OT- *Google 2-step* One Tap
- G2OT^{fpr}- G2OT with fingerprint display
- G2OT^{dis}- G2OT with action display
- G2DT^{fpr}- *Google 2-step* Double Tap (random to compare)
- G2DT^{dis}- G2DT^{fpr} with action display
- U2F- *FIDO's U2F*
- U2F$_{tb}$- TokenBinding U2F
- G2DT$^{dis}_{tb}$- TokenBinding G2DT^{dis}

Scenarios:

- NC- No Compare, the human does not compare values
- FS- Fingerprint spoof, the attacker can copy the user IP address
- PH- The user might be victim of phishing only on trusted everyday connections
- $\mathcal{M}^{interface}_{in:acc1,out:acc2}$- The interface inputs are given to the attacker with access level acc1, and acc2 for the outputs or untrusted computers

Notations:

- ✓- Property satisfied (✔if all three)
- ✗- Attack found (✖if all three)
- ✗- Attack also present in a weaker scenario
- ✓- Property also satisfied in a stronger scenario
- - - Either scenario not pertinent, or failure to reconstruct attack trace

Table 7: Global results for malware on trusted platform

Threat Scenarios		G2V	G2V^{fpr}	G2V^{dis}	G2C	G2OT^{fpr}	G2OT^{dis}	G2DT^{fpr}	G2DT^{dis}	U2F	U2F$_{tb}$	G2DT$^{dis}_{tb}$
	$\mathcal{M}^{u-dis}_{io:\mathcal{RO}}$	✔	✔	✔	✖	✔	✔	✔	✓✓✓	✔	✔	✓✓✓
	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✓✓✓	✔	✖	✓✓✓	✔	✔	✔	✔	✔	✔
	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✓✓✓	✔	✖	✓✓✓	✔	✔	✔	✔	✔	✔
NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✔	✔	✔	✔	✔
NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✔	✔	✔	✔	✔
	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✓✗✗	✓✓✓	✖	✓✗✗	✔	✓✗✗	✓✓✓	✖	✖	✓✓✓
NC	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✔	✔	✔	✖	✖	✖	✖	✖	✔	✔	✖
	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✖	✓✗✓	✔	✖	✓✗✓	✔	✓✗✓	✔	✓✗✓	✓✗✓	✔
NC	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✗✗✗	✖	✖	✖	✖
NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✗✗✗	✖	✓✗✓	✓✗✓	✖
NC	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
NC	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✔	✔	✖
	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✖	✓✗✓	✓✓✓	✖	✓✗✓	✔	✓✗✓	✓✓✓	✗✗✗	✗✗✗	✓✓✓
NC	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✖	✗✓✓	✖	✖	✖	✔	✓✓✓	✔	✔	✓✓✓
FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$	✖	✖	✗✓✓	✖	✖	✖	✔	✓✓✓	✔	✔	✓✓✓
FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✔	✔	✓✓✓	✖	✖	✖	✖	✗✓✓	✔	✔	✗✓✓
FS	$\mathcal{M}^{u-tls}_{io:\mathcal{RW}}$	✖	✖	✗✓✓	✖	✖	✖	✗✗✗	✗✓✓	✖	✖	✗✓✓
FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RW}}$	✖	✖	✗✓✓	✖	✖	✖	✗✗✗	✗✓✓	✓✗✓	✓✗✓	✗✓✓
FS	$\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$ $\mathcal{M}^{u-tls}_{io:\mathcal{RO}}$	✖	✖	✗✓✓	✖	✖	✖	✖	✗✓✓	✔	✔	✗✓✓
FS	$\mathcal{M}^{u-usb}_{in:\mathcal{RO}}$ $\mathcal{M}^{u-dis}_{io:\mathcal{RW}}$	✖	✖	✗✓✓	✖	✖	✖	✖	✗✓✓	✔	✔	✗✓✓
FS	$\mathcal{M}^{u-usb}_{io:\mathcal{RW}}$	✖	✖	✗✓✓	✖	✖	✖	✖	✗✓✓	✖	✖	✗✓✓

Protocols

- G2V- *Google 2-step* with Verification code
- G2V^{fpr}- G2V with fingerprint display
- G2V^{dis}- G2V^{fpr} with action display
- G2OT- *Google 2-step* One Tap
- G2OT^{fpr}- G2OT with fingerprint display
- G2OT^{dis}- G2OT with action display
- G2DT^{fpr}- *Google 2-step* Double Tap (random to compare)
- G2DT^{dis}- G2DT^{fpr} with action display
- U2F- *FIDO's U2F*
- U2F$_{tb}$- TokenBinding U2F
- G2DT$^{dis}_{tb}$- TokenBinding G2DT^{dis}

Scenarios:

- NC- No Compare, the human does not compare values
- FS- Fingerprint spoof, the attacker can copy the user IP address
- PH- The user might be victim of phishing only on trusted everyday connections
- $\mathcal{M}^{interface}_{in:acc1,out:acc2}$- The interface inputs are given to the attacker with access level acc1, and acc2 for the outputs or untrusted computers

Notations:

- ✓- Property satisfied (✔if all three)
- ✗- Attack found (✖if all three)
- ✗- Attack also present in a weaker scenario
- ✓- Property also satisfied in a stronger scenario
- - - Either scenario not pertinent, or failure to reconstruct attack trace

Table 8: Global results for malwares on untrusted platform

Universal equivalence and majority of probabilistic programs over finite fields

Gilles Barthe
MPI-SP & IMDEA Software Institute

Charlie Jacomme
LSV, CNRS & ENS Paris-Saclay &
Inria & Université Paris-Saclay

Steve Kremer
LORIA, Inria Nancy-Grand Est &
CNRS & Université de Lorraine

Abstract

We study decidability problems for equivalence of probabilistic programs, for a core probabilistic programming language over finite fields of fixed characteristic. The programming language supports uniform sampling, addition, multiplication and conditionals and thus is sufficiently expressive to encode boolean and arithmetic circuits. We consider two variants of equivalence: the first one considers an interpretation over the finite field $\mathbb{F}_q$, while the second one, which we call universal equivalence, verifies equivalence over all extensions $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$. The universal variant typically arises in provable cryptography when one wishes to prove equivalence for any length of bitstrings, i.e., elements of $\mathbb{F}_{2^k}$ for any k. While the first problem is obviously decidable, we establish its exact complexity which lies in the counting hierarchy. To show decidability, and a doubly exponential upper bound, of the universal variant we rely on results from algorithmic number theory and the possibility to compare local zeta functions associated to given polynomials. Finally we study several variants of the equivalence problem, including a problem we call majority, motivated by differential privacy.

CCS Concepts: • **Security and privacy → Logic and verification**; • **Theory of computation → Program reasoning**.

Keywords: program equivalence, probabilistic programs, finite fields, decidability and complexity

ACM Reference Format:
Gilles Barthe, Charlie Jacomme, and Steve Kremer. 2020. Universal equivalence and majority of probabilistic programs over finite fields. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3373718.3394746

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394746

1 Introduction

Program equivalence is one of the most fundamental tools in the theory of programming languages and arguably the most important example of relational property. Program equivalence has been studied extensively, leading to numerous decidability results and sound proof methods. This paper is concerned with the decidability of equivalence and relational properties for a core imperative probabilistic programming language. Like many other probabilistic programming languages, our language supports sampling from distributions, and conditioning distributions on an event. The specificity of our language is that it operates over finite fields of the form $\mathbb{F}_{q^k}$. Therefore, expressions are interpreted as polynomials and assertions are boolean combinations of polynomial identities. Sampling is interpreted using the uniform distributions over sets defined by assertions, and branching and conditioning are relative to assertions.

We consider two relational properties, equivalence and majority, which we define below, and several related properties, which we explain in the next paragraph. For each property, we consider two variants of the problem. In the first variant, which we call the fixed case, the value of k is fixed. In the second variant, which we call the universal variant, we require the property to hold for all possible values of k. Consider two programs P_1 and P_2 with m inputs and n outputs. These programs are interpreted as functions $[\![P_1]\!]^{q^k}, [\![P_2]\!]^{q^k} : \mathbb{F}_{q^k}^m \to \mathsf{Distr}(\mathbb{F}_{q^k}^n)$.

- q^k*-equivalence* (denoted $P_1 \approx_{q^k} P_2$) requires that P_1 and P_2 define the same distributions: $[\![P_1]\!]^{q^k} = [\![P_2]\!]^{q^k}$. Equivalently, for every input $\vec{a} \in \mathbb{F}_{q^k}^m$ and output $\vec{b} \in \mathbb{F}_{q^k}^n$,

$$\mathbb{P}\{\vec{x} = \vec{b} \mid \vec{x} \xleftarrow{\$} [\![P_1]\!]^{q^k}(\vec{a})\} = \mathbb{P}\{x = \vec{b} \mid \vec{x} \xleftarrow{\$} [\![P_2]\!]^{q^k}(\vec{a})\}.$$

  q^∞-equivalence requires the property to hold on all extensions of a field, i.e.,

$$P_1 \approx_{q^\infty} P_2 \text{ iff } \forall k.\ P_1 \approx_{q^k} P_2$$

- q^k*-majority* requires that for a fixed $r \in \mathbb{Q}$, and for every input $\vec{a} \in \mathbb{F}_{q^k}^m$ and output $\vec{b} \in \mathbb{F}_{q^k}^n$, we have

$$\mathbb{P}\{\vec{x} = \vec{b} \mid \vec{x} \xleftarrow{\$} [\![P_1]\!]^{q^k}(\vec{a})\} \leq$$
$$r \cdot \mathbb{P}\{\vec{x} = \vec{b} \mid \vec{x} \xleftarrow{\$} [\![P_2]\!]^{q^k}(\vec{a})\}.$$

  q^k*-0-majority* (denoted $P_1 \prec^r_{q^k} P_2$) is a variant of majority, where we only consider the output $b = 0^n$, rather than quantifying over all outputs. q^∞*-0-majority* requires the property to hold on all extensions of a field, i.e.,

$$P_1 \prec^r_{q^\infty} P_2 \text{ iff } \forall k.\ P_1 \prec^r_{q^k} P_2$$

The following two boolean programs illustrate the difference between equivalence and universal equivalence.

Example 1.1.

$$x \xleftarrow{\$} \mathbb{F};\ \text{return } (x^2 + x) \quad \Big| \quad \text{return } 0$$

are 2- but not 2^2*-equivalent*, and hence not 2^∞-equivalent. Indeed, when instantiating $\mathbb{F}$ with $\mathbb{F}_2$, the left hand side program simply evaluates to zero, which is not the case with $\mathbb{F}_4$. On the other hand, the programs

$$x \xleftarrow{\$} \mathbb{F};\ \text{return } (x) \quad \Big| \quad x \xleftarrow{\$} \mathbb{F};\ \text{return } (x + 1)$$

are q^∞-equivalent as both programs define the uniform distribution over $\mathbb{F}$, whatever finite field is used for the interpretation of $\mathbb{F}$. These examples also illustrate the difference with the well-studied polynomial identity testing (PIT) problem, as the first two programs are 2-equivalent, while PIT does not consider $x^2 + x$ and 0 to be equal on $\mathbb{F}_2$, nor would Q_1 and Q_2 be considered identical.

The fixed and universal variants of the equivalence and majority problems are directly inspired from applications in security and privacy. In the fixed setting, the equivalence and majority problems are related to probabilistic non-interference and differential privacy. The relationships between probabilistic non-interference and equivalence and between differential privacy and majority are explained informally as follows:

- probabilistic non-interference: for simplicity, assume that P has two inputs x (secret) and y (public), and a single (public) output. For every x, let P_x be the unique program such that $P_x(y) = P(x, y)$. Then P is non-interfering iff for every x_1 and x_2, the two programs P_{x_1} and P_{x_2} are equivalent.
- differential privacy: for simplicity consider the case where the base field is $\mathbb{F}_2$. For every program P with n inputs, define the residual programs $P_{i,0}$ and $P_{i,1}$ obtained by fixing the i-th output to 0 and 1 respectively. Then the program P is $log(r)$-differentially private iff for every i, $P_{i,0}$ and $P_{i,1}$ (and $P_{i,1}$ and $P_{i,0}$) satisfy r-majority.

In the universal setting, the parameter k can loosely be understood as the security parameter. Universal equivalence is a special case of statistical indistinguishability and as such arises naturally in provable security, where the goal is to prove (depending on applications either as end goal, or as an intermediate goal) that two programs are equivalent for all possible interpretations (e.g. for all possible lengths of bitstrings, i.e. for all $\mathbb{F}_{2^k}$).

Summary of results

We also consider the following problems, which are also motivated by security and privacy and are directly related to equivalence:

- (bounded) simulatability: given programs P_1 and P_2, does there exist a context $C[\cdot]$ (of bounded degree) such that $C[P_1]$ is equivalent to P_2;
- independence: are outputs Y and Y' of program P independent conditioned on Z, i.e. for every input x, is the distribution of Y independent from the distribution of Y', when conditioning on the value of Z? Although independence is not naturally expressed as a relational property, it has been shown in [4] that relational methods are useful for proving independence.

The first contribution of the paper is a systematic study of the complexity of the aforementioned problems in the fixed setting. We prove that the q^k*-equivalence* problem is $\mathsf{coNP}^{\mathsf{C_=P}}$-complete for any fixed k. We also study the special case of *linear* programs, i.e. multiplication, conditional and conditioning free, for which the problem can be decided in polynomial time. For the majority problem, we consider two settings: programs with and without inputs. We show that the k-majority problem for inputless programs is PP-complete, whereas the k-majority for arbitrary programs is $\mathsf{coNP}^{\mathsf{PP}}$-complete—thus the second problem is strictly harder than the first, unless $\mathsf{PH} \subset \mathsf{PP}$[1]. The proofs are given by reductions to MAJSAT and E−MAJSAT respectively. Note that we do not include any result about bounded simulatability in the finite case, since we only derive easy consequences of equivalence. These results complement recent work on the complexity of checking differential privacy for arithmetic circuits [15], see Related Work below.

The second, and main contribution, is the study of universal equivalence, q^∞-equivalence for short, and universal (0-)majority, q^∞-(0-)majority for short. First, we show that the q^∞*-equivalence* problem is in 2-EXP and $\mathsf{coNP}^{\mathsf{C_=P}}$-hard.

Our proof is based on local zeta Riemann functions, a powerful tool from algebraic geometry, that characterize the number of zeros of a tuple of polynomials in all extensions of a finite field. Lauder and Wan [19] notably propose an algorithm to compute such functions, whose complexity is however exponential. Based on this result, our proof proceeds in three steps.

[1] As $\mathsf{PH} \subset \mathsf{coNP}^{\mathsf{PP}}$, $\mathsf{PP} = \mathsf{coNP}^{\mathsf{PP}}$ would imply $\mathsf{PH} \subset \mathsf{PP}$ which is commonly believed to be false.

First, we give a reduction for arithmetic programs (no conditionals, nor conditioning) from universal equivalence to checking that some specific local zeta Riemann functions are always null. Then, we reduce the general case to programs without conditioning, and programs without conditioning to arithmetic programs. To justify the use of the local zeta Riemman functions, we also provide counterexamples why simpler methods fail or only provide sufficient conditions. Our decidability result significantly generalize prior work on universal equivalence [3], which considers the case of linear programs, see Related Work below. In the special case of *arithmetic* programs, i.e., programs without conditionals nor conditioning, equivalence can be decided in EXP-time, rather than 2-EXP.

Second, we give an exponential reduction from the universal 0-majority problem to the positivity problem for Linear Recurrence Sequences (LRS), which given a LRS, asks whether it is always positive. Despite its apparent simplicity, the positivity problem remains open. Decidability has been obtained independently by Mignotte et al [25] and by Vereshchagin [32] for LRS of order ≤ 4 and later by Ouaknine and Worrell [29] for LRS with order ≤ 5. Moreover, Ouaknine and Worrell prove in the same paper that deciding positivity for LRS of order 6 would allow to solve hard open problems in Diophantine approximation. In the general case, the best known lower bound for the positivity problem is NP-hardness [28].

Our reduction is based on the observation that the Taylor series of any rational functional satisfies a linear recurrence sequence. Therefore, every tuple of polynomials yields a linear recurrence sequence via its local zeta Riemann function. Unfortunately, the order of the linear reccurence sequence is related to the degree of the local zeta Riemann function, and thus decidability results for small orders do not apply. This suggests that the problem may not have an efficient solution. Using the results from [18], we observe that the reduction extends to a more general form of universal majority problem.

Finally, we obtain lower complexity bounds by reducing the finite case to the universal case. It remains an interesting open question whether the universal case is strictly harder than the finite case.

Figures 1 and 2 summarize our results for the equivalence and majority problems.

Related work

Universal equivalence. The case of linear programs is studied in [3]. The authors propose a decision procedure for universal equivalence based on the classic XOR-lemma [11]. We give an alternative decision procedure and analyze its complexity.

The case of linear programs with random oracles is considered in [9]. The authors give a polynomial time decision procedure for computational indistinguishability of two inputless programs. Informally, computational indistinguishability is an approximate notion of universal equivalence, stating that the statistical distance between the output of two programs on the same input is upper bounded by a negligible function of the parameter k. Their proof is based on linear algebra.

The case of pseudo-linear (i.e. linear with conditionals) programs is considered in [17]. The authors consider the universal simulatability problem, rather than the universal equivalence problem. The crux of their analysis is a completeness theorem for pseudo-linear functions. In Section 4.3, we show that universal equivalence reduces to universal simulatability. As [17] shows the decidability of universal simulatability for pseudo-linear programs, it therefore follows that universal equivalence of pseudo-linear programs is decidable.

Fixed equivalence. There is a vast amount of literature on proving equivalence of probabilistic programs. We only review the most relevant work here.

Murawski and Ouaknine [26] prove decidability of equivalence of second-order terms in probabilistic ALGOL. Their proof is based on a fully abstract game semantics and a connection between program equivalence and equivalence of probabilistic automata.

Legay *et al* [20] prove decidability of equivalence for a probabilistic programming language over finite sets. Their language supports sampling from non-uniform distributions, loops, procedure calls, and open code, but not conditioning. They show that program equivalence can be reduced to language equivalence for probabilistic automata, which can be decided in polynomial time.

Barthe *et al* [5] develop a relational program logic for probabilistic programs without conditioning. Their logic has been used extensively for proving program equivalence, with applications in provable security and side-channel analysis.

Majority problems. The closest related work develops methods for proving differential privacy or for quantifying information flow.

Frederikson and Jha [14] develop an abstract decision procedure for satisfiability modulo counting, and then use a concrete instantiation of their procedure for checking representative examples from multi-party computation.

Barthe et al [2] show decidability of ϵ-differential privacy for a restricted class of programs. They allow loops and sampling from Laplace distributions, but impose several other constraints on programs. An important aspect of their work is that programs are parametrized by $\epsilon > 0$, so their decision procedure establishes ϵ-differential privacy for all values of ϵ. Technically, their decision procedure relies on the decidability of a fragment of the reals with exponentials by McCallum and Weispfenning [24].

	x-(*conditional*)-{*equivalence*, *independence*, *uniformity*}			q^∞-*simulatability*
	linear	arithmetic	general	decidable
$x = q^k$	PTIME	coNP$^{C_=P}$-complete	coNP$^{C_=P}$-complete	coNP$^{C_=P}$-hard
$x = q^\infty$	PTIME	EXP coNP$^{C_=P}$-hard	2-EXP coNP$^{C_=P}$-hard	

Figure 1. Summary of results related to equivalence

	q^k-0-*majority*	q^k-*majority*	q^∞-0-*majority*	q^∞-*majority*
without inputs	PP-complete	coNP^{PP}-complete	PP-hard	
			$\leq_{EXP}$ POSITIVITY	
with inputs	coNP^{PP}-complete		?	
			coNP^{PP}-hard	

Figure 2. Summary of results related to majority

Gaboardi, Nissim and Purser [15] study the complexity of verifying pure and approximate (ϵ, δ)-differential privacy for arithmetic programs, as well as approximations of the parameters ϵ and δ. The parameter δ quantifies the approximation and $\delta = 0$ corresponds to the pure case. Our majority problem can be seen as a subcase of differential privacy, where r corresponds to ϵ, and $\delta = 0$. In particular, the complexity class they obtain for pure differential privacy coincides with the complexity of our 0-majority problem, even when restricted to the case $r = 1$. This means that the ϵ parameter does not essentially contribute to the complexity of the verification problem. Also, while they consider arithmetic programs, we consider the more general case of programs with conditioning.

Chistikov, Murawski and Purser [10] also study the complexity of approximating differential privacy, but in the case of Markov Chains.

Theory of fields. A celebrated result by Ax [1] shows that the theory of finite fields is decidable. In a recent development based on Ax's result, Johnson [16] proves decidability of the theory of rings extended with quantifiers $\mu_k^n x.\, P$, stating that the number of x such that P holds is equal to k modulo n. Although closely related, these results do not immediately apply to the problem of equivalence.

2 Programming Language

We consider a high-level probabilistic programming language with sampling from semi-algebraic sets and conditioning, as well as a more pure, yet equi-expressive, core language that can encode all previous constructs and define its formal semantics.

$P ::=$		*polynomials*
$\mid$	$i \in \mathbb{F}_q$	fixed value
$\mid$	x	variable
$\mid$	$P_1 + P_2$	field addition
$\mid$	$P_1 \times P_2$	field multiplication
$b ::=$		*boolean conditions*
$\mid$	$P = 0$	atomic formula
$\mid$	$b_1 \wedge b_2$	and
$\mid$	$b_1 \vee b_2$	or
$\mid$	$\neg b$	not
$e ::=$		*program expressions*
$\mid$	$x := P$	assignment
$\mid$	$\vec{r} \xleftarrow{\$} \{X \in \mathbb{F}^m \mid b\}$	sampling
$\mid$	observe b	observe
$\mid$	$e_1; e_2$	sequential composition
$\mid$	if b then e_1 else e_2	conditional branching
$\mid$	return $(P_1, \ldots, P_n)$	return of arity n

Figure 3. Program syntax

2.1 Syntax and informal semantics

We define in Figure 3 the syntax for simple probabilistic programs (without loops nor recursion[2]). Our programs will operate on finite fields. We denote by $\mathbb{F}_q$ the (unique) finite field with q elements, where $q = p^s$ for some integer s and prime p. Programs are parametrized by a finite field $\mathbb{F}$, which will be instantiated by some $\mathbb{F}_{q^k}$ during the intepretation. Given a polynomial $P \in \mathbb{F}_q[x_1, \ldots, x_m]$ and $X \in \mathbb{F}_{q^k}^m$, we denote by $P(X)$ the evaluation of P given X inside $\mathbb{F}_{q^k}$.

[2]Universal equivalence for programs over finite fields with loops becomes undecidable.

The expressions of our programs provide constructs for assigning a polynomial P to a variable ($x := P$), as well as, for randomly sampling values. With for instance $\vec{r} = r_1, \ldots, r_m$, the expression $r_1, \ldots, r_m \xleftarrow{\$} \{X \in \mathbb{F}^m \mid b\}$ uniformly samples m values from the set of m-tuples of values in $\mathbb{F}$ such that the condition b holds, and assigns them to variables $r_1, \ldots, r_m$. For example, $r \xleftarrow{\$} \{x \in \mathbb{F} \mid 0 = 0\}$ (which we often simply write $r \xleftarrow{\$} \mathbb{F}$) uniformly samples a random element in $\mathbb{F}$, while $r_1, r_2 \xleftarrow{\$} \{x_1, x_2 \in \mathbb{F}^2 \mid \neg(x_1 = 0)\}$ samples two random variables, ensuring that the first one is not 0. Note that the use of polynomial conditions allows to express any rational distribution over the base field $\mathbb{F}_q$.

The construct observe b allows to condition the continuation by b: if b evaluates to false the program fails; the semantics of a program is the conditional distribution where b holds. Expressions also allow classical constructs for sequential composition, conditional branching and returning a result.

In a well-formed program we suppose that every variable is bound at most once, and if it is bound, then it is only used after the binding. Unbound variables correspond to the inputs of the program. We moreover suppose that each branch of a program P ends with a return instruction that returns the same number n of elements; n is then called the arity of the program and denoted $|P|$. Given two sets of variables I and R, we denote by $\mathcal{P}_q(I, R)$ the set of such well-formed programs, where I is the set of unbound variables (intuitively, the set of input variables) and R the set of variables sampled by the program.

Example 2.1. Consider the following simple program :

$$\begin{aligned}&\text{if } i = 0 \text{ then return } 0\\ &\text{else } r \xleftarrow{\$} \mathbb{F}; \text{observe } r \times i = 1; \text{return } r\end{aligned}$$

This program defines a probabilistic algorithm for computing the inverse of a field element i. If i is 0, by convention the algorithm returns 0. Otherwise, the algorithm uniformly samples an element r. This is obviously not a practical procedure for computing an inverse, but we use it to illustrate the semantics of conditioning. The observe instruction checks whether r is the inverse of i. If this is the case we return r, otherwise the program fails. As we will see below, our semantics normalizes the probability distribution to only account for non-failing executions. Hence, this algorithm will return the inverse of any positive i with probability 1. Equivalently, this program can be written by directly conditioning the sample , replacing "$r \xleftarrow{\$} \mathbb{F}; \text{observe } r \times i = 1;$" by "$r \xleftarrow{\$} \{x \in \mathbb{F} \mid x \times i = 1\};$".

2.2 A core language

While the above introduced syntax is convenient for writing programs, we introduce a more pure, core language that is actually equally expressive and will ease the technical developments in the remaining of the paper. To define this core language, we add an explicit failure instruction $\bot$, similarly to [7]. It allows us to get rid of conditioning in random samples and observe instructions. Looking ahead, and denoting by $[\![P]\!]^{q^k}$ the semantics of the program P inside $\mathbb{F}_{q^k}$, we will have that $[\![r_1, \ldots, r_m \xleftarrow{\$} \{X \in \mathbb{F}^m \mid b\}; e]\!]^{q^k} = [\![r_1, \ldots, r_m \xleftarrow{\$} \mathbb{F}^m; \text{if } b \text{ then } e \text{ else } \bot]\!]^{q^k}$ and $[\![\text{observe } b; e]\!]^{q^k} = [\![\text{if } b \text{ then } e \text{ else } \bot]\!]^{q^k}$. Without loss of generality, we can inline deterministic assignments, and use code motion to perform all samplings eagerly, i.e., all random samplings are performed upfront. Therefore we can simply consider that each variable in R is implicitly uniformly sampled in $\mathbb{F}_{q^k}$. Programs are then tuples of simplified expressions $(e_1, \ldots, e_n)$ defined as follows.

$e ::=$	*simplified expressions*
$\mid P$	polynomial
$\mid \bot$	failure
$\mid$ if b then e_1 else e_2	conditional branching

We suppose that all nested tuples are flattened and write (P, Q) to denote the program which simply concatenates the outputs of P and Q. When clear from the context, we may also simply write $\vec{0}$ instead of the all zero tuple $(0, \ldots, 0)$. We denote by $\overline{\mathcal{P}}_q(I, R)$ the set of arithmetic programs, that are simply tuples of polynomials. Remark that arithmetic programs cannot fail.

One may note that the translation from the surface language to the core language is not polynomial in general. Indeed, constructs of the form (if b then $x := t_1$ else $x := t_2; P$), i.e. sequential composition after a conditional, implies to propagate the branching over the assignment to all branches of P, and doubles the number of conditional branchings of P. All complexity results will be given for the size of the program given inside the core language. Remark that in a functional style version of the surface language, where we replace $x := t$ by let $x = t$ in and removed sequential composition, the translation would however be polynomial. Similarly, for the class of programs without sequential composition after conditional branchings, the translation is also polynomial.

2.3 Semantics

We now define the semantics of our core language. The precise translation from the high level syntax previously presented and our core language is standard and omitted.

Deterministic semantics. We first define a *deterministic* semantics where all random samplings have already been defined. For a set X of variables, with $P \in \mathbb{F}_q[X]$ and $\vec{x} \in \mathbb{F}_{q^k}^{|X|}$, $P(\vec{x})$ classicaly denotes the evaluation of P inside $\mathbb{F}_{q^k}$. We also denote $b(\vec{v})$ the evaluation of a boolean test, where all polynomials are evaluated according to $\vec{v}$. For a program

$e \in \mathcal{P}_q(I,R)$ and $\vec{v} \in \mathbb{F}_{q^k}^{|I \cup R|}$, we define a natural evaluation of e, denoted $[e]_{\vec{v}}^{q^k}$, which is a value inside $\mathbb{F}_{q^k}^{|P|} \times \{\bot\}$:

$$
\begin{aligned}
[P]_{\vec{v}}^{q^k} &= P(\vec{v}) \text{ where } P \in \mathbb{F}_q[I \uplus R] \\
[\bot]_{\vec{v}}^{q^k} &= \bot \\
\left[\begin{array}{l}\text{if } b \text{ then } e_1 \\ \text{else } e_2\end{array}\right]_{\vec{v}}^{q^k} &= \begin{cases} [e_1]_{\vec{v}}^{q^k} & \text{if } b(\vec{v}) \text{ holds on } \mathbb{F}_{q^k} \\ [e_2]_{\vec{v}}^{q^k} & \text{else} \end{cases} \\
[(e_1, \ldots, e_n)]_{\vec{v}}^{q^k} &= \begin{cases} \bot \text{ if } [e_i]_{\vec{v}}^{q^k} = \bot \text{ for some } i \\ ([e_1]_{\vec{v}}^{q^k}, \ldots, [e_n]_{\vec{v}}^{q^k}) \text{ else} \end{cases}
\end{aligned}
$$

Intuitively, the set of executions corresponding to non failure executions represent the set of possible executions of the program. We next define probabilistic semantics by sampling uniformly the valuations of the random variables while conditioning on the fact that the program does not fail.

Probabilistic semantics. For any n, the set of distributions over $\mathbb{F}_q^n$ is denoted by $\mathrm{Distr}(\mathbb{F}_q^n)$. For a program $P \in \mathcal{P}_q(I,R)$ with $|P| = n$, and $|I| = m$, we define its semantics to be a function from inputs to a distribution over the outputs:

$$[\![P]\!]^{q^k} : \mathbb{F}_{q^k}^m \to \mathrm{Distr}(\mathbb{F}_{q^k}^n)$$

We assume that programs inside $P \in \mathcal{P}_q(I,R)$ do not fail all the time, i.e., for any possible input and any program its probability of failure is strictly less than 1. For program P, input $\vec{i} \in \mathbb{F}_{q^k}^m$ and output $\vec{o} \in \mathbb{F}_{q^k}^n$ we set $\mathbb{P}\{\vec{x} = \vec{o} \mid \vec{x} \xleftarrow{\$} [\![P]\!]^{q^k}(\vec{i})\}$ to

$$\frac{\mathbb{P}\{[P]_{\vec{i},\vec{r}}^{q^k} = \vec{o} \mid \vec{r} \xleftarrow{\$} \mathbb{F}_{q^k}^{|R|}\}}{\mathbb{P}\{[P]_{\vec{i},\vec{r}}^{q^k} \neq \bot \mid \vec{r} \xleftarrow{\$} \mathbb{F}_{q^k}^{|R|}\}}$$

Note that the normalization by conditioning on non-failing programs is well defined as we supposed that programs do not always fail.

3 The fixed case

We start by studying the complexity of several problems over a given finite field. In this case we only manipulate finite objects, and hence all problems are obviously decidable, by explicitly computing the distributions. We however provide precise complexity results and show that these problems have complexities in the counting hierarchy [31]. We also define the universal variant and state some results that are common to both variants of the problems.

3.1 Conditional equivalence

In this section, we prove that for any $k \in \mathbb{N}$, the q^k-*equivalence* problem is $\mathrm{coNP}^{\mathrm{C_=P}}$-complete. To this end, we introduce a technical generalization of the equivalence problem, that we call q^k-*conditional equivalence*, and we proceed in four steps, showing that:

1. without loss of generality, we can consider programs without inputs; (Lemma 3.2)
2. verifying if the conditioned distributions of two inputless programs coincide on a fixed point is in $\mathrm{C_=P}$; (Lemma 3.3)
3. verifying if the conditioned distribution of inputless programs coincide on all points is in $\mathrm{coNP}^{\mathrm{C_=P}}$; (Corollary 3.4)
4. and finally, even equivalence for programs over $\mathbb{F}_2$ is $\mathrm{coNP}^{\mathrm{C_=P}}$-hard. (Lemma 3.5)

3.1.1 Defining conditional equivalence.

q^k-*conditional equivalence* is a generalization of equivalence, where we require programs to be equivalent when the distributions are conditioned by some other program being equal to zero. Conditional equivalence is a technical generalisation, that is interesting because it is self-reducible when removing for instance the conditionals.

Definition 3.1 (q^k-*conditional equivalence*)**.** Let $P_1, Q_1 \in \mathcal{P}_q(I,R)$ and $P_2, Q_2 \in \overline{\mathcal{P}}_q(I,R)$ with $|P_1| = |Q_1| = n$. We denote $P_1 \mid P_2 \approx_{q^k} Q_1 \mid Q_2$, if:

$$\forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ \forall \vec{c} \in \mathbb{F}_{q^k}^n.\ [\![(P_1, P_2)]\!]_{\vec{i}}^{q^k}(\vec{c}, \vec{0}) = [\![(Q_1, Q_2)]\!]_{\vec{i}}^{q^k}(\vec{c}, \vec{0})$$

The universal version q^∞-*conditional equivalence* is defined similarly to q^∞-*equivalence* . Note that conditional equivalence is a direct generalization of equivalence, as for $P, Q \in \mathcal{P}_q(I,R)$, $P \approx_{q^k} Q$ if and only if $P \mid 0 \approx_{q^k} Q \mid 0$.

We also remark that equivalence over $\mathbb{Z}$ is undecidable, which is a consequence of Hilbert's 10th problem, as a polynomial over randomly sampled variables will be equivalent to zero if and only if it does not have any solutions.

We first define precisely the decision problems associated to our questions, for $k \in \mathbb{N} \cup \{\infty\}$:

q^k-*conditional equivalence*
INPUT: $P_1, Q_1 \in \mathcal{P}_q(I,R)$, $P_2, Q_2 \in \overline{\mathcal{P}}_q(I,R)$. $P_1 \mid P_2 \approx_{q^k} Q_1 \mid Q_2$?

The decision problem for q^k-*equivalence* simply corresponds to q^k-*conditional equivalence* with P_2 and Q_2 being equal to 0. In the following we will show that both problems are interreducible, and that q^k-*equivalence* and q^k-*conditional equivalence* are both $\mathrm{coNP}^{\mathrm{C_=P}}$-complete.

3.1.2 Complexity results.

Recall that $\mathrm{C_=P}$-complete is the set of decision problems solvable by a NP Turing Machine whose number of accepting paths is equal to the number of rejecting paths. halfSAT is the natural $\mathrm{C_=P}$-complete problem: is a CNF boolean formula ϕ satisfied for exactly half of its valuations. $\mathrm{coNP}^{\mathrm{C_=P}}$ is the set of decision problems whose complement can be solved by a NP Turing Machine with

access to an oracle deciding problems in $\mathsf{C_{=}P}$. The canonical $\mathsf{coNP}^{\mathsf{C_{=}P}}$ problem is (using the results from [30, Sec. 4] and [22]) A–halfSAT: given a CNF boolean formula $\phi(X, Y)$, for all valuations of X is the formula satisfied for exactly half of the valuations of Y.

Also, recall that conditional equivalence is a direct generalization of equivalence. We thus trivially have, for any $k \in \mathbb{N} \cup \{\infty\}$, that q^k-*equivalence* reduces in polynomial time to q^k-*conditional equivalence*.

We first study the complexity of deciding if the distributions of two programs are equal on a specific point. To do so, we remark that it is not necessary to consider inputs when considering equivalence or conditional equivalence. The intuition is that inputs can be seen as random values, that must be synchronized on both sides. This synchronization is achieved by explicitly adding these random variables to the output, forcing them to have the same value on both side. The following Lemma is a generalization to conditional equivalence of a Lemma from [4].

Lemma 3.2. *For any $k \in \mathbb{N} \cup \{\infty\}$, q^k-*conditional equivalence *reduces to q^k-*conditional equivalence *restricted to programs without inputs in polynomial time.*

Omitted proofs can be found in [6]. As we can without loss of generality ignore the inputs, we study the complexity of deciding if the distributions of two inputless programs coincide on a specific point. To this end, we build a Turing Machine, such that it will accept half of the time if and only if the programs given as input have the same probability to be equal to some given value. Essentially, it is based on the fact that over $\mathbb{F}_2$, if $r = 0$ then P else $(Q + 1) \approx_2 r$ if and only if $P \approx_2 Q$.

Lemma 3.3. *Let $P_1, Q_1 \in \mathcal{P}_q(\emptyset, R)$ and $P_2, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$ with $|P_1| = |Q_1| = n$. For any $\vec{c} \in \mathbb{F}_{q^k}^n$, we can decide in* $\mathsf{C_{=}P}$ *if:*

$$[\![(P_1, P_2)]\!]^{q^k}(\vec{c}, \vec{0}) = [\![Q_1, Q_2]\!]^{q^k}(\vec{c}, \vec{0})$$

As $\mathsf{C_{=}P}$ is closed under finite intersection [30], we can decide in $\mathsf{C_{=}P}$ if two distributions over a set of fixed size are equal, by testing the equality over all points. When we only consider inputless programs of fixed arity, the set of points to test is constant, and the equivalence problem is in $\mathsf{C_{=}P}$ (see [6] for details). However, when we extend to inputs, or to programs of variable arity, we need to be able to check for all possible points if the distribution are equal over this point. (Note that our encoding that allows to only consider inputless programs increases the arity.) Checking all possible points is typically in coNP. We thus obtain that:

Corollary 3.4. *q^k-*equivalence *and q^k-*conditional equivalence *are in* $\mathsf{coNP}^{\mathsf{C_{=}P}}$ *for any $k \in \mathbb{N}$.*

To conclude completeness for both q^k-*equivalence* and q^k-*conditional equivalence*, it is sufficient to show the hardness of 2-*equivalence*, which we do by reducing A–halfSAT. We simply transform a CNF boolean formula into a polynomial over $\mathbb{F}_2$, such that the polynomial is uniform if the formula is in A–halfSAT. This is a purely technical operation (see [6]).

Lemma 3.5. 2-equivalence *is* $\mathsf{coNP}^{\mathsf{C_{=}P}}$*-hard.*

The results about q^k-*conditional equivalence* naturally translate to the independence problem: are the distribution of multiple programs independent? We obtain the same complexity class by proving that the problems are interreducible. These results are detailed in [6].

3.2 Majority

The goal of this section is to show that the majority problem is $\mathsf{coNP}^{\mathsf{PP}}$-complete. To this end, we study the complexity of q^k-0-*majority*, showing:

- PP-completeness for inputless programs;
- $\mathsf{coNP}^{\mathsf{PP}}$-completeness in general.

The proof in both cases uses similar ideas as for equivalence. Note that we actually use the same Turing Machine for the Membership. As both complexity classes are closed under finite intersection, it yields the complexity of q^k-*majority*, which can be decided using q^k times q^k-0-*majority*.

3.2.1 The majority problem.

q^k-*majority* asks if, given two programs, the quotient of their distribution is bounded on all points by some rational r. q^k-0-*majority* is a subcase, where we only ask if the quotient of their distribution is bounded on a single point. This problem allows to estimate the distance between two distributions. It is close to the differential privacy question, which asks, when $\delta = 0$, if the quotient of two distributions is bounded over all points by some e^{ϵ}.

We observe that the majority problem is harder than equivalence, as majority for $r = 1$ implies equivalence. An important difference between equivalence and majority is that the presence of inputs actually changes the complexity of the majority problem.

Let us define the decision problem associated to q^k-*majority*, with $k \in \mathbb{N} \cup \{\infty\}$:

q^k-*majority*
INPUT: $P, Q \in \mathcal{P}_q(I, R)$, $r \in \mathbb{Q}$. $P \prec^r_{q^k} Q$?

We consider that r is given in input as two integers written in unary. Essentially, this is because if one whishes to encode any r, it requires an exponential blow up, but in practice, we tend to use some particular rationals such as $r = q^l$, for which there is no exponential blow up.

3.2.2 Complexity results.

We recall that PP is the set of languages accepted by a polynomial time non-deterministic Turing Machine where the acceptance condition is that a majority of paths are accepting. A natural PP-complete problem

is MAJSAT: is a boolean CNF formula satisfied for at least half of its valuations. coNP$^{\mathsf{PP}}$ is the class of of problems whose complement is decided by a NP Turing Machine with access to an oracle deciding problems in PP. The classical NP$^{\mathsf{PP}}$ problem is E−MAJSAT [22] , which given a CNF boolean formula $\phi(X, Y)$, asks if there is a valuation of X such that $\phi(X, Y)$ is satisfied for at least half of the valuation of Y.

The complexity of q^k-0-*majority* over inputless programs is derived in a similar way as for equivalence. (j'ai remplacé un paragraphe par ça) The only difficulty is that we are comparing with a rational. We thus briefly show how one can assume without loss of generality that $r = 1$ (in which case we omit r from the notation). The idea is , given $r, s \in \mathbb{N}$, that $P \prec^{\frac{r}{s}}_{q^k} Q \Leftrightarrow (P, T_r) \prec_{q^k} (Q, T_s)$, if T_j is a machine which is equal to zero with probability $\frac{1}{j}$.

Lemma 3.6. *For any $k \in \mathbb{N}$, q^k-0-*majority *reduces in polynomial time to q^k-0-*majority *with $r = 1$.*

The proof showing that q^k-0-*majority* is in PP is similar to proving that testing if two distributions are equal over a point is in C$_=$P. We prove PP-completeness by deriving the hardness from MAJSAT.

Lemma 3.7. *For any $k \in \mathbb{N}$, q^k-0-*majority *restricted to inputless programs is* PP*-complete.*

Finally, as PP is closed under finite intersection, we also get that q^k*-majority* over inputless programs with a fixed arity is PP-complete.

Let us now turn to the general version, for programs with inputs. By using some fresh inputs variables, let us remark that one can easily reduce q^k*-majority* to q^k-0-*majority*. Indeed, for $P, Q \in \mathcal{P}_q(I, R)$ and $c \in \mathbb{F}_q^{|P|}$, with a fresh $x \in I$:

$$\forall \vec{i} \in \mathbb{F}_{q^k}^{|I|}.\ [\![P]\!]^{q^k}_{\vec{i}}(c) \leq r[\![Q]\!]^{q^k}_{\vec{i}}(c)$$
$$\Leftrightarrow$$
$$(P - x) \prec^{r}_{q^k} (Q - x)$$

We show that q^k*-majority* is coNP$^{\mathsf{PP}}$ complete, and thus is most likely[3] harder than its version without inputs. The membership and hardness proofs are similar to the equivalence problem when going from C$_=$P to coNP$^{\mathsf{C_=P}}$.

Lemma 3.8. *q^k-*majority *is* coNP$^{\mathsf{PP}}$ *complete.*

4 The universal case

In this section we first give some general insights on universal equivalence showing important differences with the case of a fixed field. Then we provide our main decidability result, first for arithmetic programs, then arithmetic programs enriched with conditionals, and finally for general programs. We continue by studying two other problems in the universal case: simulatability and 0-majority.

[3] As PH $\subset$ coNP$^{\mathsf{PP}}$, PP = coNP$^{\mathsf{PP}}$ would imply PH $\subset$ PP which is commonly believed to be false.

4.1 General remarks

In this section we try to provide some insights on the difficulty of deciding q^∞*-equivalence*. First of all, we note that equivalence and universal equivalence do *not* coincide.

Example 4.1. The program $x^2 + x$ and the program 0 are equivalent over $\mathbb{F}_2$ (they are then both equal to zero), but not over $\mathbb{F}_4$.

In the case of a given finite field, equivalence can be characterized by the existence of a bijection, see for instance [4]. We denote by $\mathsf{bij}^{\mathbb{F}_q^m}$ the set of bijections over $\mathbb{F}_q^m$. Any element $\sigma \in \mathsf{bij}^{\mathbb{F}_q^m}$ can be expressed as a tuple of polynomials (see e.g. [27]), and can be applied as a substitution. The characterization can then be stated as follows, where we denote by $=_{\mathbb{F}_q}$ equality between polynomials modulo the rule of the field (i.e., $X^q = X$).

$$P \approx_q Q \Leftrightarrow \exists \sigma \in \mathsf{bij}^{\mathbb{F}_q^m}, P =_{\mathbb{F}_q} Q\sigma$$

However, there are universally equivalent programs such that there does *not* exist a universal σ suitable for all extensions.

Example 4.2. Consider, $P = xy+yx+zx$, with $\sigma : (x, y, z) \mapsto (x, y + x, z + x)$, we get that $P \approx_{2^\infty} x^2 + yz$. Now, $x \mapsto x^2$ is a bijection over all $\mathbb{F}_{2^k}$, so we also have $P \approx_{2^\infty} x + yz$ and finally $P \approx_{2^\infty} x$.

But here, a bijection between $x^2 + yz$ and x must use the inverse of x^2 whose expression depends on the size of the field. Thus, there isn't a universal polynomial σ which is a bijection such that on all $\mathbb{F}_{2^k}$, $P =_{\mathbb{F}_{2^k}} Q \circ \sigma$.

Nevertheless, we can note that for linear programs this characterization allows us to show that *q-equivalence* and q^∞*-equivalence* are equivalent. Intuitively, the bijection allowing to obtain the equality between two linear programs is also a bijection valid for all extensions of the finite field, as the bijection is linear, and is thus a witness of equivalence over all extensions. For linear programs, there exists a polynomial time decision procedure for equivalence, and hence for universal equivalence.

Lemma 4.3. *q^∞-*equivalence *restricted to linear programs is in* PTIME.

Moreover, building on results from [23] on Tame automorphisms, we can use the above characterization to design a sufficient condition which implies universal equivalence for general programs. Even though not complete this sufficient condition may be useful to verify universal equivalence more efficiently in practice. These results are detailed in [6].

4.2 Decidability of universal equivalence

We show decidability of q^∞*-equivalence*, leveraging tools from algebraic geometry, showing that:

1. q^∞*-conditional equivalence* is decidable for arithmetic programs; (Lemma 4.5)

2. it is also decidable for programs with conditionals; (Lemma 4.7)
3. it is finally decidable for programs with conditioning, e.g. failures. (Lemma 4.8)

We first recall the definition and relevant properties of local zeta Riemann functions. For a tuple P of polynomials $P_1, \dots, P_m \in \mathbb{F}_q[X_1, \dots, X_n]$, the local zeta Riemann function over T is the formal series

$$Z(P,T) = \exp\left(\sum_{k\in\mathbb{N}^*} \frac{|N_k(P)|}{k} T^k\right)$$

where $N_k(P) = \{\vec{x} \in \mathbb{F}_{q^k}^n \mid \bigwedge_{1\le i\le m} P_i(\vec{x}) = 0\}$. Weil's conjecture [33] states several fundamental properties of local zeta Riemann functions over algebraic varieties. Dwork [12] proves part of Weil's conjecture stating that the local zeta Riemann functions over algebraic varieties is a rational function with integer coefficients—recall that $Z(T)$ is a rational function iff there exist polynomials $R(T)$ and $S(T)$ such that $Z(T) = R(T)/S(T)$. Bombieri [8] shows that the sum of the degrees of R and S is upper bounded by $4(d+9)^{n+1}$, where d is the total degree of $(P_1, \dots, P_m)$. It follows that the values of N_k for $k \le 4(d+9)^{n+1}$ suffice for computing Z; since these values can be computed by brute force, this yields an algorithm for computing Z. We will by abuse of notations write $Z(P)$ instead of $Z(P,T)$ for the local zeta function of P. $Z(P)$ completely characterizes the number of times P is equal to zero on all the different extensions. For instance, $Z(P) = Z(Q)$ allows us to conclude that P and Q always evaluate to zero for the same number of valuations, and this over any $\mathbb{F}_{q^k}$. As Z can effectively be computed [19], we can use it to decide q^∞-*equivalence*.

Notice that, given two programs P and Q, the local zeta function directly allow us to conclude if they are equal to some value with the same probability for all extensions of the base field. Moreover, thanks to [18], the computability of the local zeta function can be extended from counting the number of points such that $P = 0$ for a tuple of polynomials, to counting the number of points such that ϕ holds, where ϕ is an arbitrary first order formula over finite fields.

Corollary 4.4. *Let ϕ and ψ be two first order formulae built over atoms of the form $P = 0$ with $P \in \mathbb{F}_q[X]$, and with free variables $F \subset X$. One can decide if for all $k \in \mathbb{N}$:*

$$\left|\{\vec{f} \in \mathbb{F}_{q^k}^{|F|} \mid \phi(\vec{f}) = 1\}\right| = \left|\{\vec{f} \in \mathbb{F}_{q^k}^{|F|} \mid \psi(\vec{f}) = 1\}\right|$$

Thus, for any two events which can be expressed as a first order formula over finite field one can verify if they happen with the same probability over all extensions of the base field. Remark that this cannot be used to decide universal equivalence, as equivalence cannot be expressed inside a first order formula.

We first show that, thanks to the local zeta functions, q^∞-*equivalence* is decidable for arithemic programs, i.e programs without conditionals or conditioning.

Lemma 4.5. *Let $P_1, P_2, Q_1, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$.*

$$\begin{array}{ccc} P_1 \mid P_2 & & Z((Q_1 - Q_1\sigma, Q_2, Q_2\sigma)) \\ \approx_{q^\infty} & \Leftrightarrow & = Z((P_1 - Q_1\sigma, P_2, Q_2\sigma)) \\ Q_1 \mid Q_2 & & = Z((P_1 - P_1\sigma, P_2, P_2\sigma)) \end{array}$$

where $\sigma : R \mapsto R'$ maps each variable to a fresh one.

Proof. We assimilate $P_1P_2, Q_1, Q_2 \in \overline{\mathcal{P}}_q(\emptyset, R)$ of size m with polynomials, denoting $P(X)$ the value of P given $X \in \mathbb{F}_{q^k}^m$. Given an enumeration $1 \le j \le s$ of the elements c_j of $\mathbb{F}_{q^k}^n$, for any programs T, T', we let

$$(T,T')_i^k = \left|\{X \in \mathbb{F}_{q^k}^m \mid T(X) = c_i \wedge T'(X) = \vec{0}\}\right|$$

Then, if we denote $\overrightarrow{(T,T')^k} = (T,T')_1^k, \dots, (T,T')_s^k$, that characterizes the distribution of $T|T'$, as $P_1 \mid P_2 \approx_{q^\infty} Q_1 \mid Q_2$ if and only if $\forall k \in \mathbb{N}.\ \overrightarrow{(P_1,P_2)^k} = \overrightarrow{(Q_1,Q_2)^k}$. Using the classical inner product $\vec{x}\cdot\vec{y} = \sum_i x_i y_i$, for any k and programs $U, V, U', H' \in \overline{\mathcal{P}}_q$, we have:

$$\begin{aligned} & N_k((U - V\sigma, U', V')) \\ &= \left|\left\{X, X' \in \mathbb{F}_{q^k}^m \mid \begin{array}{r} U(X) = V(X') \\ \wedge (U'(X), V'(X)) = \vec{0} \end{array}\right\}\right| \\ &= \textstyle\sum_{c\in\mathbb{F}_{q^k}^n} \left|\{X \in \mathbb{F}_{q^k}^m \mid U(X) = c \wedge U'(X) = \vec{0}\}\right| \\ &\qquad \times \left|\{X \in \mathbb{F}_{q^k}^m \mid V(X) = c \wedge V'(X) = \vec{0}\}\right| \\ &= \textstyle\sum_i (U,U')_i^k (V,V')_i^k \\ &= \overrightarrow{(U,U')^k} \cdot \overrightarrow{(V,V')^k} \end{aligned}$$

Using scalar operations, we have that:

$$\begin{aligned} & N_k(Q_1 - Q_1\sigma, Q_2, Q_2\sigma) = N_k(P_1 - Q_1\sigma, P_2, Q_2\sigma) \\ &\qquad = N_k(P_1 - P_1\sigma, P_2, P_2\sigma) \\ \Leftrightarrow\ & \overrightarrow{(Q_1,Q_2)^k} \cdot \overrightarrow{(Q_1,Q_2)^k} = \overrightarrow{(P_1,P_2)^k} \cdot \overrightarrow{Q^k} \\ &\qquad = \overrightarrow{(P_1,P_2)^k} \cdot \overrightarrow{(P_1,P_2)^k} \\ \Leftrightarrow\ & (\overrightarrow{(P_1,P_2)^k} - \overrightarrow{Q^k}) \cdot (\overrightarrow{(P_1,P_2)^k} - \overrightarrow{(Q_1,Q_2)^k}) = \vec{0} \\ \Leftrightarrow\ & \overrightarrow{(P_1,P_2)^k} = \overrightarrow{(Q_1,Q_2)^k} \end{aligned}$$

Hence,

$$\begin{aligned} & \forall k \in \mathbb{N}. \forall c \in \mathbb{F}_{q^k}^n. \\ &\qquad \left|\{X \in \mathbb{F}_{q^k}^m \mid P_1(X) = c \wedge P_2(X) = \vec{0}\}\right| \\ &\qquad = \left|\{X \in \mathbb{F}_{q^k}^m \mid Q_1(X) = c \wedge Q_2(X) = \vec{0}\}\right| \\ &\qquad\qquad \Leftrightarrow \\ & \forall k \in \mathbb{N}. \quad N_k((Q_1 - Q_1\sigma, Q_2, Q_2\sigma)) \\ &\qquad = N_k((P_1 - Q_1\sigma, P_2, Q_2\sigma)) \\ &\qquad = N_k((P_1 - P_1\sigma, P_2, P_2\sigma)) \end{aligned}$$

This concludes the proof, as for all U, V,

$$Z(U) = Z(V) \Leftrightarrow \forall k.\ N_k(U) = N_k(V)$$

□

In [6], we provide a variant of this result for the specific case of verifying if a program follows the uniform distribution over all extensions, where only one computation of a local zeta function is required.

Using the complexity for the computation of the local zeta function provided by [19, Corollary 2] we obtain the following corollary.

Corollary 4.6. q^∞-equivalence *and* q^∞-conditional equivalence *restricted to arithmetic programs are in* EXP.

We now wish to remove conditionals, in order to reduce equivalence for programs with conditional to arithmetic programs (which are simply tuples of polynomials). To remove the conditionals, the first idea is to use a classical encoding inside finite fields: $[\![\text{if } B \neq 0 \text{ then } P_1^t \text{ else } P_1^f]\!]^{q^k} = [\![P_1^f + B^{q^k-1}(P_1^t - P_1^f)]\!]^{q^k}$. This works nicely as B^{q^k-1} is equal to 0 if $B = 0$, else to 1. However, for the universal case, we need to have an encoding which does not depend on the size of the field, i.e., it must be independent of k. The key idea is that for any variable t and polynomial B:

$$(B(Bt-1) = 0 \wedge t(Bt-1) = 0) \Leftrightarrow t = B^{q^k-2}$$

We can then encode conditionals as a multiplication by some fresh variable t, where t is conditioned by the previous conditions. An induction on the number of conditionals yields our second lemma.

Lemma 4.7. *For any* $k \in \mathbb{N} \cup \{\infty\}$, q^k-conditional equivalence *restricted to programs without failures reduces in exponential time to* q^k-conditional equivalence *restricted to arithmetic programs.*

Recall that failures define the probablisitic semantics by normalization. And for instance, for some program (if $b = 0$ then P_1 else $\perp, P_2$) where P_1 and P_2 do not fail and b is a polynomial, for any $\vec{c}$, we have:

$$[\![(\text{if } b = 0 \text{ then } P_1 \text{ else } \perp, P_2)]\!]^{q^k}(\vec{c}, \vec{0}) = \frac{\mathbb{P}\{P_1 = \vec{c} \wedge P_2 = \vec{0} \wedge b = 0\}}{\mathbb{P}\{\neg(b=0)\}}$$

Handling this division by itself would be difficult if we wanted to compute the distribution. However, in our setting, we are comparing the equality of two distributions, so we can simply multiply on both side by the denominator, and try to express once again all factors as an instance of conditional equivalence. We will be able to push inside conditional equivalence some probabilities, as $[\![P]\!]^{q^k}(\vec{c}) \times \mathbb{P}\{b = 0\} = [\![P, b]\!]^{q^k}(\vec{c}, 0)$ when all variables in b do not appear in P.

As an illustration of how to remove the failures, with some program Q, we have:

$$\text{if } b = 0 \text{ then } P_1 \text{ else } \perp \mid P_2 \approx_{q^k} Q \mid 0$$
$$\Leftrightarrow \forall \vec{c}. [\![P_1, P_2, b]\!]^{q^k}(\vec{c}, \vec{0}) = \mathbb{P}\{\neg(b = 0)\}[\![Q]\!]^{q^k}(\vec{c})$$

To reduce to an instance of conditional equivalence, the issue is that we need to express as an equality the disequality $b \neq 0$. With some fresh variable t, multiplying by $\mathbb{P}\{\neg(b = 0)\}$ or conditioning on $tb - 1 = 0$ is equivalent, as b has an inverse if and only if it is different from zero. We can thus have:

$$\text{if } b = 0 \text{ then } P_1 \text{ else } \perp \mid P_2 \approx_{q^k} Q \mid 0$$
$$\Leftrightarrow P_1 \mid P_2, b \approx_{q^k} Q \mid tb - 1$$

Using those techniques, we obtain:

Lemma 4.8. *For any* $k \in \mathbb{N} \cup \{\infty\}$, q^k-conditional equivalence *reduces to* q^k-conditional equivalence *restricted to programs without failures in exponential time.*

The previous Lemmas allows us to conclude.

Theorem 4.9. q^∞-equivalence *and* q^∞-conditional equivalence *are in* 2-EXP.

And using once again [6], we obtain the same complexity results for the independence problem.

Corollary 4.10. q^∞-conditional independence *is in* 2-EXP.

Moreover, we can also extend the lower bound obtained for *q-equivalence.*

Lemma 4.11. q-equivalence *reduces in polynomial time to* q^∞-equivalence.

4.3 Bounded Universal Simulatability

Simulation-based proofs [21] are one main cornerstone of cryptography. Informally, simulation-based proofs consider a real and an ideal world, and require showing the existence of a simulator, such that no adversary can distinguish the composition of the simulator and of the ideal world from the real world. This can be modelled in our context by requiring the existence of a program S (the simulator) such that "plugging in" the ideal world into S is equivalent to the real world. In this section, we consider a simpler task, where the size of the simulator is bounded. Given a program C, we denote $\deg(C)$ the maximum degree of a progam, i.e the maximum degree of any polynomial appearing in C (the degree of a polynomial is the maximum over the sum of the degrees of each monomial).

Definition 4.12. Let $P, Q \in \mathcal{P}_q(I, R)$, R' such that $\sharp R = \sharp R'$ and $l \in \mathbb{N}$. We denote $P \sqsubseteq^l_{q^{[\infty]}} Q$, if there exists $S \in \mathcal{P}_q(\{i_1, \ldots, i_n\}, R'\})$ such that $\deg(S) \leq l$, and $S[^Q/_{\vec{i}}] \approx_{q^{[\infty]}} P$.

The associated decision problem is:

l,q-*simulatability*
INPUT: $P, Q \in \mathcal{P}_q(I, R)$. $P \sqsubseteq^l_{q^\infty} Q$?

Thanks to the bound on the degree coming from l, we can easily obtain a bound on the number of such possible contexts. This is shown in [6]. From the bound on the number of contexts and the decidability of universal equivalence, one can derive the decidability of bounded simulatability.

Theorem 4.13. *l,q*-simulatability *is decidable.*

As a lower bound, we prove that *l, q-simulatability* is as hard as universal equivalence:

Lemma 4.14. *For any* $l \in \mathbb{N}$, $k \in \mathbb{N} \cup \{\infty\}$, q^k-equivalence *reduces in polynomial time to* l, q^k-simulatability.

We conclude this section by noting that our notion of bounded simulatability is more restricted than the general paradigm of simulation-based proofs but could be a good starting point for automating simulation-based proofs.

4.4 Universal zero-majority without inputs

For arbitrary programs, we reduce q^∞-0-*majority* to the POSITIVITY problem. We recall that a Linear Recurrence Sequence (LRS) is an infinite sequence of reals $u = u_1, u_2, \ldots$ such that there exist real constants $a_1, \ldots, a_k$ such that for all $n \geq 0$,

$$u_{n+k} = a_1 u_{n+k-1} + \cdots + a_k u_n$$

The order of a LRS $u = u_1, u_2, \ldots$ is the smallest k such that the equation above holds. A LRS $u = u_1, u_2, \ldots$ is positive if $u_n \geq 0$ for every $n \in \mathbb{N}$. The positivity problem consists in deciding whether a LRS is positive.

We use the fact that from a local zeta function, which is rational, we can obtain a Linear Recurrence Sequence. Then, by considering the POSITIVITY of the LRS obtained by susbtracting two local zeta function, we actually check if the coefficients of the first one are always greater than the second one. We remark that the complexity of the problem strongly relies on the presence of multiplications, as for q^∞*-equivalence*. Indeed, in the linear case, majority implies equivalence and we obtain the following.

Lemma 4.15. q^∞*-0*-majority *restricted to linear programs is in* PTIME.

The general case has yet to be proven decidable.

Theorem 4.16. q^∞-0-majority *for inputless programs reduces in exponential time to* POSITIVITY.

Proof. Let $P, Q \in \overline{\mathcal{P}}_q(\emptyset, R)$. We assume without loss of generality that we only have to consider arithmetic programs, using the same simplifications for observe and conditionals as we did for universal equivalence.

Recall that for any P, the local zeta function $Z(P)$ (over indeterminate T) is rational thanks to Weil's conjecture [12]. Moreover, with $N_k(P) = \left|\{X \in \mathbb{F}_{q^k}^m \mid P(X) = 0\}\right|$, we have that:

$$\frac{d}{dT} log(Z(P)) = \frac{Z'(P)}{Z(P)} = \sum_k N_k(P) T^k$$

Let us call $\tilde{Z}(P) = \sum_k N_k(P) T^k$, which is also a rational function as Z is (and so is Z'). As the coefficients of $\tilde{Z}(P) - \tilde{Z}(Q)$ are $N_k(P) - N_k(Q)$, we have that $\forall k, N_k(P) \geq N_k(Q)$ if and only if $\tilde{Z}(P) - \tilde{Z}(Q)$ only has positive coefficients. It is well known that the coefficients of the Taylor serie of a rational function form a LRS (see e.g. [13]). This means that the coefficients of $\tilde{Z}(P) - \tilde{Z}(Q)$ form an LRS, which we denote z^{PQ}. We finally get that $Q \prec_{q^\infty} P$ if and only if $\forall n, z_n^{PQ} \geq 0$. □

This reduction can also be applied with the generalization of [18], and thus, for any two events about programs over finite fields, one can, given an oracle for the POSITIVITY problem, decide if the probability of the first event is greater than the second one for all extensions of the base field.

Similarly to the equivalence case, we can derive some hardness from the non universal case, but we do not obtain any completeness result.

Lemma 4.17. 2^∞-0-majority *is* PP*-hard.*

5 Conclusion

We have introduced universal equivalence and majority problems and studied their complexity and decidability. Our work could notably be used as a building block to design a decidable logic for universal probabilistic program verification. It leaves several questions of interest open:

- the exact complexity of universal equivalence is open. It is even unknown whether the universal problem strictly harder than the non-universal one;
- the decidability of universal majority is open. The decidability of POSITIVITY would yield decidability of universal 0-majority and equivalently, undecidability of universal majority would also solve negatively the POSITIVITY problem;
- the decidability of universal approximate equivalence is open. Approximate equivalence asks whether the statistical distance between the distributions of two programs is negligible in k. This notion has direct applications in provable security.

Acknowledgements. We wish to thank the anonymous reviewers for their useful comments, as well as Martin Grohe, Bruce Kapron, David Mestel, Joël Ouaknine, Pierre-Jean Spaenlehauer, Emmanuel Thomé and Mehdi Tibouchi for interesting discussions. We are grateful for the support by the ERC under the EU's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC) as well as ONR (grant N000141512750).

References

[1] James Ax. 1968. The elementary theory of finite fields. *Annals of Mathematics* 88, 2 (1968), 239–271.

[2] Gilles Barthe, Rohit Chadha, Vishal Jagannath, A. Prasad Sistla, and Mahesh Viswanathan. 2019. Automated Methods for Checking Differential Privacy. *CoRR* abs/1910.04137 (2019). arXiv:1910.04137 http://arxiv.org/abs/1910.04137

[3] Gilles Barthe, Marion Daubignard, Bruce Kapron, Yassine Lakhnech, and Vincent Laporte. 2010. On the equality of probabilistic terms. In

16th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'10) (Lecture Notes in Computer Science), Vol. 6355. Springer, 46–63.

[4] Gilles Barthe, Benjamin Grégoire, Charlie Jacomme, Steve Kremer, and Pierre-Yves Strub. 2019. Symbolic Methods in Computational Cryptography Proofs. In *32nd IEEE Computer Security Foundations Symposium (CSF'19)*. IEEE Computer Society, 136–151.

[5] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices* 44, 1 (2009), 90–101.

[6] Gilles Barthe, Charlie Jacomme, and Steve Kremer. 2020. Universal equivalence and majority on probabilistic programs over finite fields. (2020). https://hal.inria.fr/hal-02552287

[7] Benjamin Bichsel, Timon Gehr, and Martin Vechev. 2018. Fine-grained Semantics for Probabilistic Programs. In *27th European Symposium on Programming (ESOP'18) (Lecture Notes in Computer Science)*, Vol. 10801. Springer, 145–185.

[8] Enrico Bombieri. 1966. On exponential sums in finite fields. *American Journal of Mathematics* 88, 1 (1966), 71–105.

[9] Brent Carmer and Mike Rosulek. 2016. Linicrypt: a model for practical cryptography. In *36th Annual International Cryptology Conference (CRYPTO'16) (Lecture Notes in Computer Science)*, Vol. 9816. Springer, 416–445.

[10] Dmitry Chistikov, Andrzej S Murawski, and David Purser. 2019. Asymmetric Distances for Approximate Differential Privacy. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[11] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. 1985. The Bit Extraction Problem of t-Resilient Functions (Preliminary Version). In *26th Annual Symposium on Foundations of Computer Science (FOCS'85)*. IEEE Computer Society, 396–407.

[12] Bernard Dwork. 1960. On the rationality of the zeta function of an algebraic variety. *American Journal of Mathematics* 82, 3 (1960), 631–648.

[13] Graham Everest, Alf van der Poorten, Igor Shparlinski, and Thomas Ward. 2002. Exponential functions, linear recurrence sequences, and their applications.

[14] Matthew Fredrikson and Somesh Jha. 2014. Satisfiability modulo counting: A new approach for analyzing privacy properties. In *Joint Meeting of the 23rd Annual Conference on Computer Science Logic (CSL) and the 29th ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM, 1–10.

[15] Marco Gaboardi, Kobbi Nissim, and David Purser. 2019. The Complexity of Verifying Circuits as Differentially Private. *CoRR* abs/1911.03272 (2019). arXiv:1911.03272 http://arxiv.org/abs/1911.03272

[16] William Andrew Johnson. 2016. *Fun with fields*. Ph.D. Dissertation. UC Berkeley.

[17] Charanjit S Jutla and Arnab Roy. 2012. Decision procedures for simulatability. In *17th European Symposium on Research in Computer Security (ESORICS'12) (Lecture Notes in Computer Science)*, Vol. 7459. Springer, 573–590.

[18] Catarina Kiefe. 1976. Sets definable over finite fields: their zeta-functions. *Trans. Amer. Math. Soc.* 223 (1976), 45–59.

[19] Alan GB Lauder and Daqing Wan. 2006. Counting points on varieties over finite fields of small characteristic. *arXiv preprint math/0612147* (2006).

[20] Axel Legay, Andrzej S Murawski, Joël Ouaknine, and James Worrell. 2008. On automated verification of probabilistic programs. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08) (Lecture Notes in Computer Science)*, Vol. 4963. Springer, 173–187.

[21] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, Yehuda Lindell (Ed.). Springer International Publishing, 277–346. https://doi.org/10.1007/978-3-319-57048-8_6

[22] Michael L Littman, Judy Goldsmith, and Martin Mundhenk. 1998. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research* 9 (1998), 1–36.

[23] Stefan Maubach. 2001. The automorphism group over finite fields. (2001).

[24] Scott McCallum and Volker Weispfenning. 2012. Deciding polynomial-transcendental problems. *Journal of Symbolic Computation* 47, 1 (2012), 16–31.

[25] Maurice Mignotte, Tarlok Nath Shorey, and Robert Tijdeman. 1984. The distance between terms of an algebraic recurrence sequence. *Journal für die reine und angewandte Mathematik* 349 (1984), 63–76.

[26] Andrzej S. Murawski and Joël Ouaknine. 2005. On Probabilistic Program Equivalence and Refinement. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings (Lecture Notes in Computer Science)*, Martín Abadi and Luca de Alfaro (Eds.), Vol. 3653. Springer, 156–170. https://doi.org/10.1007/11539452_15

[27] Tobias Nipkow. 1990. Unification in Primal Algebras, Their Powers and Their Varieties. *J. ACM* 37, 4 (Oct. 1990), 742–776.

[28] Joël Ouaknine and James Worrell. 2012. Decision problems for linear recurrence sequences. In *6th International Workshop on Reachability Problems (RP'12) (Lecture Notes in Computer Science)*, Vol. 7550. Springer, 21–28.

[29] Joël Ouaknine and James Worrell. 2014. Positivity problems for low-order linear recurrence sequences. In *25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. Society for Industrial and Applied Mathematics, 366–379.

[30] Jacobo Toràn. 1988. An oracle characterization of the counting hierarchy. In *3rd Annual Structure in Complexity Theory Conference*. 213–223.

[31] Jacobo Torán. 1991. Complexity classes defined by counting quantifiers. *J. ACM* 38 (1991), 753–774.

[32] NK Vereshchagin. 1985. The problem of appearance of a zero in a linear recurrence sequence. *Mat. Zametki* 38, 2 (1985), 609–615.

[33] André Weil. 1949. Numbers of solutions of equations in finite fields. *Bulletin of the AMS* (1949).