Post-quantum sound CRYPTOVERIF and verification of hybrid TLS and SSH key-exchanges

Bruno Blanchet
Inria, F-75012 Paris, France
bruno.blanchet@inria.fr

Charlie Jacomme
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
charlie.jacomme@inria.fr

***Abstract*—With the potential arrival of quantum computers, communication protocols are now being updated at a fast pace to be secure even against attackers with access to such a computer. A core issue is that we need to update the existing tools used to verify security protocols, as classical security proofs do not always carry over to quantum attackers.**

In this work, we prove the post-quantum soundness of the CRYPTOVERIF prover, a tool used to semi-automatically obtain computational security guarantees over cryptographic constructions. It required an update of the whole semantics in order to define it for any black-box interactive attacker and not just probabilistic Turing machines. We also had to validate the soundness of all its proof techniques, the so-called game transformations.

We used this new post-quantum sound CRYPTOVERIF to obtain the first formal security guarantees over two IETF draft proposals designing post-quantum variants of SSH and TLS.

***Index Terms*—protocol verification; formal methods; cryptography; post-quantum; TLS**

I. INTRODUCTION

It is well known that a quantum computer can compute discrete logarithms and break Diffie-Hellman (DH)-based cryptography. So, while no one can ascertain when a quantum computer may actually be able to do so, it has become sufficiently plausible for the research community to start globally upgrading the security of existing protocols.

As a witness, multiple NIST standardization processes are ongoing for post-quantum cryptographic primitives, and notably for Key Encapsulation Mechanisms (KEMs), meant to replace the DH-based computations. On the protocol side, we find academic proposals of fully novel post-quantum key exchanges [1], [2], [3] only based on post-quantum KEMs.

For a faster deployment, other people are trying to provide small incremental updates of existing protocols, so that by adding a KEM computation in parallel to the DH ones, we get the best of both worlds. This yields so-called hybrid key-exchanges, that have the benefit of relying in the classical setting on the well known security of DH computations, while also independently adding the benefit of the less studied KEMs believed to be post-quantum secure. The Signal messaging application already deployed a new hybrid post-quantum version of its initial key exchange, dubbed PQXDH [4]. In addition, there are ongoing drafts at the Internet Engineering Task Force (IETF) for hybrid variants of Transport Layer Security (TLS) 1.3 [5] and Secure Shell (SSH) [6].

The pace at which new protocols are developed or updated raises the challenge of being able to obtain security guarantees on them. In the classical setting, the CRYPTOVERIF tool has successfully been used to provide guarantees on several protocols, and notably the classical TLS [7] and SSH [8] key exchanges. CRYPTOVERIF mechanizes the game-hopping technique of classical cryptographic proofs:

- the protocol is expressed as a set of oracles that can be queried by the attacker;
- security is expressed by secrecy (the attacker can distinguish the secret from a fresh random value only with negligible probability), correspondence (if an event has been raised, then some other event has been raised with overwhelming probability; useful for modeling authentication), and indistinguishability (the attacker can distinguish the real world from an ideal and obviously secure world only with negligible probability) properties;
- and CRYPTOVERIF performs valid rewritings over the game, the game-hops, until it can provide a sequence of transformations leading to a game in which the desired property can easily be proved without any cryptographic assumption.

An issue of classical cryptographic proofs, whether pen-and-paper or mechanized by CRYPTOVERIF, is that a proof valid against a classical attacker may not be valid against a quantum attacker. This can of course be the case when the proof relies on a security assumption over a primitive that does not have a post-quantum secure instantiation, or which is simply false for a quantum attacker. Such is the case for a proof relying on, e.g., the Decisional Diffie-Hellman (DDH) assumption over the elliptic curve X25519.

This issue is easy to resolve: one just needs to ascertain that all the cryptographic assumptions over the primitives do have post-quantum instantiations. However, classical proofs can in fact be non post-quantum for deeper reasons, for instance due to proof techniques that specifically refer to, and reason on, the internal state of the attacker. An example is the rewinding technique, where the attacker state is saved at some point, and at any point in the proof we can reset the attacker to this state.

Such operations are impossible to carry out on a quantum computer due to the "no-cloning theorem" [9]: it is impossible to duplicate a quantum state. As such, any proof that relies on such techniques is in fact not post-quantum sound. This

core issue for post-quantum sound cryptographic proofs was notably discussed in [10], [11]. More generally, the main difficulties to get post-quantum sound proofs are discussed in [12], [13], where a class of valid proofs that do carry over to quantum attackers is identified.

While CRYPTOVERIF does not appear to use problematic proof techniques and seems to naturally fall inside the category of post-quantum sound techniques, two problems arise:

- formally ensuring this is difficult, as every technique and associated soundness proof needs to be checked;
- and further, it turns out that the theoretical foundations of CRYPTOVERIF do not allow considering quantum attackers in the semantics.

Indeed, CRYPTOVERIF uses a dialect of the pi-calculus to express protocols, and the security is defined for protocols running inside any context built within the same dialect. That is, we define the possible attackers as any program written using the same programming language as the one used for protocols. This programming language naturally only allows to write classical protocols, and not quantum computations. As such, the CRYPTOVERIF semantics is inherently restricted to classical attackers and does not capture quantum ones.

In this work, we thus set out to address the corresponding research question:

Can we update the CRYPTOVERIF *semantics and associated reasoning techniques to make it post-quantum sound?*

a) Contributions: Our main contributions are three-fold:

- we make CRYPTOVERIF post-quantum sound by designing a semantics where attackers can be any black-box interactive attacker, whose computing power is only restricted by the cryptographic assumptions;
- while updating the whole semantics, we in fact made CRYPTOVERIF closer to classical cryptographic proofs: it previously reasoned over protocols expressed inside a dialect of the pi-calculus, while the semantics is now directly defined on an oracle-based syntax;
- we formally prove the security of current IETF draft proposals for a hybrid TLS [5] and SSH [6], both in the classical and post-quantum settings. These protocols are clearly important in practice. In particular, hybrid TLS is already supported by Chrome from version 116 [14].

On the practical side, CRYPTOVERIF comes with a library of classical cryptographic assumptions on primitives. We also developed in this work a post-quantum variant of this library, removing non post-quantum sound assumptions, as well as adding the classical ones for KEMs.

Our developments have been included in the official CRYPTOVERIF release version 2.09 (http://cryptoverif.inria.fr/). Its examples folder also contains our case studies.

A. Related Work

Making existing tools post-quantum sound has been a recent undertaking, for instance for EASYCRYPT [15] in [16], and SQUIRREL [17] in [18]. Yet, as all of them use different theoretical foundations (game-hops for CRYPTOVERIF, a probabilistic Relational Hoare Logic for EASYCRYPT, and the Bana-Comon [19] logic for SQUIRREL), the process is fundamentally different for each tool. Importantly, all those tools are complementary:

- EASYCRYPT is fully interactive and more expressive, it is thus better at verifying complex primitives;
- SQUIRREL is a bit more abstracted and still fully interactive; while less expressive, it enables proving stateful protocols and more complex security properties like privacy-based ones;
- CRYPTOVERIF is on its side the most automated and highly efficient at verifying, e.g., key-exchanges, but when automation fails it does not have a full logic to reason in a fine-grained way about protocol executions, thus for instance not allowing proofs of stateful protocols.

The Random Oracle Model (ROM) raises additional difficulties in the quantum setting, as the attacker must be able to make queries that are in quantum superpositions. Only [16] supports the quantum Random Oracle Model (qROM), and neither us nor [18] do. It is crucial to support it when targeting primitives, but less so for protocols, which are our main target. Indeed, we can often prove the security of protocols under different assumptions for the hash functions, typically by relying on the Pseudorandom Function (PRF) assumption.

Historically, the semantics of CRYPTOVERIF was defined over a pi-calculus syntax. An oracle-based syntax was previously proposed in [20] and a translation from an oracle front-end to the pi-calculus syntax was in fact already implemented in CRYPTOVERIF. However, this oracle syntax came with no actual semantics, and the translation with no guarantees.

TLS 1.3 was previously studied with CRYPTOVERIF in [7], and SSH in [8]. Our own models for their hybrid proposals are based on those previous models, which we extend with the new KEM-based parts. Proving a hybrid key exchange implies making two distinct proofs, one only under the classical assumptions without any assumption about the KEMs, and one only under the post-quantum sound assumptions and no assumption about the DH-based part. The existing proofs for the classical part had to be adapted, and we came up with fully novel proofs for the post-quantum part.

II. ORACLE-BASED LANGUAGE FOR CRYPTOVERIF

We first present here our novel semantics for the oracle-based input language. Recall that the semantics of CRYPTOVERIF was only defined on a pi-calculus-based one.

To allow for complex proof techniques and reasonings, the syntax of CRYPTOVERIF is rich and contains many constructs. For instance, conditionals can be seen as a computation happening either directly inside a message or at the level of processes. We only present here simplified syntax and semantics; we refer the reader to our extensive technical report [21] on CRYPTOVERIF for the full definitions.

A. Syntax

A simplified view of the syntax is given in Fig. 1. Types are sets of values, typically bit-strings, e.g., the set of public keys

of a KEM or booleans. Some types are pre-defined, $bool = \{\text{true}, \text{false}\}$, where false is 0 and true is 1; *bitstring* is the set of all bit-strings. CRYPTOVERIF models abstract computations using typed function symbols $f : T_1 \times \ldots \times T_m \to T$, where f represents an efficiently computable, deterministic function that maps each tuple in $T_1 \times \ldots \times T_m$ to an element of T.

To represent computations over bit-strings, we thus rely on *terms*, built from function applications as well as variable accesses. A variable x models a cell, which is initially undefined and can be set at most once to some value by an oracle. Such variables can be accessed by any oracle and are not restricted by the scope. Variables are indexed by terms that evaluate to integers, denoted by $x[M_1, \ldots, M_m]$, thus modeling arrays of variables. Note that some function symbols are pre-defined, such as $M = N$ for the equality test (using infix notation for practicality), and $M \neq N$ for the inequality test. Other terms will be built with function symbols modeling computations for which we do not specify a concrete implementation (but may specify functional or security properties), such as an encryption function symbol $\text{enc}(M, N)$.

A game is defined by an *oracle definition* Q, which defines the set of oracles available to the attacker. Nil 0 defines no oracle. Some oracles can be made available in parallel, using $Q \mid Q'$. The same oracle can be replicated n times using foreach $i \leq n$ do Q; the copies are indexed by the *replication index* $i \in \{1, \ldots, n\}$. And finally, an oracle definition may actually declare an oracle $O[\widetilde{i}](x[\widetilde{i}] : T) := P$, where $x[\widetilde{i}]$ is the variable storing the argument passed by the attacker when calling the oracle, T is its type, and P is the actual code of the oracle, its *oracle body*. The notation $\widetilde{i}$ represents a sequence of replication indices $i_1, \ldots, i_m$ that must correspond to the *current replication indices* at the oracle definition: replications foreach $i \leq n$ do Q can be nested and the current replication indices at a program point are the indices i_k of the replications foreach $i_k \leq n_k$ do Q enclosing that program point. All definitions of variables must use the current replication indices, which helps to make sure that variables are set at most once. Oracles are also indexed by the current replication indices so that the attacker can call a specific copy of an oracle by using the appropriate indices.

An oracle body is made of commands, and finalized by a potential return value. The command $x[\widetilde{i}] \stackrel{R}{\leftarrow} T; P$ samples uniformly at random from the set of values given by the type T, stores the result inside the array cell $x[\widetilde{i}]$, and runs P. Oracles can use conditionals with if M then P else P'. Remark that the term M must then be of type *bool*, and recall that we can compute equality inside terms.

The find construct is the one that allows to access variables out of their scope; it allows "magical" communications between protocol participants, which is convenient to express security properties and cryptographic games obtained after exploiting those properties. Formally, it performs an array lookup: find $u[\widetilde{i}] = i \leq n$ suchthat defined$(M_1, \ldots, M_l) \wedge M$ then P else P' looks for an index $i \in \{1, \ldots, n\}$ such that the array cells in terms M_1, ..., M_l are defined and the term M is true. When such an index is found, it is stored in the array cell $u[\widetilde{i}]$, and P is executed. (When several successful values of i are found, one of them is chosen randomly with uniform probability.) Otherwise, P' is executed. The full calculus of CRYPTOVERIF has a more general find construct, with multiple indices and then branches. For each variable access $x[M_1, \ldots, M_m]$, either x must be syntactically in scope and $M_1, \ldots, M_m$ are the current replication indices, or the definition of $x[M_1, \ldots, M_m]$ is guaranteed by a defined condition of a find construct. To lighten notations, indices can be omitted when they are the current replication indices.

$M, N ::=$	terms
$x[M_1, \ldots, M_m]$	variable access
$f(M_1, \ldots, M_m)$	function application
$Q ::=$	oracle definitions
0	nil
$Q \mid Q'$	parallel composition
foreach $i \leq n$ do Q	replication n times
$O[\widetilde{i}](x[\widetilde{i}] : T) := P$	oracle declaration
$P ::=$	oracle body
yield	end
return$(N); Q$	return
let $x[\widetilde{i}] : T = O[M_1, \ldots, M_l](N)$ in P else P'	oracle call
$x[\widetilde{i}] \stackrel{R}{\leftarrow} T; P$	random number
if M then P else P'	conditional
find $u[\widetilde{i}] = i \leq n$ suchthat defined$(M_1, \ldots, M_l) \wedge M$ then P else P'	array lookup
insert $Tbl(M_1, \ldots, M_l); P$	insert in table
get $Tbl(x_1[\widetilde{i}] : T_1, \ldots, x_l[\widetilde{i}] : T_l)$ suchthat M in P else P'	get from table
event $e(M_1, \ldots, M_l); P$	event

Fig. 1. Syntax for games (simplified)

Tables allow for instance to store the long term keys of agents. A table Tbl can be seen as a list of tuples; insert $Tbl(M_1, \ldots, M_l); P$ inserts the element $M_1, \ldots, M_l$ in the table Tbl and then runs P, and get $Tbl(x_1 : T_1, \ldots, x_l : T_l)$ suchthat M in P else P' tries to find an element $(x_1, \ldots, x_l)$ in the table Tbl such that M is true. If there exists such an element, then it executes P with $x_1, \ldots, x_l$ bound to that element. Otherwise, it executes P'.

The command return(N) returns the value of a term N to the caller. The current oracle then terminates, and control is given back to the caller. When terminating, an oracle can specify a new set of oracles that become available to the attacker; return$(N); Q$ thus also executes the given Q definition. An oracle can also terminate without any continuation nor returning any value by using the yield instruction, which corresponds to terminating with an error.

The oracle call let $x[\widetilde{i}] : T = O[M_1, \ldots, M_l](N)$ in P else P' calls oracle O with indices M_1, ..., M_l and with argument N. When oracle O terminates with return, the returned result

is stored in $x[\widetilde{i}]$ and P is executed. When oracle O terminates with yield, P' is executed. The oracle call is currently not allowed in games manipulated by CRYPTOVERIF. However, its presence in the language is useful for the soundness proof of CRYPTOVERIF, in particular to deal with proofs by reduction. Moreover, allowing oracle calls paves the way for future extensions of CRYPTOVERIF, for sharing code by calling the same oracle, for composition results, and for dealing with loops encoded as recursive oracle calls.

Finally, oracles can raise so-called events, by event $e(M_1, \ldots, M_l); P$. Those do not model any actual computation made by the underlying protocol, but are used to model and reason over the possible executions of a protocol. We typically raise an event whenever some key exchange participant finalizes a session and derives a key, and use it to define authentication by requiring that if a participant raises an event, then its interlocutor raises a matching event.

Example 1 Consider a symmetric encryption function, defined by a triple of function symbols $\mathrm{kgen} : \mathit{keyseed} \mapsto \mathit{skeys}$, $\mathrm{enc} : \mathit{bitstring} \times \mathit{skeys} \times \mathit{seed} \mapsto \mathit{bitstring}$ and $\mathrm{dec} : \mathit{bitstring} \times \mathit{skeys} \mapsto \mathit{bitstring}_\perp$. Remark that in CRYPTOVERIF, all function symbols represent deterministic functions. The key generation function kgen thus goes from a seed type $\mathit{keyseed}$ for the keys to the actual key space skeys. Similarly, enc expects in addition to the plaintext and the key some randomness source from the seed space. The decryption function dec may fail; its return value is thus of type $\mathit{bitstring}_\perp$, the set of bitstrings extended with a dedicated failure symbol $\perp$.

We can define a small game in our syntax as:

$$\begin{array}{l}\mathsf{foreach}\ i' \leq n'\ \mathsf{do}\ O_{init}[i'](y[i'] : \mathit{bitstring}) := \\ \quad r[i'] \xleftarrow{R} \mathit{keyseed}; \mathsf{return}(\mathrm{true}); \\ \quad \mathsf{foreach}\ i \leq n\ \mathsf{do}\ O_e[i', i](x[i', i] : \mathit{bitstring}) := \\ \quad\quad r'[i', i] \xleftarrow{R} \mathit{seed}; \mathsf{return}(\mathrm{enc}(x[i', i], \mathrm{kgen}(r[i']), r'[i', i]))\end{array}$$

Each copy of the oracle O_{init} defines a new key by sampling r inside the key seed space, then simply gives back the control to the attacker by returning true. (Its argument $y[i']$ is not used and could be omitted.) This oracle can be used to instantiate up to n' distinct keys, and each time it gives access to a new oracle O_e, that can itself be called up to n times. The attacker is expected to give some input message $x[i', i]$, and obtains in return the encryption of this message under the key $\mathrm{kgen}(r[i'])$, each time with an independent fresh random $r'[i', i]$.

Omitting current replication indices and unused arguments of oracles, this game can also be written:

$$\begin{array}{l}\mathsf{foreach}\ i' \leq n'\ \mathsf{do}\ O_{init} := r \xleftarrow{R} \mathit{keyseed}; \mathsf{return}(\mathrm{true}); \\ \quad \mathsf{foreach}\ i \leq n\ \mathsf{do}\ O_e(x : \mathit{bitstring}) := \\ \quad\quad r' \xleftarrow{R} \mathit{seed}; \mathsf{return}(\mathrm{enc}(x, \mathrm{kgen}(r), r'))\end{array}$$

B. Semantics

The previous CRYPTOVERIF semantics did not directly include an attacker, and only reasoned over a protocol P expressed in a pi-calculus. Security was then defined by considering all the possible executions of $C[P]$ for all possible contexts C representing an attacker and also expressed inside the pi-calculus. As the pi-calculus was expressive enough to model any probabilistic Turing Machine, it did capture classical computational attackers. Yet, expressing the security by using such contexts does not allow considering quantum attackers, as they cannot be captured by the syntax for protocols. To resolve this issue, we define a new semantics for CRYPTOVERIF that directly includes an attacker interacting with the set of oracles. The semantics will in fact be defined for any black-box interactive attackers, which is a safe way to model a quantum attacker while ensuring we never reason unsoundly over its inner quantum state.

We thus completely redefine the semantics, as presented here. We only give a high-level idea of the configurations and semantic rules, and once again refer the reader to the technical report [21] for the exhaustive version.

From a high-level point of view, we inherit many elements from the previous semantics. The current state of the system is modeled by a configuration. The possible evolution of the system is represented by a set of reduction rules between configurations. The sequence of configurations obtained by these reduction rules is called a trace, which models one of the possible executions of the system. The semantics is probabilistic: each rule has an execution probability attached to it, and each trace of the system then occurs with a probability computed from the probabilities of its steps. Each configuration contains a map E defining the current values of the variables, a set $\mathcal{T}$ storing the contents of tables, and a sequence $\mu\mathcal{E}v$ listing the previously raised events.

The new additions to the semantics aim at modeling the attacker as a black-box interactive machine interacting with a set of oracles. To enable oracle calls both from the attacker as well as other oracles, we extend configurations with a stack $\mathcal{S}t$ to represent the nested oracle calls. The attacker is at the bottom of the stack and can perform a new oracle call whenever it receives an answer to its previous query. The top of the stack contains the oracle being executed. When a command of this oracle is executed, it is simply removed and the rest of the oracle remains on the top of the stack, until a return command is executed. The oracle is then popped from the stack and the returned value is passed either to the next oracle on the stack or to the attacker. The configuration also contains the set of callable oracles $\mathcal{Q}$, where called oracles are removed from the set and new oracles may be added during the execution.

We now turn to formally defining those elements.

a) Configurations: The semantics depends on an attacker that may interact with the oracles. We consider attackers $\mathcal{A}$ from a recursively defined attacker space $\mathcal{BB}$, that are probabilistic functions of type $\mathcal{A} : \mathit{Bitstring} \mapsto \mathit{Distr}(\mathit{Bitstring} \times \mathcal{BB})$. To each input, the attacker $\mathcal{A}$ gives out a probability distribution yielding a bit-string and a new attacker. It models an interactive black-box attacker that given an input, gives back with some probability both an output and its next attacker

step. Here, $Bitstring = \{0,1\}^*$ is the set of concrete bit-strings, and we implicitly rely on the efficient bijections from types and oracle names to concrete bit-strings so that the attacker can interact with the oracles.

In the semantics, each term or command is labeled with a program point μ. A semantic configuration is a tuple $E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v$, where

- E is an environment mapping array cells to values.
- $\mathcal{S}t$ is an execution stack of the form $(\sigma_0, P_0) :: \ldots :: (\sigma_l, P_l) :: \mathcal{A}$ where P_j are oracle bodies and σ_j are the associated mapping sequences which give the values of replication indices. The oracle body P_0 is the one currently being executed, and the oracle bodies P_1, …, P_m correspond to oracle calls that have not returned yet. The stack always contains at least one pair (σ_j, P_j) and always contains as a last element an attacker $\mathcal{A}$.
  The sequence $\sigma = [i_1 \mapsto a_1, \ldots, i_m \mapsto a_m]$ is a sequence of mappings from replication indices to integers $i_j \mapsto a_j$. Its image $\mathrm{Im}(\sigma)$ is the sequence of integers $a_1, \ldots, a_m$.
- $\mathcal{Q}$ is the multi-set of oracle declarations running in parallel with P_0 (the top of the stack $\mathcal{S}t$), with their associated mapping sequences giving values of replication indices. It is the set of callable oracles.
- $\mathcal{T}$ defines the contents of tables. It is a list of $Tbl(a_1, \ldots, a_m)$ indicating that table Tbl contains the element $(a_1, \ldots, a_m)$.
- $\mu\mathcal{E}v$ is a sequence representing the events executed so far. Each element of the sequence is of the form $(\mu, \widetilde{a}) : e(a_1, \ldots, a_m)$, meaning that the event $e(a_1, \ldots, a_m)$ has been executed at a program point μ with replication indices evaluating to $\widetilde{a}$.

Such a configuration effectively captures the possible states of a given protocol and attacker at some point in time, along with some bookkeeping tailored for security proofs.

b) Execution Rules: The semantics is defined by reduction rules of the form $E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \mathcal{S}t', \mathcal{Q}', \mathcal{T}', \mu\mathcal{E}v'$ meaning that $E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v$ reduces to $E', \mathcal{S}t', \mathcal{Q}', \mathcal{T}', \mu\mathcal{E}v'$ with probability p. In $\xrightarrow{p}_t$, the index t just serves in distinguishing reductions that yield the same configuration with the same probability in different ways, so that the probability of a certain reduction can be computed correctly:

$$\Pr[E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \to E', \mathcal{S}t', \mathcal{Q}', \mathcal{T}', \mu\mathcal{E}v'] = \sum_{E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \mathcal{S}t', \mathcal{Q}', \mathcal{T}', \mu\mathcal{E}v'} p$$

This allows us to define the probability of a trace $Tr = E_1, \mathcal{S}t_1, \mathcal{Q}_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p_1}_{t_1} \ldots \xrightarrow{p_{m-1}}_{t_{m-1}} E_m, \mathcal{S}t_m, \mathcal{Q}_m, \mathcal{T}_m, \mu\mathcal{E}v_m$ as $\Pr[Tr] = p_1 \times \ldots \times p_{m-1}$. Bounding the probability of bad traces violating the security property will then be the goal of CryptoVerif reasonings.

To define the relation $\xrightarrow{p}_t$ that will capture the execution of the commands, we first define how to reduce an oracle definition to the corresponding set of newly defined oracles, using the auxiliary reduction relation $\rightsquigarrow$, which transforms configurations of the form $\mathcal{Q}$. We provide the natural reduction rules in Fig. 2, where (Nil) removes nil definitions, the

$$\{(\sigma, {}^{\mu}0)\} \uplus \mathcal{Q} \rightsquigarrow \mathcal{Q} \quad \text{(Nil)}$$

$$\{(\sigma, {}^{\mu}(Q_1 \mid Q_2))\} \uplus \mathcal{Q} \rightsquigarrow \{(\sigma, Q_1), (\sigma, Q_2)\} \uplus \mathcal{Q} \quad \text{(Par)}$$

$$\{(\sigma, {}^{\mu}\mathsf{foreach}\ i \leq n\ \mathsf{do}\ Q)\} \uplus \mathcal{Q} \rightsquigarrow \{(\sigma[i \mapsto a], Q) \mid a \in [1, n]\} \uplus \mathcal{Q} \quad \text{(Repl)}$$

$$\{(\sigma, {}^{\mu}O[\widetilde{i}](x[\widetilde{i}] : T) := P)\} \uplus \mathcal{Q} \rightsquigarrow \{(\sigma, {}^{\mu}O[\sigma(\widetilde{i})](x[\widetilde{i}] : T) := P)\} \uplus \mathcal{Q} \quad \text{(DefOracle)}$$

$\mathrm{reduce}(\mathcal{Q})$ is the normal form of $\mathcal{Q}$ by $\rightsquigarrow$

Fig. 2. Semantics for oracle definitions

rules (Par) and (Repl) respectively expand parallel compositions and replications, and the rule (DefOracle) evaluates the replication indices in an oracle declaration.

We now define $\xrightarrow{p}_t$, by assuming that oracle definitions in $\mathcal{Q}$ in configurations $E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v$ are always in normal form by $\rightsquigarrow$, so they always start with an oracle declaration. The full semantics has around 50 rules, we describe 7 in Fig. 3 and we omit, e.g., the reductions that evaluate terms M to concrete values a and assume that all terms are already evaluated.

All the rules execute the current command of the oracle at the top of the execution stack. Rule (Insert) naturally inserts the new value $(a_1, \ldots, a_l)$ inside the given table Tbl. (New) performs a random sampling over the domain of the type: it takes one concrete value $a \in T$, extends the environment E to map the variable $x[\sigma(\widetilde{i})]$ to this value, and then continues the execution. This reduction can then be executed with probability $D_T(a)$, which denotes the probability of sampling a in T ($1/|T|$ for the uniform distribution). (If1) directly resolves a positive conditional branching (recall that terms are already evaluated, as other omitted reduction rules would take care of evaluating the actual condition M to either true or false). A similar rule deals with false conditions. Rules for find and get are more complex, and are fundamentally unchanged from the pi-calculus version, so we prefer omitting them here. Rule (Event) adds the executed event to $\mu\mathcal{E}v$.

Rule (OracleCall) calls oracle $O[a_1, \ldots, a_l]$: it collects in multiset S the available oracles with matching name O and indices $a_1, \ldots, a_l$. When no matching oracle is found, execution blocks. Otherwise, we choose one at random (σ', Q_0) and call it: we store the argument b of the call in the parameter $x[\widetilde{i'}]$ of Q_0, by extending the environment E; we remove the called oracle from the set of available oracles $\mathcal{Q}$; and we push the body of the called oracle on the stack to execute it. The called oracle is removed from the available oracles so that it cannot be called again. If the considered oracle is an instance of a replicated oracle, only one copy is removed and the others remain. The probability expresses the uniform choice of (σ', Q_0) in S: $S(\sigma', Q_0)$ is the number of copies of (σ', Q_0) in the multiset S and $|S|$ is the cardinal of S. As we show below (Lemma 1, Item 2), S contains at most one element, so in fact $S(\sigma', Q_0)/|S| = 1$. Rule (Return) performs a return: it pops the return from the stack, replaces the caller P_0 with its in branch P_1 on the top of the stack and stores the

return value d in the variable $x[\widetilde{i}]$ of the caller by extending the environment E. Hence, it executes the in branch P_1 of the caller with $x[\widetilde{i}]$ equal to the returned value. A similar rule deals with yield.

Finally, we present the main rule corresponding to an interaction with our black box attacker. Rule (AttIO) corresponds to the attacker receiving a return value and then triggering a new oracle call in response. The stack contains a last oracle doing a return with value d to the attacker who is next in the stack. $\mathcal{A}(d)$ then defines a distribution $D^d_{\mathcal{A}}$, from which we sample the next message sent by the attacker m_0 as well as the next attacker step $\mathcal{A}_0$. m_0 defines the next oracle call, with an oracle name O with its replication indices $a_1, \ldots, a_k$, as well as the value for the input parameter b. We build the multiset S of potential oracles matching this call, and take one at random that we actually call by putting it on top of the stack, dropping it from $\mathcal{Q}$, and setting its input variable to b inside the environment E. The oracles available after this step are the oracles available before $\mathcal{Q}$, plus the oracles defined after the executed return $\mathcal{Q}' = \mathrm{reduce}(\{(\sigma, Q)\})$, minus the oracle that the attacker calls (σ', Q_0).

c) Traces: The semantics enables us to define the set of traces of a protocol for a given attacker, where we simply consider all possible reductions from an initial game directly giving control to the attacker.

Definition 1 The initial configuration for running oracle definition Q against attacker $\mathcal{A}$ is $\mathrm{initConfig}(Q, \mathcal{A}) = \emptyset, (\sigma_0, \mathsf{return}(\mathit{start})) :: \mathcal{A}, \mathcal{Q}, \emptyset, \emptyset$ where $\mathcal{Q} = \mathrm{reduce}(\{(\sigma_0, Q)\})$, σ_0 is the empty mapping sequence, and start is a constant.

The set of possible executions starting from this initial configuration then corresponds to the set of traces of the protocol, defined for a given attacker.

Definition 2 A *trace of* Q for attacker $\mathcal{A}$ is a trace that starts from $\mathrm{initConfig}(Q, \mathcal{A})$: $\mathit{Tr} = \mathrm{initConfig}(Q, \mathcal{A}) \xrightarrow{p_1}_{t_1} \ldots \xrightarrow{p_{m-1}}_{t_{m-1}} \mathit{Conf}_m$. Let $\mathcal{T}r^{\mathcal{A},Q}$ be the set of all traces of Q for an attacker $\mathcal{A} \in \mathcal{BB}$, and $\mathcal{T}r^{\mathcal{A},Q}_{\mathrm{full}}$ be the subset of traces of $\mathcal{T}r^{\mathcal{A},Q}$ whose last configuration Conf_m cannot be reduced.

C. Syntax Restrictions and Invariants

CryptoVerif enforces a set of well-formedness restrictions on the possible initial games, that enables us to derive invariants on the concrete executions. We present here a core invariant that summarizes the exhaustive list of [21].

Invariant 1 (Single and correct definitions, typing) The oracle definition Q_0 satisfies Invariant 1 if and only if

1) in every definition of $x[i_1, \ldots, i_m]$ in Q_0, the indices $i_1, \ldots, i_m$ of x are the current replication indices, and two different definitions of the same variable x in Q_0 are in different branches of a if, find, or get.
2) in every definition of oracle $O[i_1, \ldots, i_m]$ in Q_0, the indices $i_1, \ldots, i_m$ of O are the current replication indices, and two different definitions of the same oracle O in Q_0 are in different branches of a if, find, or get.
3) every occurrence of a variable access $x[M_1, \ldots, M_m]$ in Q_0 is either
   - syntactically under the definition of $x[M_1, \ldots, M_m]$ (in which case $M_1, \ldots, M_m$ are in fact the current replication indices at the definition of x), or
   - in a defined condition of find, or
   - $x[M_1, \ldots, M_m]$ occurs in the condition M of find $u[\widetilde{i}] = i \leq n$ suchthat defined$(M'_1, \ldots, M'_l) \wedge M$ then P else P' and $x[M_1, \ldots, M_m]$ is a subterm of M'_j for some $j \leq l$, or
   - $x[M_1, \ldots, M_m]$ occurs in the then branch P of find $u[\widetilde{i}] = i \leq n$ suchthat defined$(M'_1, \ldots, M'_l) \wedge M$ then P else P' and $x[M_1, \ldots, M_m]$ is a subterm of $M'_j\{u[\widetilde{i}]/i\}$ for some $j \leq l$ ($M'_j\{u[\widetilde{i}]/i\}$ means M_j with $u[\widetilde{i}]$ substituted for i).
4) Q_0 is well-typed.

Item 1 guarantees that no variable will be bound twice in an execution (the second part of the item guarantees that no definition can be executed twice with the same current replication indices), Item 2 does the same for oracles, and Item 3 guarantees that all variable accesses are over defined variables. In the last two cases of Item 3, the definition of $x[M_1, \ldots, M_l]$ is guaranteed by the defined condition of the find. Item 4 guarantees that Q_0 is well-typed: in short, the type system maps each oracle to a type that specifies the type of its indices, its argument, and its result and each variable to a type that specifies the type of its indices and of its content. It guarantees that all definitions of the same oracle or variable have the same type, that oracles are called with indices, argument, and result of the appropriate type, that returned terms, arguments of functions, indices of variables, arguments of events, elements of tables have the appropriate type, and that conditions of if, find, and get are of type bool.

This invariant is preserved by all game transformations performed by CryptoVerif. From this invariant, we can obtain a set of properties that hold during execution as summarized in the following lemma.

Lemma 1 *If an oracle definition Q satisfies Invariant 1, then for any attacker $\mathcal{A} \in \mathcal{BB}$, for any trace $\mathit{Tr} \in \mathcal{T}r^{\mathcal{A},Q}$, for any $\mathit{Conf} \in \mathit{Tr}$, we let E be the environment of Conf and we have that:*

1) *E contains at most one definition for each variable;*
2) *for every application of the rules* (OracleCall) *and* (AttIO) *in Tr, there is at most one element in S;*
3) *if Conf accesses $x[\widetilde{a}]$, then $x[\widetilde{a}] \in \mathrm{Dom}(E)$.*
4) *Conf is well-typed.*

This lemma is an aggregation of the lemmas proved in the long version [21, Sections 2.4.3 to 2.4.6]. Intuitively, all the points are consequences of Invariant 1 along with a proof by induction over the derivation of the trace; each item is a consequence of the item of Invariant 1 with the same number.

$$E, (\sigma, {}^{\mu}\mathsf{insert}\ Tbl(a_1, \ldots, a_l); P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P) :: \mathcal{S}t, \mathcal{Q}, (\mathcal{T}, Tbl(a_1, \ldots, a_l)), \mu\mathcal{E}v \quad \text{(Insert)}$$

$$\frac{a \in T \qquad E' = E[x[\sigma(\widetilde{i})] \mapsto a]}{E, (\sigma, {}^{\mu}x[\widetilde{i}] \overset{R}{\leftarrow} T; P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{D_T(a)}_{N(a)} E', (\sigma, P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v} \quad \text{(New)}$$

$$E, (\sigma, {}^{\mu}\mathsf{if}\ \mathrm{true}\ \mathsf{then}\ P\ \mathsf{else}\ P') :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \quad \text{(If1)}$$

$$E, (\sigma, {}^{\mu}\mathsf{event}\ e(a_1, \ldots, a_l); P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e(a_1, \ldots, a_l)) \quad \text{(Event)}$$

$$\frac{\begin{array}{c} P_0 = \mathsf{let}\ x[\widetilde{i}] : T = O[a_1, \ldots, a_l](b)\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2 \\ S = \{(\sigma', Q) \in \mathcal{Q} \mid Q = {}^{\mu''}O[a_1, \ldots, a_l](x'[\widetilde{i}'] : T') := P' \text{ and } b \in T' \text{ for some } \mu'', \sigma', x', \widetilde{i}', T', P'\} \\ (\sigma', Q_0) \in S \qquad Q_0 = {}^{\mu'}O[a_1, \ldots, a_l](x'[\widetilde{i}'] : T') := P' \end{array}}{\begin{array}{c} E, (\sigma, {}^{\mu}P_0) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow[O(\sigma', Q_0)]{S(\sigma', Q_0)/|S|} \\ E[x'[\sigma'(\widetilde{i}')] \mapsto b], (\sigma', P') :: (\sigma, {}^{\mu}P_0) :: \mathcal{S}t, \mathcal{Q} \uplus \setminus\{(\sigma', Q_0)\}, \mathcal{T}, \mu\mathcal{E}v \end{array}} \quad \text{(OracleCall)}$$

$$\frac{d \in T \qquad \mathcal{Q}' = \mathrm{reduce}(\{(\sigma, Q)\}) \qquad P_0 = \mathsf{let}\ x[\widetilde{i}] : T = O[a_1, \ldots, a_k](b)\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2}{E, (\sigma, {}^{\mu'}\mathsf{return}(d); Q) :: (\sigma', {}^{\mu}P_0) :: \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E[x[\sigma'(\widetilde{i})] \mapsto d], (\sigma', P_1) :: \mathcal{S}t, \mathcal{Q} \uplus \mathcal{Q}', \mathcal{T}, \mu\mathcal{E}v} \quad \text{(Return)}$$

$$\frac{\begin{array}{c} \mathcal{Q}' = \mathrm{reduce}(\{(\sigma, Q)\}) \\ S = \{(\sigma', Q) \in \mathcal{Q} \uplus \mathcal{Q}' \mid Q = {}^{\mu''}O[a_1, \ldots, a_k](x'[\widetilde{i}] : T') := P' \text{ and } b \in T' \text{ for some } \mu'', \sigma', x', \widetilde{i}, T', P'\} \\ (\sigma', Q_0) \in S \qquad Q_0 = {}^{\mu'}O[a_1, \ldots, a_k](x[\widetilde{i}] : T) := P' \\ \mathcal{A}(d) = D^d_{\mathcal{A}} \qquad m_0, \mathcal{A}_0 \overset{R}{\leftarrow} D^d_{\mathcal{A}} \qquad m_0 = (O, a_1, \ldots, a_k, b) \end{array}}{\begin{array}{c} E, (\sigma, {}^{\mu}\mathsf{return}(d); Q) :: \mathcal{A}, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow[O(\sigma', Q_0, \mathcal{A}_0, a_1, \ldots, a_k, b)]{S(\sigma', Q_0)/|S| \times D^d_{\mathcal{A}}(m_0, \mathcal{A}_0)} \\ E[x[\sigma'(\widetilde{i})] \mapsto b], (\sigma', P') :: \mathcal{A}, \mathcal{Q} \uplus \mathcal{Q}' \uplus \setminus\{(\sigma', Q_0)\}, \mathcal{T}, \mu\mathcal{E}v \end{array}} \quad \text{(AttIO)}$$

Fig. 3. Semantics for oracle bodies (partial)

The type system is extended to configurations by typing each oracle definition and body in it, as well as the environment, the contents of tables, and the executed events.

a) Implementation: The previous implementation of CryptoVerif provided an optional oracle-based input syntax, that was translated into the main internal dialect of the pi-calculus. We updated the implementation to reverse the situation: internally, CryptoVerif only relies on the oracle semantics, and the input syntax as a pi-calculus is translated to oracles. All invariant checks have also been implemented, including the new invariant (Invariant 1, Item 2) and the typing of oracles. This makes the core of CryptoVerif closer to classical cryptographic proofs, and thus easier to understand and verify.

III. Post-Quantum Soundness

Having updated the semantics of CryptoVerif to both be closer to classical cryptographic proofs as well as inherently consider a black-box attacker, we now turn to the core goal, defining a post-quantum sound notion of security, and updating the old game transformations.

A. Black-box Interactive Attackers

Our attacker will be built in two parts, the black-box attacker $\mathcal{A} \in \mathcal{BB}$, as well as an evaluation context C, useful, e.g., to push code inside the attacker in reduction proofs, but also for the attacker $\mathcal{A}$ to return its final result by raising a dedicated final event. Formally, an evaluation context C is a context built from $[\,]$, $Q \mid C$, $C \mid Q$.

From now on, we consider in this section a fixed attacker space $\mathcal{BB}$, and all security definitions are given for this $\mathcal{BB}$. There must exist a function allowing to extract a runtime from the execution of an attacker. Classically, $\mathcal{BB}$ is the set of interactive probabilistic Turing machines, but the theory of CryptoVerif is now sound for any kind of black-box interactive attacker, where the computational power is restricted by the cryptographic axioms. As cryptographic reductions sound for a black-box interactive attacker are also sound for quantum Turing Machines, this notably implies the post-quantum soundness of CryptoVerif, by considering as $\mathcal{BB}$ the set of interactive quantum Turing machines.

We then define the (simplified) notion of indistinguishability used by CryptoVerif, where for any process Q, $\Pr^{\mathcal{A}}[Q : S]$ is the probability that an execution of Q raises the dedicated attacker success event S, which corresponds to $\sum_{Tr \in \mathcal{T}r^{\mathcal{A},Q}_{\mathrm{full}}, \exists (E, \mathcal{S}t, \mathcal{Q}, \mathcal{T}, \mu\mathcal{E}v) \in Tr, \exists \mu, \exists \widetilde{a}, (\mu, \widetilde{a}) : S \in \mu\mathcal{E}v} \Pr[Tr]$.

Definition 3 (Indistinguishability) Let Q and Q' be two games that satisfy Invariant 1.

An evaluation context C is said to be *acceptable* for Q if and only if C and Q do not use any common variable nor any common table, and $C[Q]$ satisfies Invariant 1.

We write $Q \approx_p Q'$ when, for all attackers $\mathcal{A} \in \mathcal{BB}$, for all

evaluation contexts C acceptable for Q and Q',

$$|\Pr^{\mathcal{A}}[C[Q] : S] - \Pr^{\mathcal{A}}[C[Q'] : S]| \leq p(\mathcal{A}, C)$$

This definition formalizes that the probability that algorithms $\mathcal{A}$ and C distinguish the games Q and Q' is at most $p(\mathcal{A}, C)$. Notice that the attacker context is not allowed to access the variables of Q nor use the same tables, as it would otherwise be able to obtain any secret value. The probability p typically depends on the run-time of $\mathcal{A}$ and C but may also depend on other parameters, such as the number of queries to each oracle made by C or $\mathcal{A}$. That is why p takes as arguments the whole algorithms $\mathcal{A}$ and C. In practice, CRYPTOVERIF instantiates p by a function of the run-time of the attacker, the numbers of queries to the oracles, the probability of breaking the security assumptions made on primitives, and the probability of eliminated collisions with random values.

B. Sound Game Transformations

In essence, CRYPTOVERIF implements a set of transformations over games, where the original and transformed games are indistinguishable. For instance, a basic valid transformation is to remove dead else branches in conditionals, as we can trivially prove that $(O := \mathsf{if}\ \mathrm{false}\ \mathsf{then}\ P\ \mathsf{else}\ P') \approx_0 (O := P)$. Another captures the low probability of collisions between random samplings: $(O := x \xleftarrow{R} T; y \xleftarrow{R} T; \mathsf{return}(x = y)) \approx_{\frac{1}{|T|}} (O := \mathsf{return}(\mathrm{false}))$. More generally, the reasoning techniques of CRYPTOVERIF can be split into five categories:

- a set of syntactical transformations fully preserving the set of traces, like swapping the order of two commands, renaming a variable, replacing a variable with its value, ... For all these transformations, the final game is perfectly indistinguishable from the initial one, independently of the power of the attacker.
- collecting a set of facts that hold at some program points (which variables are defined, which conditionals and thus equations are true, ...); infer new facts by combining equalities and eliminating collisions; and use those facts to simplify the game or to prove security properties. Here, the probability of distinguishing the final game from the initial one is bounded by the probability of the eliminated collisions, independently of the power of the attacker.
- performing up-to-bad reasoning: the final game can be distinguished from the initial one only when an event is raised, and CRYPTOVERIF bounds the probability of that event, following Shoup's lemma [22].
- guessing transformations, in which one guesses the tested session, the value of a variable, or which branch of a test is taken. For these transformations, the effect on probabilities is also independent of the attacker.
- the application of a cryptographic assumption. In CRYPTOVERIF, cryptographic assumptions are represented by indistinguishability axioms $L \approx_p R$ (such an axiom for Indistinguishability under Adaptive Chosen-Ciphertext Attacks (IND-CCA2) of KEMs is given below). Such an axiom is applied to a game Q by finding a context C such that $Q \approx_0 C[L]$, replacing L with R inside C and simplifying the result, yielding $Q \approx_0 C[L] \approx_{p'} C[R] \approx_0 Q'$, with $p'(\mathcal{A}, C') = p(\mathcal{A}, C'[C])$. This reasoning corresponds to a proof by reduction and is valid as long as the axiom is valid.

All those techniques are intuitively post-quantum sound, and CRYPTOVERIF did not rely on any dangerous technique such as rewinding. However, it was crucial to go over all the soundness proofs of the existing game transformations, propagating the presence of the black-box attacker, and verifying that indeed the state of the attacker was never reasoned over.

We adapted in [21] the soundness proofs of all game transformations, that we do not detail any more here. In effect, the core argument was to make the semantics more abstract in terms of the attacker, and ensure that this propagated correctly in all the remaining proofs. No major technical hurdle was encountered, and the propagation was thus mainly a long and careful task.

Formally, we thus have that the semantics and all transformations are valid against a black-box interactive attacker. This class of attackers is very broad, and does not in fact make any assumption on the computational power of the attacker: the only restriction is that it interacts classically over the network with the other agents. This class notably includes the quantum attacker, which computational power is then restricted by the cryptographic axioms.

From a user point of view on the implemented tool, our main result can be summarized as an informal core theorem.

Theorem 1 (Informal PQ soundness of CRYPTOVERIF) *A* CRYPTOVERIF *proof only relying on post-quantum sound cryptographic assumptions holds against a quantum attacker.*

C. Cryptographic Assumptions

CRYPTOVERIF comes with a library of pre-defined cryptographic assumptions, written in an optimized fashion for its automated reasoning. It includes symmetric and asymmetric encryptions with various security assumptions, groups with Diffie-Hellman assumptions (DDH, Computational Diffie-Hellman (CDH), Gap Diffie-Hellman (GDH), ...), MACs and signatures with Existential Unforgeability under Adaptive Chosen Message Attacks (EUF-CMA), the ROM, collision-resistance, and PRF assumptions for hash functions, ... We adapted this library for post-quantum proofs.

a) KEMs: Since KEMs are essential post-quantum primitives, and were not included in the library of primitives, we added them. A KEM is defined by three algorithms:

- $sk_e, pk_e \xleftarrow{R} \mathrm{KEM.keygen}()$ is used to sample a fresh secret/public key-pair.
- $ct, ss \xleftarrow{R} \mathrm{encaps}(pk_e)$ is used to encapsulate a new shared secret ss based on the public key pk_e, producing the ciphertext ct. One can think of it as sampling a fresh secret ss, then using an asymmetric encryption scheme to encrypt this shared secret for some public key.

Game Q_0 :

$O_{init} := r \xleftarrow{R} T_{keyseed}$; return(pkgen($r$));
  foreach $i \leq n$ do $O_e(x : pkey) :=$
    return(encaps(x))
| foreach $i \leq n$ do $O_d(x : bitstring) :=$
    return(decaps(x, skgen(r)))

Game Q_1 :

$O_{init} := r \xleftarrow{R} T_{keyseed}$; return(pkgen($r$));
  foreach $i \leq n$ do $O_e(x : pkey) :=$
    if $x =$ pkgen(r) then
      $k \xleftarrow{R} T_{ss}$; $ct, ss \xleftarrow{R}$ encaps(x);
      insert $E(k, ct)$; return((ct, k))
    else return(encaps(x))
| foreach $i \leq n$ do $O_d(x : bitstring) :=$
    get $E(k, ct)$ suchthat $x = ct$ in
      return(k)
    else return(decaps(x, skgen(r)))

Fig. 4. IND-CCA2 for KEMs in CryptoVerif

- Finally, $ss \leftarrow \mathrm{decaps}(ct, sk_e)$ enables one to decapsulate the ciphertext, yielding the shared secret ss or in case of failure a dedicated constant $\perp$.

We added to the library the suitable IND-CCA2 assumption for KEMs. While the IND-CCA2 assumption for asymmetric encryption states that the encryption of any two messages must be indistinguishable, the KEM variant only asks that encryptions of any two random keys be indistinguishable. To define this assumption in CryptoVerif, we can, similarly to Example 1, first define a game where we set up one honest public key and the attacker has access to a decapsulation and encapsulation oracle for this key, as illustrated by game Q_0 in Fig. 4. In this game, the functions pkgen and skgen generate the public key, resp. secret key, from a random seed r, so that we have $\mathrm{KEM.keygen}() = r \xleftarrow{R} T_{keyseed}$; return(skgen($r$), pkgen($r$)). The function encaps is defined as an abbreviation $\mathrm{encaps}(x : pkey) = r' \xleftarrow{R} T_{encapsseed}$; return(encaps_r($x, r'$)) so that it generates random coins r' and calls the deterministic function encaps_r with these coins. The game Q_0 is the real-world version, where intuitively the attacker will interact with agents performing honest encapsulations and decapsulations. We then express IND-CCA2 by assuming that this real-world is indistinguishable from an idealized world defined as follows. For each new encapsulation for the honest public key, we sample a completely fresh key, and we store inside a table E the link between the ciphertext and this fresh key. The decapsulation oracle checks whether the attacker provided a ciphertext that corresponds to an honest one stored inside the table, in which case it returns the ideal secret key. Otherwise, it decapsulates normally. We formalize this intuition as Q_1 in Fig. 4, and finally the IND-CCA2 assumption corresponds to stating that $Q_0 \approx_{pIND} Q_1$. (This is a simplified single public key version for clarity.) Inside a security proof, this assumption allows us to replace the shared secret derived by two honest parties by the fresh secret k fully independent of anything seen by the attacker.

We also added a definition for Indistinguishability under Chosen-Plaintext Attacks (IND-CPA) of KEMs, similar to the IND-CCA2 definition but without decapsulation oracle.

b) Insecure Primitives: We added definitions for primitives with only assumptions that do not depend on the power of the attacker, that is, we keep definitions of functions, functional assumptions (such as decrypting a ciphertext with the correct key yields the cleartext) and collisions whose probability does not depend on the attacker, but we remove security assumptions. Then, a one-line change in a model allows us to switch from a secure primitive to an insecure one in the presence of quantum attackers.

c) The Post-Quantum Library: Combing through the library, we identified which assumptions do not have a post-quantum instantiation, typically using the NIST standardization process as a witness. Overall, only the Diffie-Hellman assumptions and the ROM do not have valid instantiations. We built a variant of the library only containing post-quantum sound assumptions, which enables us to state our final practical result: CryptoVerif proofs only relying on the post-quantum library yield concrete guarantees against quantum attackers.

IV. Case Studies

Given the importance of the corresponding protocols, we chose to start with proofs for the IETF drafts for hybrid variants of TLS 1.3 [5] and SSH [6]. A summary of our security results can be found in Table I, and we detail thereafter each of our modelings and proof method.

A. Core Idea of Hybrid Proofs

In this section, we give the main ideas of how hybrid key-exchanges are built, and how we can prove their security in CryptoVerif. We thus abstract away from TLS and SSH, and consider a generic signed DH key exchange, where two parties exchange ephemeral DH shares g^a and g^b, and compute a shared secret g^{ab}. To hybridize it, the core idea is to rely on a KEM that has a post-quantum secure instantiation.

a) Hybridizing a DH-Based Key-Exchange: We show in Fig. 5 a common pattern used to hybridize a DH-based key exchange, by adding a KEM computation in parallel. All hybrid parts in this figure are highlighted with a light gray background. The core idea is to integrate inside the key derivation a KEM shared secret, by using a dual-PRF $\mathrm{dPRF}(g^{ab}, ss)$. A dual-PRF ensures that if either of the two arguments is a fresh random secret, then the output can be securely used as a derived key. That is, the security of only one of the constructs is enough to get the full security. There are in fact multiple ways to define such a dual-PRF, and in fact to combine KEMs, as explored in the so-called KEM combiners [23], [24]. We will see some of the concrete instantiations over TLS and SSH.

Since dPRF takes as first argument the DH shared secret, it must be a PRF keyed with a random group element, and not

Protocol	Security Property	Cryptographic assumptions	Run-time	Model size
SSH [6]	Authentication Forward Secrecy	EUF-CMA, snPRF-ODH, PRF, CR	12 s	$\sim$ 420 LoC 26 steps
	Post-Quantum Forward Secrecy	EUF-CMA, IND-CCA2, PRF, CR	9 s	
TLS [5] (0,0.5,1-RTT,PSK,Client Auth)	Authentication (both sides) Forward Secrecy	EUF-CMA, snPRF-ODH, PRF, CR	1028 s	$\sim$ 2000 LoC 18+10+33 steps
	Post-Quantum Forward Secrecy	EUF-CMA, IND-CCA2, PRF, CR	933 s	

TABLE I

SUMMARY OF THE PROPERTIES PROVED IN CRYPTOVERIF

The model size is in the number of Line of Codes (LoC), it is only to give an intuition on the relative complexity of the protocols and models. The steps correspond to the number of manually specified tactics that guide the proof. The run-time is the total time needed by CRYPTOVERIF to automatically verify and complete the proof script. A laptop was used with 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz and 30Gb of RAM. TLS is made of 3 separate models capturing its possible handshake modes, generated by preprocessing; we give the total number of lines of the source files and the number of steps for each proof (18 steps for the initial handshake, 10 for the handshake with non compromised PSK, and 33 for the handshake with compromised PSK).

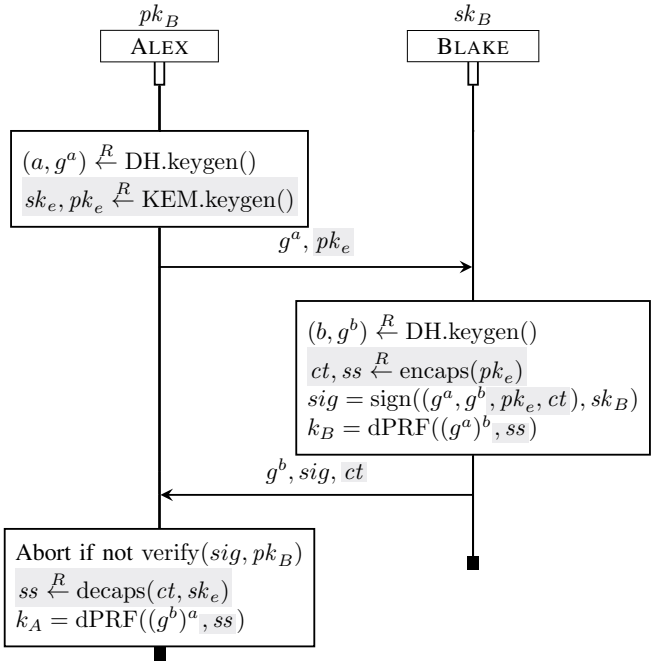

Fig. 5. Hybridization of a signed Diffie-Hellman key exchange

just one keyed with a random bit-string as usual. Moreover, one often combines the PRF and DH assumptions, using the Pseudorandom Function Oracle Diffie-Hellman (PRF-ODH) assumption [25]. CRYPTOVERIF supports several variants of PRF-ODH. The most basic PRF-ODH notion, nnPRF-ODH, says that an attacker that has g^a and g^b for random a, b has a negligible probability of distinguishing $x \mapsto \text{dPRF}(g^{ab}, x)$ from a random function. In general, the lrPRF-ODH notion additionally gives the attacker access oracles $O_\text{a}(x, Y)$, which returns $\text{dPRF}(Y^a, x)$ when $Y^a \neq g^{ab}$, and $O_\text{b}(x, X)$, which returns $\text{dPRF}(X^b, x)$ when $X^b \neq g^{ab}$, where the letter $l \in \{\text{n, s, m}\}$ (resp. $r \in \{\text{n, s, m}\}$) determines how many times oracle O_a (resp. O_b) can be called: n means never, s means a single time, m means multiple times. Both oracles return $\perp$ when they would compute $\text{dPRF}(g^{ab}, x)$, which the attacker tries to distinguish from random. Their presence allows one to prove the protocol secure even if the attacker can test whether its element Y or X is the expected one, for instance by seeing whether a session continues. (This definition is similar to the one of [25]. It still differs in that in our more generic definition, the order of oracle calls does not matter, so that lrPRF-ODH is equivalent to rlPRF-ODH, and the attacker may evaluate $x \mapsto \text{dPRF}(g^{ab}, x)$ several times to distinguish it from a random function.) nnPRF-ODH can be proved from CDH and ROM, or from DDH and PRF; mmPRF-ODH from GDH and ROM.

b) Security Properties: In the classical setting, we consider two main security properties:

- *authentication*: whenever the Initiator derives a key, the Responder is authenticated to the Initiator and both agents agree on the data used to derive the key;
- *forward secrecy*: as long as the long term keys were not compromised when the key exchange took place, the derived keys will remain secret.

On a DH-based key exchange, we typically expect to prove a theorem such as: "*Assuming the DH group and the* dPRF *satisfy PRF-ODH and the signature is EUF-CMA, the key exchange satisfies both authentication and forward secrecy*".

Over a hybrid version, the first goal is still to prove such a theorem, to demonstrate that the update did not downgrade the security of the key exchange against classical attackers.

The second goal is to prove once again the *forward secrecy*, assuming that quantum attackers do not exist yet but will exist in the future. Crucially, we do not make assumptions over the security of DH values and we consider that the signature scheme is EUF-CMA up to a point. We then prove that "*Assuming that the KEM is IND-CCA2 against a quantum attacker,* dPRF *is a PRF against a quantum attacker where the key is the second argument, the signature is EUF-CMA against a classical attacker, and quantum attackers do not exist yet when the key exchange is performed, the key-exchange satisfies forward secrecy against a quantum attacker*".

Interestingly, when proving forward secrecy, the property already explicitly allows for the compromise of the long-term signing key. Such a compromise can be seen as modeling an attacker suddenly having access to a quantum computer, and being able to compromise the signature scheme.

To reach both goals in our CRYPTOVERIF analyses, we will thus make two distinct proofs for each protocol model, one

under PRF-ODH relying on the DH exchange, and another one under IND-CCA2 and PRF relying on the KEM.

Notice that a third possible goal is to show authentication and forward secrecy, assuming quantum attackers exist during the key exchange. For this property, we would show that "*Assuming that, against a quantum attacker, the KEM is IND-CCA2,* dPRF *is a PRF where the key is the second argument, the signature is EUF-CMA, the key-exchange satisfies authentication and forward secrecy against a quantum attacker*". This goal is not considered in the candidate TLS and SSH drafts that we look at: they do not consider post-quantum signatures. Moreover, for TLS, considering this goal would require updating the whole certificate chains to use post-quantum signatures. However, we remark that our CRYPTOVERIF proofs for the second goal also show this result, assuming post-quantum signatures, because they actually show that "*Assuming that the KEMs is IND-CCA2,* dPRF *is a PRF where the key is the second argument, and the signature is EUF-CMA when the key exchange is performed, the key-exchange satisfies authentication and forward secrecy*". Note that, since post-quantum signatures provide the same security property (EUF-CMA) as classical signatures (although against different attackers), we can reuse the same assumption for signatures in CRYPTOVERIF.

c) Proof Methodology: When given an initial game stating the security of a protocol Q, CRYPTOVERIF tries to automatically build a sequence of transformations in order to obtain an indistinguishability $Q \approx_p Q_i$, where the security is obvious in Q_i, and we then get a bound on the advantage of the attacker with the probability p given by CRYPTOVERIF.

On simple protocols, the automated heuristic is very efficient and can often directly build a proof. On our case studies, we however have to guide the proof, by specifying in the CRYPTOVERIF tactic language the main steps of the proof.

In all our case studies, whether we make the proof under the DH-based assumption or the KEM-based assumption, we guide the proof by following some rather similar steps:

1) Given the game $A \mid B$ where A is the code of Alex and B the code of Blake, we first duplicate their code based on a conditional check. We test in B whether the DH ephemeral, resp. KEM public-key, it receives has been generated by A. This test is implemented using the find construct. The code that follows the test is duplicated: let B_h be the copy when the test succeeds (B received honest values) and B_d the one when the test fails (B received dishonest values). Similarly, we test in A whether the DH ephemeral, resp. KEM ciphertext, has been sent by B_h. We define A_h and A_d similarly to B_h and B_d. CRYPTOVERIF enables us to introduce such tests, as they preserve the possible executions.
2) Next, we use EUF-CMA, to show that A_d never succeeds, because the signature check must fail (as by definition A_d received some dishonest values);
3) We now consider the ephemeral DH or KEM values used jointly by A_h and B_h, and apply the respective assumptions to turn the derived key into a fresh key and conclude the proof: we use PRF-ODH for DH to directly turn the derived key into a fresh key, resp. IND-CCA2 to first turn the KEM shared secret into a fresh key and next PRF to turn the derived key into a fresh key.

Note that to prove the forward secrecy property, we always prove some authentication property first, as it allows us to use the security assumptions over the honest DH or KEM values.

All the security theorems presented next follow this high-level schema, with adaptations based on the concrete case.

B. Hybrid TLS 1.3

TLS is used to secure internet browsing, and is thus a crucial target. During the development of TLS 1.3, its draft 18 was analyzed with CRYPTOVERIF in [7], [26], [27]. In this work, we update the models to meet the actual final Request For Comments (RFC) of the standard, as well as analyze the current draft [5] for a hybrid variant of it. Compared to the high-level view of Fig. 5, there are major differences complexifying the analysis, among which:

- Several intermediate keys are derived during the key exchange in order to encrypt early data messages (the 0-RTT messages) and parts of the handshake.
- The key derivation contains multiple calls to some underlying key derivation functions.
- After some initial handshake and first key derivation, it is possible upon a new connection to do a new handshake that reuses one of the keys derived by the first handshake as a Pre-Shared Key (PSK).
- After the first back and forth, the initiator can also include in the last handshake message a signature to authenticate itself (client authentication).
- Some initial messages take place during a so-called algorithm negotiation, where the client and the server try to agree on the ciphersuites to use.

The original CRYPTOVERIF models captured for draft 18 the handshakes with DH and/or PSK, with 0-RTT, 0.5-RTT, and 1-RTT, optional client authentication, and the possible compromise of signature keys and of the PSK. They do not include algorithm negotiation, resumption tickets (which may be used to store the PSK), nor post-handshake client authentication. They delegate handshake encryption to the attacker, by publishing the handshake encryption keys; hence, the CRYPTOVERIF models give more power to the attacker than models including handshake encryption. We update the original CRYPTOVERIF models while preserving their scope.

a) Update to RFC from Draft 18: The main difference between the final RFC [28] and draft 18 [29], and the only one that impacts our proofs, is inside the key derivation, which was slightly updated. The TLS key derivation relies on the HKDF construction [30], where a function $\mathrm{HKDF\text{-}extract}(IKM, salt)$ is used with some Input Keying Material IKM to extract some clean randomness key, that can then be used to derive many independent secrets with $\mathrm{Derive}(key, label)$ where two distinct labels yield two independent secrets. In draft 18, the key derivation proceeds as:

- $ES = \text{HKDF-extract}(PSK, 0)$
- $HS = \text{HKDF-extract}(DHE, ES)$
- $MS = \text{HKDF-extract}(0, HS)$

When the PSK or the DH shared secret DHE is absent, it is replaced with 0. Other keys are derived from ES, HS, and MS by calls to Derive. A problem with this derivation is that Derive and HKDF-extract rely on the same function HMAC, so to be sure that there is no collision between $HS = \text{HKDF-extract}(DHE, ES)$ and other keys derived from ES by Derive, we need the assumption that the arguments of HMAC in these calls do not collide, and similarly for keys derived from HS. The final RFC removed this assumption by updating the key derivation as:

- $HS = \text{HKDF-extract}(DHE, \text{Derive}(ES, label_1))$
- $MS = \text{HKDF-extract}(0, \text{Derive}(HS, label_2))$

and using distinct labels for all calls to Derive.

The CRYPTOVERIF proofs use separate lemmas to prove properties about the key schedule. We just had to update these lemmas to take into account this change, while the main proofs of the TLS handshake remain unchanged.

b) The Hybrid Proposal: The hybrid TLS 1.3 proposal from [5] aims at enabling a hybrid version with minimal modifications, to ease deployment. It thus follows very closely our high-level design presentation, adding a KEM exchange to obtain an additional shared secret ss. Then, it proposes to integrate it inside the key schedule by updating the HS derivation to $HS = \text{HKDF-extract}(DHE \| ss, \text{Derive}(ES, label_1))$.

c) Proofs: Taking those changes into account, we adapted the existing CRYPTOVERIF models, that consisted of three main files capturing the possible execution modes of TLS (with or without a PSK, that may itself be compromised) by:

- refactoring them so that the shared secret derivation can be modified modularly;
- adding a 204-line file that models the hybrid key exchange of [5], producing $DHE \| ss$ as shared secret.

With such an adaptation, we only had to slightly update the previous classical proofs of secrecy and authentication in order to (re)-obtain the classical security properties.

Theorem 2 (Classical TLS 1.3 security, simplified) *Under the snPRF-ODH assumption for the group and the HMAC function, the EUF-CMA assumption for signatures, the PRF assumption for HMAC, and the Collision Resistance (CR) assumption for the hash function, the hybrid TLS 1.3 handshake ensures authentication and forward secrecy. (Authentication is bilateral when the client is authenticated; only the server is authenticated otherwise.)*

We however had to do from scratch the new proof for the security part obtained thanks to the additional KEM.

Theorem 3 (Post-quantum TLS 1.3 security, simplified) *Under the post-quantum IND-CCA2 assumption for the KEM, the post-quantum PRF assumption for HMAC, the post-quantum CR assumption for the hash function, and the classical EUF-CMA assumption for signatures, the hybrid TLS 1.3 handshake ensures forward secrecy even against quantum attackers, provided quantum attackers do not exist yet when the handshake is performed.*

All the assumptions for this theorem are consequences of past cryptographic results, except for the PRF assumption. Indeed, the key derivation contains a call of the form $\text{HMAC}(i, DHE \| ss)$, which we see as being a PRF keyed by ss. This is unavoidable with the current design. We conjecture that this assumption could be fulfilled by adapting the proof from [31] that HMAC is a dual-PRF to such a use-case.

In complement, we also proved that under the assumption that the PSK is secure, as implied by resp. Theorem 2 or 3, and the PRF assumption for HMAC, the PSK handshake also ensures resp. classical or post-quantum security.

C. Hybrid SSH

The SSH key exchange protocol is built as a more classical signed DH than TLS, and is way simpler. Its up-to-date version was previously verified with CRYPTOVERIF in [8].

For the hybridization, it follows the high-level design previously presented: the shared key is derived as $KS = \text{dPRF}(DHE, ss) = h(DHE \| ss)$ where h is a hash function (SHA-256 or SHA-512). We assume that dPRF satisfies the PRF assumption considering ss as the key and the snPRF-ODH assumption jointly with the Diffie-Hellman group considering DHE as the key. Yet, SSH still posed a challenge, due to its design following old guidelines that complicate the proof.

a) Hash Modeling: The core issue is that instead of authenticating by using a signature of the form $\text{sign}((g^a, g^b, pk_e, ct), sk_B)$, what SSH does is to use $\text{sign}(H, sk_B)$ where $H = h(g^a \| g^b \| pk_e \| ct \| KS)$ is a session id. Notice two important differences:

- The transcript is hashed before being signed, so to make sure that the signature also authenticates the transcript, we need the CR assumption on the hash function.
- The shared key KS is included inside the hash. Since EUF-CMA signatures may leak the signed message, we need to make sure that h hides KS. We could achieve that by assuming that $f(x, KS) = h(x \| KS)$ is a PRF, considering KS as the key.

To be able to encode both assumptions, we cannot use the standard trick of adding a key to the collision-resistant hash function, to model the choice of the hash function itself. We need to use a notion of collision-resistance for hash functions without keys, as explained in [32]. Although we know that collisions exist because the range of the hash is smaller than its input domain, this notion models that we do not know how to construct an attacker that finds a collision. We now intuitively prove that if one effectively knows how to build an attacker on the protocol with probability p, then one effectively knows how to build an attacker on the assumptions with probability p'. And instead of assuming that no attacker can win CR, we in fact assume that today, nobody knows how to effectively

construct one. To make a valid proof in this setting, one has to make sure that the proofs are constructive, which is in fact the case for CRYPTOVERIF.

Finally, SSH computes the transport keys by $h(KS\|H\|l\|H)$, where l is a letter among A, B, C, D, E, F. To show that these keys are random, we also need that $g(KS, x) = h(KS\|x)$ is a PRF, for fixed-length x, with the same key KS as f above. Since the keys are the same, we would need a joint PRF assumption on f and g.

Instead of using that unusual assumption, we rely on the same solution as [33]: we consider the specific computation PRFc performed in SSH using f and g and defined below, and we assume that it is a PRF:

$$\begin{aligned}&\mathrm{PRFc}(K, x):\\ &\ H \leftarrow h(x\|K)\\ &\ \mathsf{for}\ l \in \{\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D},\mathrm{E},\mathrm{F}\}, K_l \leftarrow h(K\|H\|l\|H)\\ &\ \mathsf{return}(H\|K_\mathrm{A}\|K_\mathrm{B}\|K_\mathrm{C}\|K_\mathrm{D}\|K_\mathrm{E}\|K_\mathrm{F})\end{aligned}$$

As mentioned in [33], one can prove that PRFc is a PRF from the ROM assumption on h. They, and we, leave for future work the proof of that property in the standard model.

b) Proofs: We adapted the existing models by first removing the ROM and assuming that h is CR and PRFc is a PRF, and then making the hybridization with the KEM. The first change forced us here to fully redo the existing classical proofs, previously made in the ROM, in order to obtain the classical security.

Theorem 4 (Classical SSH security, simplified) *Under the snPRF-ODH assumption for the group and* dPRF*, the CR assumption for* h*, the PRF assumption for* PRFc*, and the EUF-CMA assumption for signatures, the hybrid SSH key exchange ensures unilateral authentication and forward secrecy.*

The snPRF-ODH assumption is needed for proving forward secrecy of SSH, in order to simulate reveal queries for client keys in case the server is corrupted. The nnPRF-ODH assumption would be sufficient without forward secrecy; this is coherent with the remark of [33] that DDH is sufficient, since [33] does not prove forward secrecy.

We also did from scratch the post-quantum proof.

Theorem 5 (Post-quantum SSH security, simplified) *Under the post-quantum IND-CCA2 assumption for the KEM, the post-quantum PRF assumption for* dPRF *and* PRFc*, the CR assumption for* h*, the classical EUF-CMA assumption for signatures, the hybrid SSH key exchange ensures forward secrecy even against quantum attackers, provided quantum attackers do not exist yet when the handshake is performed.*

D. Lessons Learned

a) Adaptability of CRYPTOVERIF *Proofs:* Whenever a protocol is updated in any way, it is crucial to prove again the security of the protocol. Notably, it is common that even just an extension of a protocol may downgrade the security of its original core. With a pen-and-paper proof, if there is an update, one has to carefully go through each and every step of the proof to verify that it still holds.

With our case-studies, we have seen a valuable strength of CRYPTOVERIF: there is a form of proof stability w.r.t. protocol extensions. Indeed, all the proof guidance of the previous version mostly carried over to the hybridized version. Intuitively, the proof steps that have to be specified manually mostly talk about the core security argument, and a correct extension should not change this core.

b) Hybridizing Existing Protocols: Our post-quantum CRYPTOVERIF proofs advocate that if one follows carefully a few guidelines, existing complex key exchanges can indeed be made post-quantum secure with relatively small changes. For future developments, we outline here two important sufficient conditions for provable security over the hybridization:

- including the KEM ephemerals inside the authenticated data helps to carry out a straightforward proof;
- ensuring that the DH and KEM secrets are correctly combined, typically following generic KEM combiners; if they are concatenated, the respective lengths of the secrets must be fixed to enable the proof.

Those two implicit guidelines were followed for the SSH and TLS drafts, which greatly eased our proofs.

c) Symmetric primitives: Even though we make the same assumptions on symmetric primitives in the classical and post-quantum settings, we still need to check if these assumptions can hold against quantum attackers, and with which level of security. According to NIST [34, last question], this may be valid keeping the same primitives and key lengths, at least for AES.

V. CONCLUSION

We have updated CRYPTOVERIF, making it closer to classical cryptographic proofs, as well as establishing its post-quantum soundness.

We demonstrate the importance of our approach on two IETF drafts under discussion, formally obtaining the first mechanized security proofs for those proposals, while we are also not aware of handwritten proofs.

CRYPTOVERIF now appears well suited to take part in some more design and standardization processes of hybrid protocols, as well as fully post-quantum ones. Our main limitation is that we do not cover the qROM. On the protocol side, it is not needed for most key-exchanges, our main current target. This would however be a true limitation in order to prove the security of some primitives, as well as specific protocols like e-voting.

Acknowledgments: This work was partly done while Charlie Jacomme was at Inria, Paris, France. This work benefited from funding managed by the French National Research Agency under the France 2030 programme with the references ANR-22-PECY-0006 (PEPR Cybersecurity SVP) and ANR-22-PETQ-0008 (PEPR Quantic PQ-TLS).

References

[1] C. Boyd, Y. Cliff, J. M. G. Nieto, and K. G. Paterson, "One-round key exchange in the standard model," *International Journal of Applied Cryptography*, vol. 1, no. 3, p. 181, 2009. [Online]. Available: http://www.inderscience.com/link.php?id=23466

[2] A. Fujioka, K. Suzuki, K. Xagawa, and K. Yoneyama, "Strongly secure authenticated key exchange from factoring, codes, and lattices," in *Public Key Cryptography – PKC 2012*, ser. Lecture Notes in Computer Science, vol. 7293. Springer, May 2012, pp. 467—-484, extended version available at https://eprint.iacr.org/2012/211.

[3] P. Schwabe, D. Stebila, and T. Wiggers, "Post-quantum TLS without handshake signatures," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1461–1480.

[4] E. Kret and R. Schmidt, "The PQXDH key agreement protocol." [Online]. Available: https://signal.org/docs/specifications/pqxdh/

[5] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-hybrid-design-09, Sep. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/09/

[6] P. Kampanakis, D. Stebila, and T. Hansen, "Post-quantum Hybrid Key Exchange in SSH," Internet Engineering Task Force, Internet-Draft draft-kampanakis-curdle-ssh-pq-ke-01, Apr. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-kampanakis-curdle-ssh-pq-ke/01/

[7] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *IEEE Symposium on Security and Privacy (S&P'17)*. IEEE, May 2017, pp. 483–503.

[8] D. Cadé and B. Blanchet, "From computationally-proved protocol specifications to implementations and application to SSH," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 4, no. 1, pp. 4–31, Mar. 2013.

[9] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, Oct. 1982. [Online]. Available: http://www.nature.com/articles/299802a0

[10] J. Watrous, "Zero-knowledge against quantum attacks," *SIAM Journal on Computing*, vol. 39, no. 1, pp. 25–58, 2009.

[11] A. Ambainis, A. Rosmanis, and D. Unruh, "Quantum attacks on classical proof systems: the hardness of quantum rewinding," in *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 2014, pp. 474–483.

[12] F. Song, "A Note on Quantum Security for Post-Quantum Cryptography," *arXiv:1409.2187 [quant-ph]*, Sep. 2014, arXiv: 1409.2187. [Online]. Available: http://arxiv.org/abs/1409.2187

[13] T. Gagliardoni, "Quantum Security of Cryptographic Primitives," *arXiv:1705.02417 [quant-ph]*, May 2017, arXiv: 1705.02417.

[14] D. O'Brien, "Protecting Chrome traffic with hybrid Kyber KEM," https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html, Aug. 2023.

[15] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, Aug. 2011, pp. 71–90.

[16] M. Barbosa, G. Barthe, X. Fan, B. Grégoire, S.-H. Hung, J. Katz, P.-Y. Strub, X. Wu, and L. Zhou, "EasyPQC: Verifying post-quantum cryptography," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2564–2586.

[17] D. Baelde, S. Delaune, A. Koutsos, C. Jacomme, and S. Moreau, "An interactive prover for protocol verification in the computational model," in *42nd IEEE Symposium on Security and Privacy (S&P'21),*. IEEE, May 2021, pp. 537–554.

[18] C. Cremers, C. Fontaine, and C. Jacomme, "A logic and an interactive prover for the computational post-quantum security of protocols," in *2022 IEEE Symposium on Security and Privacy (S&P'22)*. IEEE, 2022, pp. 125–141.

[19] G. Bana and H. Comon-Lundh, "A computationally complete symbolic attacker for equivalence properties," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 609–620.

[20] B. Blanchet and D. Pointcheval, "Automated security proofs with sequences of games," in *Advances in Cryptology – CRYPTO 2006*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Springer, Aug. 2006, pp. 537–554.

[21] B. Blanchet and C. Jacomme, "CryptoVerif: a computationally-sound security protocol verifier," Inria, Research report RR-9526, Oct. 2023, available at https://inria.hal.science/hal-04253820.

[22] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, Nov. 2004, available at http://eprint.iacr.org/2004/332.

[23] F. Giacon, F. Heuer, and B. Poettering, "KEM combiners," in *Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I 21*, ser. Lecture Notes in Computer Science, vol. 10769. Springer, 2018, pp. 190–218.

[24] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila, "Hybrid key encapsulation mechanisms and authenticated key exchange," in *Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, J. Ding and R. Steinwandt, Eds., vol. 11505. Springer, 2019, pp. 206–226.

[25] J. Brendel, M. Fischlin, F. Günther, and C. Janson, "PRF-ODH: Relations, instantiations, and impossibility results," in *CRYPTO 2017*, ser. Lecture Notes in Computer Science, vol. 10403. Springer, Aug. 2017, pp. 651–681.

[26] B. Blanchet, "Composition theorems for CryptoVerif and application to TLS 1.3," in *31st IEEE Computer Security Foundations Symposium (CSF'18)*. IEEE, Jul. 2018, pp. 16–30.

[27] ——, "Dealing with dynamic key compromise in CryptoVerif," in *37th IEEE Computer Security Foundations Symposium (CSF'24)*. IEEE, Jul. 2024, preprint available at https://inria.hal.science/hal-04271666.

[28] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," RFC 8446, https://www.rfc-editor.org/rfc/rfc8446, Aug. 2018.

[29] ——, "The Transport Layer Security (TLS) protocol version 1.3, draft-ietf-tls-tls13-18," https://tools.ietf.org/html/draft-ietf-tls-tls13-18, Oct. 2016.

[30] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology (CRYPTO)*, ser. Lecture Notes in Computer Science, vol. 6223. Springer, 2010, pp. 631–648.

[31] M. Backendal, M. Bellare, F. Günther, and M. Scarlata, "When messages are keys: Is HMAC a dual-PRF?" in *Advances in Cryptology – CRYPTO 2023*, ser. Lecture Notes in Computer Science, vol. 14083. Springer, 2023, pp. 661–693.

[32] P. Rogaway, "Formalizing human ignorance: Collision-resistant hashing without the keys," in *International Conference on Cryptology in Vietnam*, ser. Lecture Notes in Computer Science, vol. 4341. Springer, 2006, pp. 211–228.

[33] F. Bergsma, B. Dowling, F. Kohlar, J. Schwenk, and D. Stebila, "Multi-ciphersuite security of the secure shell (SSH) protocol," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 369–381.

[34] NIST, "Post-quantum cryptography, faqs." [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/faqs