Generalized Proofs of Knowledge with Fully Dynamic Setup

Christian Badertscher[1⋆][0000−0002−1353−1922], Daniel Jost[2⋆⋆][0000−0002−6562−9665], and Ueli Maurer[3]

[1] IOHK, Zurich, Switzerland – christian.badertscher@iohk.io
[2] New York University, USA – daniel.jost@cs.nyu.edu
[3] ETH Zurich, Switzerland – maurer@inf.ethz.ch

Abstract. Proofs of knowledge (PoK) are one of the most fundamental notions in cryptography. The appeal of this notion is that it provides a general template that an application can suitably instantiate by choosing a specific relation. Nonetheless, several important applications have been brought to light, including proofs-of-ownership of files or two-factor authentication, which do not fit the PoK template but naturally appear to be special cases of a more general notion of proofs of knowledge or possession. One would thus expect that their security properties, in particular privacy and soundness, are simply derived as concrete instantiation of a common generalized PoK concept with well understood security semantics. Unfortunately, such a notion does not exist, resulting in a variety of tailor-made security definitions whose plausibility must be checked on a case-by-case basis.

In this work, we close this gap by providing the theoretical foundations of a generalized notion of PoK that encompasses dynamic and setup-dependent relations as well as interactive statement derivations. This novel combination enables an application to directly specify relations that depend on an assumed setup, such as a random oracle, a database or ledger, and to have statements be agreed upon interactively and dynamically between parties based on the state of the setup. Our new notion is called *agree-and-prove* and provides clear semantics of correctness, soundness, and zero-knowledge in the above generalized scenario.

As an application, we first consider proofs-of-ownership of files for client-side file deduplication. We cast the problem and some of its prominent schemes in our agree-and-prove framework and formally analyze their security. Leveraging our generic zero-knowledge formalization, we then devise a novel scheme that is provably the privacy-preserving analogue of the well-known Merkle-Tree based protocol. As a second application, we consider two-factor entity authentication to showcase how the agree-and-prove notion encompasses proofs of ability, such as proving the correct usage of an abstract hardware token.

⋆ Work done in part while author was at the University of Edinburgh, Scotland, and at ETH Zurich, Switzerland.

⋆⋆ Research supported by the Swiss National Science Foundation (SNF) via Fellowship no. P2EZP2_195410. Work done in part while at ETH Zurich, Switzerland.

1 Introduction

The concept of an *interactive proof* in which a prover's goal is to convince a verifier of the validity of a given statement is a fundamental theoretical concept in complexity theory and is established as a cornerstone in cryptography as well. Especially the task of proving to a party that one knows a certain piece of information, without necessarily revealing it, is an essential task in cryptography and in the design of cryptographic protocols. The formal concept capturing the essence of this task is called *proof of knowledge* [10, 16, 8, 3] and has turned out to be a building block with countless applications. In a nutshell, the task of a prover is to convince the verifier that he knows a witness w for a statement x satisfying a relation $R(w, x)$. Part of the elegance of this definition, fostering its wide applicability, is that it does not make any particular assumption about the statements or witnesses, that is, the definition is independent of how statements are generated. Furthermore it does not make many assumption about the relation except that for cryptographic applications relations based on hardness assumptions are typically considered.

The formalization as a rather low-level building block has however a major downside when it comes to capturing the security of natural (higher-level) interactive proofs occurring in practice for at least two reasons: First, statements and witnesses are traditionally treated as rather rigid objects of the definition in the sense that they are considered as static objects provided as inputs to the respective parties. In real-world settings, however, a much more dynamic interaction can be observed: typically, we have two parties, both with a certain prior state, to approach each other and first (interactively) *agree* on a statement (and potentially the prover generating a proper witness), and only then *prove* the agreed statement. This was first identified as a problematic shortcoming by Camenisch et al. [5] who put forth a formal treatment of an appropriate PoK generalization that reflects such a two-stage process. Clearly, complex theoretical questions and definitional challenges arise from the interplay between these two phases. For example, the first phase (which we call agreement phase) might have in general impact on the obtained security guarantees: on the one hand, an involved agreement phase might be followed by a more efficient proof [5, 2] while, on the other hand, an agreement phase does pose additional challenges such as to retaining the desired zero-knowledge guarantees. Hence, the agreement phase plays a crucial role in practice that cannot be neglected.

Another shortcoming stems from the rigid treatment of the proof relation (the above generalization falls into this category). First steps towards having a more dynamic view on the relation were taken in the form of relation generators that outputs an explicit description of a relation to the parties [18, 7] (and afterwards, a statement can be adaptively chosen by the adversary). While more dynamic in nature, this treatment leaves again a substantial gap: In many real-world situations, a setup (such as a database, a ledger, a random oracle, or quantum devices) takes a crucial role in defining the relation that a prover asserts to a verifier. While certain setup functionalities can be partially represented as an additional input to parties (such as a CRS), already for a random oracle an

adaptation of the theory is needed, as shown by Bernhard, Fischlin and Warinschi [4] in the adaptive PoK setting (without interactive agreement). Moreover, their treatment still does not allow the relation to depend on the random oracle, missing the opportunity to lay the foundation for zero-knowledge proofs about relations involving random oracle queries. More generally, initially generating an encoding of the relation prevents it from depending on the dynamic state of a setup. Finally, the lack of support for more complex setups hinders the formalization of applications where the relation must only be black-box checkable for either the prover or verifier (e.g., for privacy reasons).

We hence face a situation that the current state of the art on generalizations of PoK does not allow to adequately capture the security goal of a broad range of applications which ought to be instantiations of some generalized proof-of-knowledge notion. For instance, the basic security of password-based client authentication schemes naturally appears to be captured as having to know the password. But since the relation is characterized by a database whose description cannot be given explicitly to the prover and because the prover shall not verify password guesses on his own, none of the above (more generalized) notions apply; therefore, the security is typically described in a property based manner assuming that the password is drawn according to some high-entropy distribution, deviating from the established PoK paradigm [20]. Similarly, in the realm of cloud file storage, the security of schemes where a client aims to convince the server that he knows a specified file (e.g., client-side deduplication), has been formalized using a min-entropy based security definition [12], and it is not clear how this maps to practice. While all these examples arguably follow a generic and dynamic agree-and-prove paradigm with non-trivial setup, their respective tailored security definitions miss this connection. An additional undesirable consequence of the lack of such a generalized formalism is that ad-hoc privacy notions for these applications [19, 11] must be invented instead of simply relying on a well-understood zero-knowledge definition for a generalized PoK.

1.1 Our Contributions

Agree-and-prove. Based on the above considerations, we introduce a new notion called *Agree-and-Prove* that does include all the above (missing) elements. Our notion generalizes proofs (and arguments) of knowledge to dynamic settings where the prover and verifier, based on a setup and their initial state, first have to agree on a statement (agreement phase), of which the prover then convinces the verifier in a second phase (proof phase). It provides clear interpretations for the correctness, soundness, and zero-knowledge properties in the presence of setups and interactive statement derivation, and is therefore suitable as a unifying cryptographic concept behind the above mentioned scenarios. And in addition to the above scenarios, in a recent work [17] (Eurocrypt '21), Vidick and Zhang applied the agree-and-prove notion to the quantum setting (switching the underlying computational model to quantum algorithms) to prove the security of quantum proofs of knowledge with classical verification and (quantum-)setup dependent relations.

We stress that capturing such a general notion for PoK comes with a number of new subtleties to overcome compared to [5, 4] (for a more detailed comparison, we refer to the full version of this work [1]). It indeed seems to be an intricate task to formally capture the relevant probabilistic experiments (for correctness, soundness, and zero-knowledge) because they have to deal with (1) the omnipresent dependency—especially of the relation to be proven—on the state of the setup, (2) the kind of access of the different entities to the setup (including the simulator), (3) the question how the state of the setup has been generated before a proof is executed, (4) that different entities might have different side information regarding the state of the setup, and (5) the cryptographic experiments should have a clear semantics when applied in a bigger context, and as such it should be clear how they compose in larger systems.

Our framework can thus be seen as a unification of all the aforementioned approaches. The agree-and-prove notion is parametrized by an arbitrary setup functionality on which the agreed statement and the associated relations can depend. We formulate both cases of programmable vs. non-programmable setups. Moreover, we define the equivalent of zero-knowledge and consider both, prover and verifier zero-knowledge which is needed in dynamic settings where both parties potentially have information they do not want to reveal. Finally, for the sake of generality, our definitions of zero-knowledge are parametrized using explicit leakage functions, accommodating for protocols that leak limited information (in case no leakage is impossible). Note that the criticality of particular leakage function is application dependent. We conclude the definitional section with a discussion on how the new, stand-alone notion can be understood in the context of composability.

Application to proofs of ownership. We capture proofs of ownership of files, that aim to achieve secure client-side deduplication in a cloud storage system, as a natural instantiation of agree-and-prove. Recall that the connection between client-side deduplication and proofs of ownership stems from the fact that it can be much more efficient to convince (by means of a secure protocol) a server that one has a file than just uploading the file again in full [12], and only uploading the entire file in case the server does not have it.

We point out that compared to previous definitions in this space, including [12, 19], our formalism is not tailor-made for a particular application, but inherited from a higher-level abstraction. Moreover, our formalization does not impose a distribution on files (i.e., following an entropy-based approach to knowledge). Nevertheless, if desired, such notions can be recovered within our framework by assuming a stronger setup where files in the database have intrinsic min-entropy. We point out, however, that assuming a stronger setup can significantly affect the efficiency of admissible protocol (reducing the complexity of the proof phase), up to the extent that extraction becomes trivial but soundness is still satisfied. Such situations are encountered in Sections 3.1 and 4 of this work.

In addition, we demonstrate that our notion is flexible enough to instantiate variations of proof of ownership with different levels of security. In particular, we show that a naive hash-based scheme — whose apparent insecurity in practice

originally served as the driving motivation for the formalization and study of proof of ownership [12] — can be proven both to be either secure or insecure, depending on whether the setup formalizes the hash function as (unrealistic) local random oracle or a (more realistic) global random oracle, respectively. We further show how to retain security in this global random oracle setting by employing a stronger proof phase in which a Merkle-Tree based proposal as in [12] is executed. This serves as a good example to see how different agree phases influence the complexity of the associated proof phase—yet the overall agree-and-prove interface to an application, in this case providing the abstraction of a secure proof of ownership, remains identical.

Privacy-preserving proofs of ownership. We extend proofs of ownership to a privacy-aware setting in Section 3.3. Consider a situation where a set of clients (e.g. employees of the same company) share a secret key under which they apply client-side encryption of the files before uploading them to a server. We present a novel scheme that allows an employee to prove that he knows the plaintext of a ciphertext without having to know the randomness that was used during the encryption (and in particular, without knowing the ciphertext stored by the server). We prove that our protocol does not reveal more information to both client or server than what is generally necessary for the task of client-side deduplication and is still communication efficient (in that the ciphertext stored by the server does not have to be communicated to the client). Analogously to above, in comparison with previous approaches to privacy in this context [19, 11], we formulate a cryptographic definition of privacy for proofs of ownership that is justified by a generalized zero-knowledge definition for agree-and-prove schemes.

Overall, our construction is designed as the privacy-preserving analogue of the above Merkle-Tree based solution (using collision-resistant hash functions). The thereby added privacy layer enables a modular analysis with a clear separation into the two tasks of proving ownership and protecting the privacy, which we believe is a desirable simplification compared to more "interleaved" approaches such as [19, 11]. Furthermore, our construction is secure under standard cryptographic assumptions and compared to [11] does not use random oracles.

Application to client authentication. In Section 4, a second application of agree-and-prove is presented. First, it is shown how password-based authentication naturally fits as an instantiation of the notion. Then, it is discussed how advanced security properties arising in the context of password-based authentication, such as protection from precomputed rainbow-tables, can be taken into account. Finally, we present a direct instantiation of Agree-and-Prove that captures two-factor authentication. The fact that the knowledge-relation can depend on the setup is thereby leveraged to demonstrate that the agree-and-prove notion can not only (as expected) formalize proofs and arguments of knowledge, but is in fact the cryptographic tool to capture in a similar spirit *proofs of possession* or *proofs of ability* such as the possession and use of a hardware-token.

1.2 Extended Overview of Results

The focus of the first part of this work is defining the agree-and-prove (AaP) framework. The model we present in Section 2, in a nutshell, consists of three main components: first, *the scenario* that formalizes the setup and the setup dependent relations which describe the set of statements to be proven dependent on the setup. Second, the interactive protocols for prover and verifier, and third, the formal experiments for correctness, soundness, and zero-knowledge.

In Section 3, we dive into the application of proofs-of-ownership of files that aim to achieve secure client-side deduplication in a cloud storage system. In a nutshell, these schemes consist of a client convincing a server that it has a file F (already stored in the server's database), but without uploading the entire file. Roughly speaking, we model this as a scenario where the setup consists of a (server-)database, the statements are file identifiers contained in the database, and the proof relation simply consists of all pairs (x, w) in the database where w is the file with identifier x. The main security concern is thereby that the client cannot falsely convince the server, corresponding to soundness.

A secure, i.e. sound, protocol for this scenario can be derived on various assumptions including the (local) random oracle model or based on collision resistant hashing and Merkle-Trees. For the former, we observe that working with idealized hash functions (i.e., random oracles) is already sufficient to conclude knowledge, but the assumption of a local random oracle is unrealistic in this setting. The latter approach improves on this: the file F is divided into a sequence of blocks, considered as leaves in a binary tree, and intermediate nodes are the hashes of its two children. If the prover can repeatedly provide siblings paths from a randomly selected leaf to the root (which is also known to the server), then standard hardness-amplification results imply knowledge of F. The only missing piece is privacy, where our new protocol in Section 3.3 comes into play. Briefly, here we have the situation that the setup is actually a database containing user-encrypted files. The privacy level is captured by explicitly specifying the leakage that client and server must admit beyond the validity of the statement. For our protocol, the incurred leakage can be summarized as follows: per protocol run, a cheating client can at most learn whether for a (chosen) identifier id and bitstring v, there is a file whose Merkle-Tree root equals v. On the other hand, a cheating server can learn, beyond the validity of the statement, at most which entry in its database is the subject of the interactive proof and, if such an entry exists, the length of the plaintext.

Our scheme $\mathcal{S}_{\mathsf{priv}}$ specifies the protocols for the client and the server and keeps the basic structure of a Merkle-Tree solution. However, instead of a basic agreement on the file identifier and its Merkle-Tree root, a more complicated plaintext-comparison on encrypted data must take place. In order to ensure that the comparison does not leak more than needed, we employ standard NIZKs and specific OR-proofs that allow an honest server to conceal certain information if misbehavior is detected. Once agreement is reached, the proof phase performs the random checks on encrypted data, verifying the validity again using specific NIZK proofs. This is detailed in Section 3.3 of the main body of this work.

Theorem (informal). *The Agree-and-Prove scheme $\mathcal{S}_{\mathsf{priv}}$ used between client and server when performing a proof-of-ownership of files over an encrypted database satisfies the following three security properties:*

1. *Soundness: Except with negligible error, the client cannot convince the server that it possesses a file corresponding to an encrypted entry in the server's database unless he knows the file.*
2. *Server privacy: In one run of the protocol, the server does not reveal more (than one bit of) information to the client than whether a chosen pair (id, v) is a valid combination of identifier and Merkle-root of a file that is encrypted in the server database.*
3. *Client privacy: in one run of the protocol, the client does not reveal more information to the server than the file identifier and, if the file exists in the database, at most the length of the file plus one additional bit of information, namely whether the client holds the valid plaintext of the database entry.*

The appealing property of this protocol is that it has a much simpler structure than previous constructions, it admits a modular security proof, and has all properties to qualify as a privacy-preserving proof of ownership without relying on entropy assumption on the file distribution.

Finally, Section 4 shows how password-based authentication can be cast as an Agree-and-Prove scheme. While this is rather straightforward, Section 4 shows how to capture hardware tokens in our theoretical framework: a hardware token can be modeled as a setup $\mathcal{F}_{\text{2-FA}}$ that internally stores a key pair for the user, but only exports a decryption capability to the user, but not the private key. Therefore, 2FA cannot be modeled as a proof of knowledge of the secret key. Using the AaP we can directly formalize the idea: the proof relation is simply based on a setup $\mathcal{F}_{\text{2-FA}}$ and formalizes that the client to be authenticated has the ability to perform decryption operations (w.r.t. to a secret key whose public key is known). Of course, the functionality also contains a database relating users to their passwords to precisely model 2FA. In Section 4, we show that standard 2FA protocols consisting of a password-based authentication protocol and a challenge-response mechanism based on a hardware token can be captured as an agree-and-prove scheme $\mathcal{S}_{\text{2FA}}$:

Theorem (informal). *The Agree-and-Prove scheme $\mathcal{S}_{\text{2FA}}$ that authenticates a client to the server using a password and a hardware token (formalized as a setup $\mathcal{F}_{\text{2-FA}}$ as described above) satisfies the following security property:*

– *Soundness: Except with negligible probability, a successful run of the authentication protocol implies that the client (associated with a known public key) knows the correct password and has access to the correct decryption capability.*

This concludes the extended overview of our work and we begin with the main body of the paper focusing first on the formal foundations of the general model.

2 Agree-and-Prove: Definition

In this section, we introduce our notion of an agree-and-prove scheme. Such a scheme is intended to capture a setting where two parties, the prover and the verifier, dynamically want to agree on a statement of whose validity the prover then wants to convince the verifier. The statement is not fixed beforehand and can in particular depend on the environment in which they execute the protocol as well as the parties' prior knowledge.

2.1 The Scenario

Analogous to a proof-of-knowledge scheme, an agree-and-prove scheme is only well defined with respect to a goal it should achieve. While in proof of knowledge such a goal is simply given by an NP-relation, it is now generalized for agree-and-prove schemes.

First, we consider in our notion a *setup* that models some assumptions on the world in which we execute the protocol. Such a setup can simply consist of a CRS or a random oracle, but can also model further assumption such as a file database assigning files to certain identifiers in the case of a proof of ownership. Second, characterizing which statement the parties may agree on—in dependence of the setup and the parties' prior knowledge—is an integral part of specifying the goal of an agree-and-prove scheme. This is characterized by an *agreement condition*. Third, the *proof relation* characterizes what it means to satisfy the statement they agreed on (for which we simply use the common term proof). This relation generalizes the NP-relation of the proof-of-knowledge formalization, as it can capture notions of knowledge as well as more general properties about the relation between the statement and the setup.

We formally define this intuition below: An agree-and-prove scenario captures what is the assumed setting in which the protocol is executed and specifies the goal of the scheme.

Definition 1 (Agree-and-Prove Scenario). *An agree-and-prove scenario Ψ is a triple $\Psi := (\mathcal{F}, \mathcal{R}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}, C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)})$, consisting of the following components:*

- *A* setup functionality $\mathcal{F}$*, which is a PPT ITM that consists of an initialization procedure* init *and then provides an oracle $\mathcal{O}_{\mathcal{F}}(i, \mathsf{q}, arg)$, where $i \in \{\mathsf{I}, \mathsf{P}, \mathsf{V}\}$ denotes a role,* q *denotes a keyword, and arg denotes the argument for this query. For technical reasons, the setup functionality keeps track of all queries (including the answer) by the prover, exposing them as an oracle $\mathcal{O}_{\mathcal{F}}(\mathtt{QUERIES})$.*
- *An* agreement condition $C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}$*, which is a PPT oracle machine taking a unary encoding of the security parameter κ, two auxiliary inputs and a statement as inputs, and producing a decision bit as output.*
- *A* proof relation $\mathcal{R}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}$*, which is a PPT oracle machine taking a unary encoding of the security parameter κ, a statement x, and a witness w as inputs, and outputting a decision bit.*

Generic Setup Functionality $\mathcal{F}$

- init: Setup-Functionality initialization procedure
- $\mathcal{O}_{\mathcal{F}}(i, \mathsf{q}, arg)$: Interaction of setup with the participants, where
  - $i \in \{\mathsf{I}, \mathsf{P}, \mathsf{V}\}$ denotes a role.
  - q is a keyword.
  - arg is the argument for this query.
- $\mathcal{O}_{\mathcal{F}}(\texttt{QUERIES})$: Recorded queries of role P. Upon invocation, the oracle returns a list of $(\mathsf{q}, arg, reply)$ triples corresponding to all the queries made by role P so far.

Fig. 1: A generic setup functionality $\mathcal{F}$, which consists of an initialization procedure init and then provides an oracle $\mathcal{O}_{\mathcal{F}}(i, \mathsf{q}, arg)$, where $i \in \{\mathsf{I}, \mathsf{P}, \mathsf{V}\}$ denotes a role, q denotes a keyword, and arg denotes the argument for this query. Furthermore, $\mathcal{F}$ keeps track of all the prover's queries.

Observe that the setup functionality, as depicted in Figure 1, contains three oracles that operate on shared state and randomness. Both the prover and the verifier have their own oracles $\mathcal{O}_{\mathcal{F}}(\mathsf{P}, \cdot, \cdot)$ and $\mathcal{O}_{\mathcal{F}}(\mathsf{V}, \cdot, \cdot)$, respectively. This allows us, for instance, to express that if the setup contains a login database, then only the verifier has access to the passwords. In addition, there is also the third oracle $\mathcal{O}_{\mathcal{F}}(\mathsf{I}, \cdot, \cdot)$ capturing the information and prior influence that third parties can have about the setup. For example, the setup can either be a shared private key, or it can be a public CRS, where only in the latter case the oracle $\mathcal{O}_{\mathcal{F}}(\mathsf{I}, \cdot, \cdot)$ can access it. Some leakage about the information obtained through this oracle might also be passed to the parties as prior knowledge, capturing that for instance a dishonest prover might obtain hashes from other parties without knowing the respective queries.

2.2 The Protocols

For a given agree-and-prove scenario we can now define the notion of a corresponding agree-and-prove scheme. Such a scheme consists of two pairs of protocols for the prover and verifier, (P_1, V_1) and (P_2, V_2), where the former pair agrees on the statement for which the latter one then will execute the necessary proof. More concretely, the prover and verifier P_1 and V_1, respectively, output the statement they agreed on at the end of the first phase, or chooses to abort the protocol by outputting $\perp$ in case they could not agree. If they do agree on a statement, then at the end of the second phase the prover and verifier P_2 and V_2, respectively, output whether the proof has been successful or not.

Definition 2 (Agree-and-Prove Scheme). *An* agree-and-prove scheme *is a quadruple $\mathcal{S} := (P_1, P_2, V_1, V_2)$, consisting of the following four interactive PPT oracle machines:*

- *A (honest) first-phase prover* $P_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}$ *taking a unary encoding of the security parameter* κ *and an auxiliary input* aux_p *as inputs. It produces a statement* x_p *or* $\perp$ *as output, as well as a state* st_p*.*
- *A (honest) first-phase verifier* $V_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)}$ *taking a unary encoding of the security parameter* κ *and an auxiliary input* aux_v *as inputs. It produces a statement* x_v *or* $\perp$ *as output, as well as a state* st_v*.*
- *A (honest) second-phase prover* $P_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}$ *taking a state* st_p *as input, as well as a unary encoding of the security parameter* κ*, and producing as output a bit that indicates whether the proof has been accepted.*
- *A (honest) second-phase verifier* $V_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)}$ *taking a state* st_v *as input, as well as a unary encoding of the security parameter* κ*, and producing as output a bit that indicates whether it accepts or rejects.*

Observe that both the prover and the verifier can keep state between the two phases. Furthermore, note that both the prover and the verifier get an auxiliary input aux_p and aux_v, respectively, as input which models the parties' prior knowledge about the world, and where aux_p typically contains a witness (amongst other inputs). Finally, note that in a slight abuse of notation, we treat an empty output at the end of the agreement phase as $x = \perp$.

*Remark 1 (*On variations of the computational model*).* We formulate the above algorithms as interactive and PPT for the sake of concreteness and since our presented applications live in this world. However, as for the traditional notions, various computational models and properties can be considered for agree-and-prove such as allowing unbounded provers in Definition 2 or considering computational instead of information-theoretic soundness in Section 2.3 or different runtime requirements for extractors. Also, intermediate computational classes such as unbounded provers with limited calls to the setup (e.g., random oracle calls) would be possible to consider. On another dimension, one can restrict the number of messages exchanged or number of queries made in the proof phase. An obvious example would be to restrict the prover to send only a single message in the second phase which would overall establish (a generalized notion of) non-interactive proofs.

We move on to define the execution of an agree-and-prove scheme:

Definition 3. *Let* aux_p *and* aux_v *denote two bit-strings, let* $\mathcal{F}$ *denote a setup functionality, and let* $\mathcal{S} := (P_1, P_2, V_1, V_2)$ *denote an agree-and-prove scheme. Then,*

$$((x_p, st_p); (x_v, st_v); T) \leftarrow \left\langle P_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}, V_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)} \right\rangle \left((1^\kappa, aux_p); (1^\kappa, aux_v)\right)$$

denotes the execution of the agreement phase between the honest first phase prover P_1 *and the honest first phase verifier* V_1*. Note that we use the notation* $(a; b; T) \leftarrow \langle A, B \rangle (x, y)$ *to denote the interactive protocol execution of interactive algorithms* A *and* B *invoked on the inputs* x *and* y*, respectively, and where a*

and b are the resulting outputs of A and B, respectively, and where T denotes the communication transcript. Moreover,

$$(v; v'; \cdot) \leftarrow \left\langle P_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}, V_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)} \right\rangle \big((1^\kappa, st_p); (1^\kappa, st_v)\big)$$

denotes the execution of the proof phase between the honest second phase prover P_2 and verifier V_2, with v and v' being the decision bit of the prover and verifier, respectively.

2.3 The Basic Security Notion

In this section, we define the agree-and-prove security notion that generalizes the traditional security requirements expected from proofs of knowledge.

Prior knowledge and context. Recall from the previous section that both parties take an auxiliary input. While the setup models the world which we assume the protocol to be executed in, those auxiliary inputs model the parties' prior knowledge (a similar concept was used in [9, Section 4.7.5] on identification schemes). In the security experiment, those inputs will be generated by a respective algorithm.

Definition 4 (Input Generation Algorithm). *An input generation algorithm $I^{\mathcal{O}_{\mathcal{F}}(\mathsf{I},\cdot,\cdot)}$ for an agree-and-prove scenario $\Psi := (\mathcal{F}, \mathcal{R}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}, C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)})$ is a PPT oracle machine taking a unary encoding of the security parameter κ as input and producing a pair of bit-strings (aux_p, aux_v), specifying the auxiliary inputs for the prover and verifier respectively, as output.*

Note that this algorithm gets oracle access to the setup functionality via its own oracle $\mathcal{O}_{\mathcal{F}}(\mathsf{I}, \cdot, \cdot)$. This allows us to capture the prior knowledge and context in which the protocol is executed as part of the setup functionality itself and therefore as part of the agree-and-prove scenario. The input generation algorithm is then universally quantified over in the security definition, making a clean separation between the part we do make assumptions about (the functionality) and the part which we do not make assumption about, such as the prior knowledge or context as derived from the functionality (cf. also Section 2.5).

Programmability and non-programmability. There are many cases in which one would like to formalize that an extractor can program the setup (e.g., a backdoor in a CRS model). He should, however, be only allowed to do so in a "correct", i.e., undetectable, manner, as otherwise he might for instance force the prover and verifier to disagree on the statement and abort, thereby making the extraction game trivial. To this aim, we introduce the notion of a setup generation algorithm to formally capture (valid) programmability.

Definition 5 (Setup Generation Algorithm). *A setup generation algorithm* SGen *is a PPT taking a unary encoding of the security parameter κ as input. It outputs (the description of) a setup functionality $\mathcal{F}'$ and a trapdoor td as output.*

We say that the setup generation algorithm SGen *is* admissible *with respect to time* $T(\cdot)$ *for an agree-and-prove scenario* $\Psi := (\mathcal{F}, \mathcal{R}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}, C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)})$*, if for every oracle machine* $\mathcal{A}$ *with running time bounded by* $T(\cdot)$ *the following advantage is negligible in* κ*:*

$$\mathsf{Adv}^{\mathsf{AP\text{-}Setup}}_{\Psi,\mathsf{SGen},\mathcal{A}} := \Pr^{\mathcal{F}.\mathsf{init}(1^\kappa)}[\mathcal{A}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}(1^\kappa) = 1] - \Pr^{(\mathcal{F}',td)\leftarrow\mathsf{SGen}(1^\kappa);\ \mathcal{F}'.\mathsf{init}(1^\kappa)}[\mathcal{A}^{\mathcal{O}_{\mathcal{F}'}(\cdot,\cdot,\cdot)}(1^\kappa) = 1].$$

We formulate the security games that potentially require programmability in proofs using this generated setup instead of the real one. The extractor then gets a trapdoor td (e.g. for the generated CRS) that he can use for the extraction during the prove phase (or in the zero-knowledge case to simulate proofs). Other than that, the generated setup is directly used in the security game.

On the other hand a non-programmable setup corresponds to restricting setup generation algorithms that do not produce any leakage, which is, in accordance with the above definition, essentially equivalent to just taking the real setup functionality $\mathcal{F}$. Finally, one can also easily model a mixture of programmable and non-programmable setups by considering $\mathcal{F} := (\mathcal{F}_1, \ldots, \mathcal{F}_k)$ as one setup functionality with $\mathsf{SGen}(1^\kappa) := (\mathsf{SGen}_1(1^\kappa), \ldots, \mathsf{SGen}_k(1^\kappa))$ being the corresponding setup-generation, where for each non-programmable setup $\mathcal{F}_i$, SGen_i is required to produce no leakage.

The Security Definition. Based on the notion of an input generation algorithm I and a setup generation algorithm SGen, we now define the security game. Before giving the definition, we explain and motivate the security conditions appearing in Figure 2 in the following paragraphs.

Correctness. First, the parties must agree on a common statement and on the outcome of the proof at the end of the agreement and proof phases, respectively. Second, they need to agree on a legitimate statement, with respect to the parties' prior knowledge aux_p and aux_v, respectively, as well as the setup functionality $\mathcal{F}$. The legitimacy is indicated by evaluating the validity condition $C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}$ of the agree-and-prove scenario (see Section 2.1), with three possible outcomes: The output 0 indicates a violation of correctness (e.g., to exclude trivializing the problem by always agreeing on a trivial statement). The output 1 indicates a valid statement for which the prover must be able to convince the verifier. Finally, the output $*$ indicates a statement on which the parties may agree, for which the proof phase may however either accept or reject. Therefore, the ternary output captures in a fine-grained manner under which circumstances the agreement on a statement must imply the success of the following prove-phase and when this is not necessarily guaranteed, which happens for example in situations where the client cannot verify the relation and we still want to keep the agree-phase simple. Furthermore, note that we do not require the honest prover to explicitly output a witness for the proof relation—the fact that he in principle knows such a witness is covered by the soundness condition.

Experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Corr},\Psi}_{\mathcal{S},I}$

Require: 1^κ, where $\kappa \in \mathbb{N}$
  $\mathsf{flag}_{\mathrm{Corr}} \leftarrow 1$
  Execute $\mathcal{F}.\mathsf{init}(1^\kappa)$
  $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
  $((x_p, st_p); (x_v, st_v); \cdot) \leftarrow \langle P_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}, V_1^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, aux_p); (1^\kappa, aux_v)\big)$
  $c \leftarrow C^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x_v)$
  if $x_p \neq x_v$ **or** $c = 0$ **then**
    $\mathsf{flag}_{\mathrm{Corr}} \leftarrow 0$; **return**
  else if $x_v \neq \bot$ **then**
    $(v; v'; \cdot) \leftarrow \langle P_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}, V_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, st_p); (1^\kappa, st_v)\big)$
    $\mathsf{flag}_{\mathrm{Corr}} \leftarrow (v' = v) \wedge (c = * \vee v' = \mathsf{accept})$

Experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Ext},\Psi}_{\mathcal{S},\mathsf{SGen},I,E,\hat{P}}$

Require: $(1^\kappa, p)$, where $\kappa \in \mathbb{N}$ and $p\colon \mathbb{N} \to [0,1]$
  $(\mathcal{F}', td) \leftarrow \mathsf{SGen}(1^\kappa)$
  Execute $\mathcal{F}'.\mathsf{init}(1^\kappa)$
  $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
  $((\cdot, st_p); (x, st_v); T) \leftarrow \langle \hat{P}_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, V_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, aux_p); (1^\kappa, aux_v)\big)$
  if $x = \bot$ **then**
    $\mathsf{flag}_{\mathrm{Ext}} \leftarrow 1$; **return**
  Define $\mathsf{succ} :=$
    $\Pr\Big[v' = \mathsf{accept} : (\cdot; v'; \cdot) \leftarrow \langle \hat{P}_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, V_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, st_p); (1^\kappa, st_v)\big)\Big]$
  if $\mathsf{succ} \leq p(\kappa)$ **then**
    $\mathsf{flag}_{\mathrm{Ext}} \leftarrow 1$; **return**
  $w \leftarrow E^{\mathcal{O}_E}(1^\kappa, x, aux_p, st_p, T, td)$
  $\mathsf{flag}_{\mathrm{Ext}} \leftarrow (\mathcal{R}^{\mathcal{O}_{\mathcal{F}'}(\cdot,\cdot,\cdot)}(1^\kappa, x, w) = 1)$

---

where:
  $\mathcal{O}_E := \mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot), \mathcal{O}_{\mathcal{F}'}(\mathtt{QUERIES}), \mathcal{R}^{\mathcal{O}_{\mathcal{F}'}(\cdot,\cdot,\cdot)}(1^\kappa,\cdot,\cdot), \mathcal{O}_{\mathsf{BBR}}(\hat{P}_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}(st_p))$

Fig. 2: Security experiments for an Agree-and-Prove scheme. Top: The correctness experiment. Bottom: The extraction experiment to formalize soundness (the case where an honest verifier $V = (V_1, V_2)$ interacts with a dishonest prover $\hat{P} = (\hat{P}_1, \hat{P}_2)$).

Soundness. The extraction experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Ext},\Psi}_{\mathcal{S},\mathsf{SGen},I,E,\hat{P}}$ formalizes that every (potentially dishonest) prover that can convince the verifier with probability at least $p(\kappa)$ of a statement x must know a witness w that satisfies the proof relation $\mathcal{R}^{\mathcal{O}_{\mathcal{F}}(\cdot,\cdot,\cdot)}(1^\kappa, x, w)$. Analogous to a proof of knowledge, we phrase this via the existence of an extractor. More precisely, this extraction property refers to the proof phase of the protocol, formalizing that the above guarantee holds for every valid statement x which the prover manages to agree on with the verifier. To reflect this in the security game, the agreement phase is executed exactly once, and cannot be rewound, thereby fixing the statement x, the prover's and verifier's state st_p and st_v respectively, and the state of the setup functionality. (See the two final remarks on the experiment below for a more formal notion of "state"). It is important to note that our definition simultaneously captures what is typically referred to as validity and soundness[4], as we let the extractor run w.r.t. any derived statement. That is, if V_1 accepts an invalid statement (without witness), there exists trivially no extractor that provokes $\mathsf{flag}_{\mathrm{Ext}} = 1$.

Back to extraction, with respect to this overall state after phase one, the extractor has to provide a witness w (within a reasonable time bound along the lines of Goldreich [9, Definition 4.7.1]). To achieve extraction, the extractor gets the statement x, the prover's state st_p, and the communication transcript T of the agreement phase. Furthermore, he gets black-box rewinding access to the dishonest prover (communication) strategy $\hat{P}_2$, access to the prover's oracle of the setup functionality $\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)$, and access to the list of setup queries made by the prover, which is provided by the oracle $\mathcal{O}_{\mathcal{F}}(\texttt{QUERIES})$. In contrast to a traditional proof of knowledge where the relation is deterministic and publicly known, we also provide an oracle to the extractor with black-box access to the predicate defining the proof relation (which in general could depend on the randomness of the setup functionality). We refer the reader to the discussion after Definition 6 for the rationale behind these choices in comparison with traditional proof-of-knowledge systems. Two formal considerations about the extraction experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Ext},\Psi}_{\mathcal{S},\mathsf{SGen},I,E,\hat{P}}$ are in order:

- For sake of concreteness, we understand the black-box rewinding oracle $\mathcal{O}_{\mathsf{BBR}}(\hat{P}_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}(st_p))$ as a stateful *message-specification function* along the lines of Goldreich [9, Definition 4.7.1]: more formally, when invoked with a random tape r, the oracle creates the machine $\hat{P}_2(st_p)$ in its initial configuration with random tape r. It then provides black-box access to the communication behavior by accepting incoming messages, performing the state transitions of $\hat{P}_2$ until it outputs the next message to be sent. Note that during this computation, oracle calls to $\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)$ might be made (which can neither be intercepted nor undone unless the setup functionality would allow this form of resetting).

[4] In traditional proof-of-knowledge games, the extraction game is called validity and only valid statements $x \in L$ for some language L, are considered, whereas soundness requires an extra condition to capture security for the case that $x \notin L$.

– The formal expression[5]

$$\mathsf{succ} \coloneqq \Pr\Big[v' = \mathsf{accept} : (\cdot; v'; \cdot) \leftarrow \big\langle \hat{P}_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot)}, V_2^{\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)} \big\rangle\big((1^\kappa, st_p); (1^\kappa, st_v)\big)\Big]$$

is associated to the following probability space: first, the start configuration of both machines of prover and verifier is the initial configuration with the specified input tape. The start configuration of machine $\mathcal{F}$ is the configuration at the end of the first phase (i.e., a snapshot). The probability space is then formed over the random coins of prover and verifier, and over the coins, i.e., positions on the random tape, of $\mathcal{F}$, that have not been read up to and until the above start configuration of machine $\mathcal{F}$.

We state the definition of the security requirements of an agree-and-prove scheme.

Definition 6 (Agree-and-Prove Security). *Let $p\colon \mathbb{N} \to [0,1]$. An agree-and-prove scheme $\mathcal{S}$, for an agree-and-prove scenario Ψ, is secure up to soundness error p, if the following conditions hold, where the experiments are defined in Figure 2.*

– Correctness*: The experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Corr},\Psi}_{\mathcal{S},I}$ returns with $\mathsf{flag}_{\mathrm{Corr}} = 1$ with probability 1 for all input-generation algorithms I and all κ.*
– Soundness*: There exists an extractor algorithm E and an admissible setup generation algorithm SGen (with respect to time $T(\cdot)$ which is at least the running time of E), such that for all dishonest provers $\hat{P} = (\hat{P}_1, \hat{P}_2)$ and input generation algorithms I, the experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Ext},\Psi}_{\mathcal{S},\mathsf{SGen},I,E,\hat{P}}$ on input $(1^\kappa, p)$ returns with $\mathsf{flag}_{\mathrm{Ext}} = 1$ except with negligible probability. Furthermore, for some $c > 0$, the expected number of steps of extractor E within the experiment $\mathsf{Exp}^{\mathsf{AP\text{-}Ext},\Psi}_{\mathcal{S},\mathsf{SGen},I,E,\hat{P}}$ on input $(1^\kappa, p)$ is required to be upper bounded by $\kappa^c/(\mathsf{succ} - p(\kappa))$ (where the experiment ensures that $\mathsf{succ} > p(\cdot)$).*

We next discuss some of the motivation and rationale behind the definition.

Discussion of selected elements. We first observe that providing the prover with the transcript of the agreement phase implies that in the proof phase we do not necessarily have a full-fledged proof or argument of knowledge of a witness as it, or parts of it, could already be contained in the agreement phase, thereby allowing for a more efficient proof phase.

Moreover, providing the extractor with the prover's input, state, and the setup queries from the first round also entails a couple of implications: we formalize naturally that it is sufficient for the prover to *know* a witness in order to pass the test—in contrast to more traditional definitions of proofs of knowledge requiring that the *communication needs to prove that he knows* one.

[5] We would like to stress that the formal evaluation of the expression has no side-effects on the state of any of the involved entities.

For instance, consider a shared URF $U(\cdot)$ between the prover and the verifier as a setup. If the statement is that the prover knows the pre-image x of y under some one-way permutation f, i.e, x such that $f(x) = y$ for x, y known to a verifier, then we would consider sending the correct evaluation under $U(x)$ as convincing, as a prover cannot guess $U(x)$ without querying it except with negligible probability. On the other hand, x cannot be extracted from the communication transcript $U(x)$. We consciously opted for this relaxed definition of knowledge to allow for broader applicability of the concept and because we believe it to capture the essence of a more general understanding of knowledge. For example, in the case of proof-of-ownership of files it is crucial that the communication complexity can be significantly smaller than the file.

2.4 Zero Knowledge

Analogous to a proof of knowledge, we can also require the agree-and-prove scheme to be zero knowledge. That is, whatever a (potentially dishonest) verifier can compute after interacting with the honest prover can also be computed by an appropriate simulator. Since we consider an interactive agreement phase where both parties get private information and a different view on the setup functionality, it however also makes sense to consider prover zero-knowledge. That is, we can phrase that both a verifier, as well as a prover should not learn anything about the other party's input nor about the other party's view on the setup functionality.

While a zero knowledge agree-and-prove protocol certainly represents the optimal case it is often already desirable to limit and explicitly quantify the leakage. To this end, we introduce the notion of a leakage oracle that the simulator is allowed to invoke. Furthermore, in the classical ZK definition, it is assumed that the verifier is always allowed to learn the statement and whether the prover has a valid witness. Since in our agree-and-prove notion the statement and the witness are not a priori fixed, this also has to be modeled as an explicit leakage. The classical zero-knowledge definition is then obtained by considering a leakage oracle that only reveals this information.

Definition 7. *A* leakage oracle $\mathcal{L}$ *for a setup functionality* $\mathcal{F}$ *is an oracle PPT ITM that consists of an initialization procedure* init *and an oracle denoted by* $\mathcal{L}^{\mathcal{O}_{\mathcal{F}}(\mathsf{P},\cdot,\cdot),\mathcal{O}_{\mathcal{F}}(\mathsf{V},\cdot,\cdot)}(1^\kappa, aux, query)$*, allowing the simulator to ask certain queries query which are evaluated on the other party's input aux and view on the setup.*

We now proceed to define the classical property that the scheme is zero-knowledge, up to some explicit leakage, with respect to the dishonest verifier. Our definition follows the spirit of the standard (standalone) simulation paradigm where two different settings are compared and should be indistinguishable: one is the real protocol execution, and the other one is the execution where the actions of the dishonest party are simulated by a simulator (having access to the leakage oracle). The distinguishing metric is formalized by a distinguisher D that is given the output of the dishonest verifier, the protocol outputs of

Experiment $\mathsf{Exp}^{\mathsf{AP\text{-}ZK\text{-}V},\Psi,\mathcal{S},\mathcal{L}_P}_{\hat{V},I,S,\mathsf{SGen},D}$

Require: 1^κ, where $\kappa \in \mathbb{N}$
  $b \leftarrow \{0,1\}$
  if $b = 0$ **then**
    $\mathcal{F}' \leftarrow \mathcal{F}$
    Execute $\mathcal{F}'.\mathsf{init}(1^\kappa)$
    $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
    $((x, st_p); out_1; \cdot) \leftarrow \langle P_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, \hat{V}_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, aux_p); (1^\kappa, aux_v)\big)$
    $(v; out_2; \cdot) \leftarrow \langle P_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, \hat{V}_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, st_p); (1^\kappa, out_1)\big)$
  else
    $(\mathcal{F}', td) \leftarrow \mathsf{SGen}(1^\kappa)$
    Execute $\mathcal{F}'.\mathsf{init}(1^\kappa)$ and $\mathcal{L}_P.\mathsf{init}(1^\kappa)$
    $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
    $(x, v; out_1, out_2) \leftarrow S^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot),\mathcal{L}_P^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot),\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}(1^\kappa, aux_p,\cdot)}(1^\kappa, aux_v, td)$
  $b' \leftarrow D^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x, v, out_1, out_2)$
  $\mathsf{flag}_{\mathrm{Guessed}} \coloneqq b = b'$

Experiment $\mathsf{Exp}^{\mathsf{AP\text{-}ZK\text{-}P},\Psi,\mathcal{S},\mathcal{L}_V}_{\hat{P},I,S,\mathsf{SGen},D}$

Require: 1^κ, where $\kappa \in \mathbb{N}$
  $b \leftarrow \{0,1\}$
  if $b = 0$ **then**
    $\mathcal{F}' \leftarrow \mathcal{F}$
    Execute $\mathcal{F}'.\mathsf{init}(1^\kappa)$
    $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
    $(out_1; (x, st_v); \cdot) \leftarrow \langle \hat{P}_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, V_1^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, aux_p); (1^\kappa, aux_v)\big)$
    if $x \neq \perp$ **then**
      $(out_2; v; \cdot) \leftarrow \langle \hat{P}_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot)}, V_2^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}\rangle\big((1^\kappa, out_1); (1^\kappa, st_v)\big)$
    else
      $out_2, v \leftarrow \perp$
  else
    $(\mathcal{F}', td) \leftarrow \mathsf{SGen}(1^\kappa)$
    Execute $\mathcal{F}'.\mathsf{init}(1^\kappa)$ and $\mathcal{L}_V.\mathsf{init}(1^\kappa)$
    $(aux_p, aux_v) \leftarrow I^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa)$
    $(out_1, out_2; x, v) \leftarrow S^{\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot),\mathcal{L}_V^{\mathcal{O}_{\mathcal{F}'}(\mathsf{P},\cdot,\cdot),\mathcal{O}_{\mathcal{F}'}(\mathsf{V},\cdot,\cdot)}(1^\kappa, aux_v,\cdot)}(1^\kappa, aux_p, td)$
  $b' \leftarrow D^{\mathcal{O}_{\mathcal{F}'}(\mathsf{I},\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x, v, out_1, out_2)$
  $\mathsf{flag}_{\mathrm{Guessed}} \coloneqq b = b'$

Fig. 3: The zero-knowledge security experiments for an AaP scheme. The first experiment phrases verifier zero-knowledge, whereas the second one phrases prover zero-knowledge. The distinguisher is given the auxiliary input, both parties' outputs of the interaction.

the honest party, and access to the context information, i.e., it is given the auxiliary input and access to interface I of the setup.[6] Note that the outputs of the dishonest verifier in this case (denoted out_1, out_2 in the security game) can contain anything the malicious strategy decides to output[7] (in particular, the entire transcript and information about the setup).

In summary, the definition captures that if the AaP scheme is run as a sub-system in a context specified by the input-generation algorithm, then the resulting trace it leaves by means of transcript and output of the honest party does not leak more than what is specified by the ideal leakage oracle and therefore is in line with the stand-alone simulation paradigm used in traditional zero-knowledge.

Definition 8. *Let $\mathcal{L}_P$ denote a leakage oracle. An agree-and-prove scheme $\mathcal{S}$, for an agree-and-prove scenario Ψ, is* verifier zero-knowledge up to leakage $\mathcal{L}_P$ *if for all dishonest verifiers $\hat{V} = (\hat{V}_1, \hat{V}_2)$ and all input generation algorithms I, there exists an efficient simulator S and an admissible setup generation algorithm SGen such that for all efficient distinguishers D it holds that the experiment $\mathsf{Exp}^{\mathsf{AP\text{-}ZK\text{-}V},\Psi,\mathcal{S},\mathcal{L}_P}_{\hat{V},I,S,\mathsf{SGen},D}$ on input 1^κ returns with $\mathsf{flag}_{\mathrm{Guessed}} = 1$ with probability at most negligibly larger than $\frac{1}{2}$. The experiment is defined in Figure 3.*

Note that in the experiment depicted in Figure 3 we assumed that a dishonest verifier $\hat{V}_1$ is not restricted to output something of the form (x, st_v) at the end of the agreement phase, but can produce an arbitrary output instead. Moreover, observe that a dishonest verifier is not forced to abort, but in principle can always try to execute the proof phase with the honest prover.

We also define the symmetrical property that the scheme is zero-knowledge with respect to the prover. When defining this notion, care has to be taken that observing the honest verifier aborting after the agreement phase does not leak information either. Our definition reflects this in the sense that the honest verifier refuses to execute V_2 in case V_1 ended with an abort, i.e., at most signaling the abort to anyone. Again, an AaP protocol run must be indistinguishable from a simulated run where the simulator has access to a specific leakage oracle, and must simulate the protocol interaction with the malicious prover.

Definition 9. *The scheme is said to be* prover zero-knowledge up to leakage $\mathcal{L}$, *if the same property as in Definition 8 holds for all dishonest provers $\hat{P} = (\hat{P}_1, \hat{P}_2)$ in the experiment $\mathsf{Exp}^{\mathsf{AP\text{-}ZK\text{-}P},\Psi,\mathcal{S},\mathcal{L}_V}_{\hat{P},I,S,\mathsf{SGen},D}$, which is defined in Figure 3.*

[6] The motivation what information to give to the distinguisher will become apparent in Section 2.5 when we discuss compositional aspects of the notion. We point out that the interface of the honest party to the setup, in this case P, is never exposed to any attacker or the distinguisher because the honest party is running the protocol and only the actual protocol outputs are visible to an "environment".

[7] We point out that both outputs are needed: for example if the protocol aborts after the first phase, we require the agree-phase not leak anything beyond what is specified by the leakage oracle.

2.5 On the Composability of the Notion

It is important that a notion has a clear interpretation in a larger context when used as a submodule of a system. We show how the standalone security definition for agree-and-prove can be embedded in a larger context, how the setup functionality of the standalone should be understood, and when it can be shared among different agree-and-prove instances. More formally, we show under which circumstances an agree-and-prove scheme can be securely used as a subroutine of a larger protocol when either the setup functionality is assumed to be per instance or shared among different protocol instances. While the full treatment is deferred to the full version [1] to give more room to concrete examples, the basic intuition is to show how the input-generation algorithm can encode a more general MPC-style model of protocol execution to reflect the context in which an AaP scheme is executed.

3 Application to Proof-of-Ownership of Files

File deduplication is a cornerstone of every cloud storage provider. Client-side, rather than server-side, deduplication furthermore provides the additional benefit of reducing the bandwidth requirements and improving the speed. In such a scheme, the client—instead of just uploading the file—first tries to figure out whether the server already possesses a copy of the file, and if so simply requests the server to also grant him access to the file. Several commercial providers implemented client-side deduplication using a naive scheme of identifying the file using hash values. This allowed users to covertly abuse the storage as a content distribution network, prompting the storage providers to disable client-side deduplication [14].

As a response, Halevi, Harnik, Pinkas, and Shulman-Peleg [12] introduced the first rigorous security treatment of client-side deduplication, formalizing the primitive of a proof of ownership. While intuitively their notion formalizes that a client can only claim a file he knows, Halevi et al. formalized the proof-of-ownership concept as an entropy-based notion, rather than a proof-of-knowledge based notion.

In this section, we show that our agree-and-prove notion is the natural candidate for formalizing the security of client-side deduplication. Besides the basic requirements, we also present a privacy preserving scheme that is applicable if users additionally employ client-side encryption.

3.1 Proof-of-Ownership with a Local RO

We first abstract this application as an agree-and-prove scenario which includes the setup and the relation we want to prove. We finally give a description of the scheme.

Setup. We describe the setup in very simple terms. We want to deal with an array of pairs $L = (\mathrm{id}_i, F_{\mathrm{id}_i})_{i\in[n]}$, where $\mathrm{id}_i, F_{\mathrm{id}_i} \in \{0,1\}^*$ and for all i, id_i is unique in L. The setup of the Proof-of-Ownership scenario is thus a functionality $\mathcal{F}_{\mathsf{DB},\mathsf{RO}}$ that first expects such a list from the input-generation algorithm (recall that the input-generation algorithm also defines the state of the prover). The setup gives the verifier access to the list L. The setup further provides a (non-programmable) random oracle to the prover and the verifier (but not to the input generation algorithm). The description can be found in Figure 4.

Agreement and Proof Relations. Our goal is to show that a very simple File-Ownership protocol is indeed a valid agree-and-prove scheme with the above setup. The statement that prover and verifier agree on is a file identity and the relation to be proven is that the prover knows the file with the corresponding identity. More formally, the agreement condition is as

$$C^{\mathcal{O}_{\mathcal{F}_{\mathsf{DB},\mathsf{RO}}}(\cdot,\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x) = \begin{cases} 1 & \text{if } x = \bot \vee \exists i : L(i) = (x, \cdot), \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

which can be efficiently implemented by C by calling $\mathcal{O}_{\mathcal{F}_{\mathsf{DB},\mathsf{RO}}}(\mathsf{V}, \texttt{getFile}, x)$ and verifying that the answer is some $F \neq \bot$.

Accordingly, the proof relation is defined via the condition

$$\mathcal{R}^{\mathcal{O}_{\mathcal{F}_{\mathsf{DB},\mathsf{RO}}}(\cdot,\cdot,\cdot)}(1^\kappa, x, w) = 1 \ :\leftrightarrow \ (x, w) \in L \tag{2}$$

which can be efficiently implemented by $\mathcal{R}$ by calling $\mathcal{O}_{\mathcal{F}_{\mathsf{DB},\mathsf{RO}}}(\mathsf{V}, \texttt{getFile}, x)$ and verifying that the returned value F equals w.

The scheme. The simple scheme is formally in [1] and we give here a brief overview. The agreement phase consists of the prover stating the identity of the file, of which ownership is to be proven, and providing a hash of it. The verifier checks the hash of the claimed file and upon success, informs the prover. The agreed statement is $x = \mathrm{id}$ such that there is an index i with $L(i) = (\mathrm{id}, \cdot)$. As we will see in the analysis, after this agreement phase no further proof phase is needed. Hence, the prover P_2 halts and V_2 outputs 1 if and only if V_1 did successfully derive the statement.

Analysis. In order for the verifier to accept, in the agreement phase the prover has to send (id, h) such that $H(F_{\mathrm{id}}) = h$. Since the auxiliary input from I does not depend on H, there are two possibilities: either the prover queried the random oracle at position F_{id}, in which case he has the file, or he guessed the hash. The latter can, however, only happen with negligible probability. Hence we get the following statement formally proven in [1].

Theorem 1. *The agree-and-prove scheme defined above, for the scenario formalizing proof-of-ownership with a local RO, is secure with soundness error* $p(\kappa) \coloneqq 0$.

File-Ownership setup $\mathcal{F}_{\mathsf{DB,RO}}$

- init: $H \leftarrow []$ and L is the empty array.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{DB,RO}}}(\mathsf{I}, \texttt{defineDB}, L')$: If L is the empty array, then do the following: If L' is an array of pairs $(\mathrm{id}_i, F_{\mathrm{id}_i})_{i\in[n]}$, where $\mathrm{id}_i, F_{\mathrm{id}_i} \in \{0,1\}^*$ and for all i, id_i is unique in L', then set $L \leftarrow L'$. Output L in any case.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{DB,RO}}}(i, \texttt{getFile}, \mathrm{id})$: If $i \in \{\mathsf{I}, \mathsf{V}\}$ and $\mathrm{id} \in L$ then return F_{id}. In any other case, return $\perp$.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{DB,RO}}}(i, \texttt{ROeval}, x)$: If $i \in \{\mathsf{P}, \mathsf{V}\}$ do the following: if $H[x]$ is not yet defined, first choose $y \twoheadleftarrow \{0,1\}^\kappa$ and set $H[x] := y$; finally, return $H[x]$. If $i = \mathsf{I}$, return $\perp$.

Fig. 4: The description of the concrete setup functionality.

3.2 Proof-of-Ownership with a Global RO

While the above approach is sound if prover and verifier share a random function among each other (which is only used locally in the agree-and-prove context), it is not considered secure in practice, since one does usually have to assume that access to such a random function is not exclusive to the prover. In this section, we discuss the alternative, where the random oracle is accessible by all three roles and in particular by the input generation algorithm.

The new scenario. We modify the setup slightly to allow all roles access to the random oracle, i.e., even an input generation algorithm could obtain the RO outputs and hence hashes might be part of the prior knowledge. The resulting setup functionality $\mathcal{F}_{\mathsf{DB,GRO}}$ is defined analogous to the one from Figure 4, except that also `ROeval` queries are also admitted for the role I. The relations remain the same as in equations (1) and (2), except that they are with respect to the setup $\mathcal{F}_{\mathsf{DB,GRO}}$. The proofs says essentially this: the resulting scheme is only an AaP scheme, if the extraction problem is trivial (as otherwise, someone with a precomputed hash could just convince the server, and would hence win the soundness experiment).

Insecurity of the Simple Scheme. It is easy to see that with a global RO, the scheme in Section 3.1 loses its guarantees. To be more concrete, the scheme has the following property in a GRO setting, that basically says that the scheme can only be secure if the file identifier and the hash are sufficient to efficiently recover the file corresponding to the identifier. This results in a trivial scheme, as anyone can efficiently obtain knowledge about any file in L.

Lemma 1. *Consider the scheme of Section 3.1 in the GRO setting. There exists an input generation algorithm I and a dishonest prover strategy $(\hat{P}_1, \hat{P}_2)$ for which the extraction problem of Figure 2 is at least as hard as the extraction*

problem that must recover the file F only given its identifier id *and its hash h and with access to the setup.*

Note that the above stated provable insecurity reflects practice. For instance, consider the case where a cloud storage provider uses a proof-of-ownership protocol to perform client side deduplication based on the above protocol with a standard hash function. In this setting, malicious parties can covertly abuse it as a file sharing platform by only having to exchange small hash values, with which they can then download the entire file from the cloud storage prover, as for instance pointed out by Mulazzani et al. [14].

A Secure Alternative. The natural way to obtain a secure protocol in this setting, is to let the prover prove his knowledge in the second phase. While one way to do so would be to simply send the entire file, we consider here a more efficient protocol proposed in [12]. We also extend the structure of the protocol to be privacy preserving in the next section.

The scheme. In the agreement phase, the prover still sends the identity to the verifier. In the verification phase both prover and verifier first encode F using an erasure code to obtain $X = E(F)$ and then split X into blocks of size b. The verifier then chooses uniformly at random a subset (of size n) of the blocks for which the prover has to demonstrate knowledge. We assume here an erasure code (E, D) which can restore the original data item, as long as at most an α fraction of the symbols of the encoding X are missing, for some fixed $\alpha \in (0, 1)$.

Instead of simply sending those blocks, the protocol makes use of a Merkle-Tree. That is, both the prover and verifier already compute $X = E(F)$ in the agreement phase, and then calculate the Merkle-Tree using $\mathsf{GenMT}^h(X, b)$ (where we assume here h be a random oracle for sake of simplicity). The prover then additionally sends the root value and the number of leaves ℓ of a Merkle-Tree along with the file identifier, and the verifier only accepts the statement if they match. During the second phase, the verifier can then check the correctness of the blocks by only using the control information consisting of the root and the number of leaves of the tree (instead of the entire file), thereby keeping its state small at the expense of some communication overhead. A formal description of the corresponding prover and verifier protocols is given in [1].

Analysis. Assume there is a prover who knows less than a $1 - \alpha$ fraction of the blocks, and thus cannot recover it with the erasure code. If we ask this prover to send us block b_i, for an i chosen uniformly at random by the verifier, then we will catch him with probability at least α. So if we ask him for a uniformly drawn subset of n blocks, we catch him with probability $1 - (1 - \alpha)^n$. We make this intuition precise, building on results from [12] in the following security statement and its proof that we give in [1].

Theorem 2. *The above described agree-and-prove scheme for the scenario capturing proof-of-ownership with a global RO is secure up to soundness error $p(\kappa) \coloneqq (1 - \alpha)^{n(\kappa)}$.*

3.3 On Including Privacy and Zero-Knowledge

Consider a company that wants to use an external cloud provider for file storage. To protect the confidentiality of their trade secrets they most likely want to opt for client side encryption of all the files. As naturally each file might be distributed among many employees, and the provider charges for the overall storage requirement, file deduplication is highly desirable. While all employees (or at least certain subgroups) might share the same key, coordinating on the randomness used to encrypt each file is not practical and deterministic encryption does often not provide the required level of security. Thus, neither server-side nor the naive client-side deduplication on the ciphertext are feasible.

In this section, we provide a private version of the proof-of-ownership scheme that enables client-side deduplication in this setting. The goal is that the storage provider should not be required to be trusted, and thus essentially learn nothing during the protocol run. At the same time, the storage provider should only provide access to the files to those users that already possess it, thereby preventing a rogue employee from just downloading all of the company's files. The basic idea of the protocol is that we keep the overall structure of the previous protocol, but patch it using encryption and NIZKs.

Setup. The setup corresponds to a snapshot of the system at the moment where a user wants to run the protocol. That is, it contains a list of encrypted files indexed by their respective identifiers (where the files can again be chosen by the input-generation algorithm), which have already been uploaded, together with the corresponding control information needed to run the protocol. The control information consists of an ElGamal encrypted Merkle root of the plaintext (an unencrypted root would allow the server to test whether a file is equal to a given bit-string), the number of leaves in the tree, as well as a signature binding the control information to the file identifier.

The verifier can access the encrypted files, the control information, as well as the public ElGamal key and the signature verification key. The setup either provides the prover access to the public keys only (modeling an outsider), or additionally to the symmetric key, the ElGamal decryption key, and the signing key (modeling an insider). Finally, the setup also provides the necessary CRS for the NIZK proofs to all parties. See Figure 5. Looking ahead, we will assume that the setup be programmable (to program the CRS).

Agreement and proof relations. The statement that prover and verifier agree on is simply the analogous statements from the previous section, i.e.,

$$C^{\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\cdot,\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x) = 1 \;:\leftrightarrow\; x = \bot \vee \exists i : L(i) = (x, \cdot), \tag{3}$$

Accordingly, the proof relation is defined via the condition

$$\mathcal{R}^{\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\cdot,\cdot,\cdot)}(1^\kappa, x, w) = 1 :\leftrightarrow (x, w) \in L \wedge \mathsf{KeysAssigned}, \tag{4}$$

where it is additionally checked that the prover not only knows the file but it also has the necessary keys. As before, both predicates can be efficiently evaluated using the available oracles.

File-Ownership setup $\mathcal{F}_{\mathsf{priv}}$

- The setup is (implicitly) parametrized by a cryptographic hash-function familiy $\mathcal{H}$, an erasure code (E, D), the leaf-size b for the Merkle-Tree, a symmetric encryption scheme SE, a signature scheme Sig, the ElGamal encryption scheme $\mathsf{ElGamal}$, and four associated NIZK proof systems.
- init:
  1: $\mathsf{KeysAssigned} \leftarrow \mathsf{false}$
  2: Choose $h \twoheadleftarrow \mathcal{H}$
  3: $k^{\mathsf{SE}} \leftarrow \mathsf{SE.Gen}(1^\kappa)$, $(ek^{\mathsf{ElGamal}}, dk^{\mathsf{ElGamal}}) \leftarrow \mathsf{ElGamal.Gen}(1^\kappa)$, $(vk^{\mathsf{Sig}}, sk^{\mathsf{Sig}}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$
  4: $(crs^{pt}, crs^{con}, crs^{mt,h})$
  $\leftarrow (\mathsf{NIZK}^{pt}.\mathsf{Gen}(1^\kappa), \mathsf{NIZK}^{con}.\mathsf{Gen}(1^\kappa), \mathsf{NIZK}^{mt,h}.\mathsf{Gen}(1^\kappa))$
- $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{I}, \texttt{defineDB}, L')$: If L is the empty array then do the following: if L' is an array of pairs $(\mathrm{id}_i, F_{\mathrm{id}_i})_{i\in[n]}$, where $\mathrm{id}_i, F_{\mathrm{id}_i} \in \{0,1\}^*$ and for all i, id_i is unique in L', then set $L \leftarrow L'$. In any case, output L.
  Once L is defined, for $(\mathrm{id}, F_{\mathrm{id}}) \in L$ do:
  - $X_{\mathrm{id}} \leftarrow \mathsf{SE.Enc}(k, F_{\mathrm{id}})$
  - $(T_{\mathrm{id}}, \ell_{\mathrm{id}}) \leftarrow \mathsf{GenMT}^h(E(F_{\mathrm{id}}), b)$. Let $v_{\mathrm{root,id}}$ be the root of T_{id}.
  - $c_{\mathrm{root,id}} \leftarrow \mathsf{ElGamal.Enc}(ek^{\mathsf{ElGamal}}, v_{\mathrm{root,id}})$.
  - $\sigma_{\mathrm{id}} \leftarrow \mathsf{Sig.Sgn}(sk^{\mathsf{Sig}}, (\mathrm{id}, c_{\mathrm{root,id}}, \ell_{\mathrm{id}}))$
  - $D[\mathrm{id}] \leftarrow (X_{\mathrm{id}}, c_{\mathrm{root,id}}, \ell_{\mathrm{id}}, \sigma_{\mathrm{id}})$.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{I}, \texttt{assignKeys}, -)$: Set $\mathsf{KeysAssigned} \leftarrow \mathsf{true}$.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{V}, \texttt{getFile}, \mathrm{id})$: If $\mathrm{id} \in L$ then return $D[\mathrm{id}]$. Otherwise, return $\perp$.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{P}, \texttt{getKey}, -)$: Return $(k^{\mathsf{SE}}, dk^{\mathsf{ElGamal}}, sk^{\mathsf{Sig}})$ if $\mathsf{KeyAssigned}$. Otherwise, return $\perp$.
- $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{i}, \texttt{getPub}, -)$: Return a description of h, $(crs^{pt}, crs^{eq}, crs^{mt,h}, crs^{mt,h})$, and $(ek^{\mathsf{ElGamal}}, vk^{\mathsf{Sig}})$

Fig. 5: The setup for the privacy-preserving file-ownership setting.

The achieved level of privacy. Assume a prover and verifier execute a privacy-preserving proof-of-ownership scheme. Clearly at the end of a successful protocol run, the verifier will have learned whether the prover had a file which was already present in his database. More specifically, he will learn which identifier this file had, which appears to be inevitable if we were to use the protocol to handle client-side deduplication for cloud storage. Analogously, the prover will learn whether for his input (id, F) it held that $F = F_{\mathrm{id}}$, which also seems necessary in a setting where he needs to upload the entire file otherwise.

In the remainder of the section we design a privacy preserving version of the previous Merkle-Tree based scheme. Let us briefly discuss the implication of sticking to this overall structure on privacy. In the agreement phase of the previous protocol, the prover sent the identity together with the root of the Merkle-Tree. The verifier would accept if and only if he has a file with the corresponding identity that has the same root, and only in the proof phase the

Prover Leakage $\mathcal{L}_P$

– init: –

– $\mathcal{L}_P^{\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{P},\cdot,\cdot),\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{V},\cdot,\cdot)}(1^\kappa, aux_p, query)$:

1: Parse aux_p as (id, F) and obtain $(k^{\mathsf{SE}}, dk^{\mathsf{ElGamal}}, sk^{\mathsf{Sig}})$ by calling $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{P}, \mathtt{getKey}, -)$. Return aborted if either one is not possible. Continue otherwise.

2: Compute $(T, \ell) \leftarrow \mathsf{GenMT}^h(E(F), b)$ and let v_{root} denote the root of T.

3: Obtain $(X_{\mathrm{id}}, C', \ell', \sigma)$ from $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{V}, \mathtt{getFile}, \mathrm{id})$

- If this fails or $\ell \neq \ell'$, return $(\mathrm{id}, \ell, 0)$
- Else, $v'_{\mathrm{root}} \leftarrow \mathsf{ElGamal.Dec}(dk^{\mathsf{ElGamal}}, C')$ and let $a \coloneqq (v_{\mathrm{root}} = v'_{\mathrm{root}})$. Return (id, ℓ, a).

Fig. 6: The description of leakage a dishonest verifier can obtain about the prover's view in a single run of the protocol.

prover had to show that he knows the entire file. In our scheme, a dishonest prover that has the decryption key dk will be able to learn whether for an identifier id and a Merkle-Tree root v_{root} of his choice, there exists a file F_{id} with root v_{root}, leaking slightly more than an optimal protocol. This leakage can be turned into a formal description of a leakage oracle $\mathcal{L}_V$ in a straightforward way. We defer the description to Figure 7.

On the other side, in our protocol a dishonest verifier will learn the file identifier the prover has and also the length of the prover's file (the number of Merkle leaves). Furthermore, if a file with this identifier exists in his database, then he will also learn whether it is the same one. A formal definition of the leakage machine $\mathcal{L}_P$ is given in Figure 6.

The scheme. The scheme basically follows the approach of the previous scheme of using a Merkle tree (in this section, we assume a collision-resistant hash function h and not a random oracle), however encrypts all the nodes of the tree using ElGamal encryption and then proves the consistency using NIZK proofs. We give here an overview and refer to [1] for the pseudo-code of the scheme. In the following, let $G = \langle g \rangle$ be a cyclic group of prime order q with a generator g, in which the decisional Diffie-Hellman assumption is assumed to hold, and let $h\colon G^2 \to G$ be a collision resistant function.

We first describe the agreement phase. While in the original protocol the prover sends the identity id to the server together with the root of the Merkle tree, in the privacy preserving scheme he sends the identity alongside a fresh ElGamal encryption $(c_0, c_1) \coloneqq (g^r, g^{dkr} \cdot v_{\mathrm{root}})$ of the root. If the verifier has a file with that identity, then they proceed to check whether it encrypts the same root as the corresponding one from the verifier's control information $(c'_0, c'_1) \coloneqq (g^s, g^{dks} \cdot v'_{\mathrm{root}})$, i.e, whether $v'_{\mathrm{root}} = v_{\mathrm{root}}$. To this end, the verifier chooses $t \in \mathbb{Z}_q^*$ uniformly at random and sends back $(d_0, d_1) \coloneqq \left(g^{t(s-r)}, g^{t \cdot dk(s-r)} \cdot (v'_{\mathrm{root}} \cdot v_{\mathrm{root}}^{-1})^t\right)$

Verifier Leakage $\mathcal{L}_V$

- init: queried $\leftarrow$ false
- $\mathcal{L}_V^{\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{P},\cdot,\cdot),\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{V},\cdot,\cdot)}(1^\kappa, aux_v, query)$:
  1: If queried, return $\perp$, else set queried $\leftarrow$ true and continue.
  2: Parse $query$ as (id, v, dk). Return $\perp$ if not possible.
  3: Obtain $(k^{\mathsf{SE}}, dk^{\mathsf{ElGamal}}, sk^{\mathsf{Sig}})$ via a call to $\mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{P}, \mathtt{getKey}, -)$. Return $\perp$ if not possible.
  4: Call $D \leftarrow \mathcal{O}_{\mathcal{F}_{\mathsf{priv}}}(\mathsf{V}, \mathtt{getFile}, \mathrm{id})$.
  - If $D = \perp$, return 0.
  - Else, parse D as $(X_{\mathrm{id}}, c_{\mathrm{root,id}}, \ell_{\mathrm{id}}, \sigma_{\mathrm{id}})$.
    Let $v_{\mathrm{root,id}} \leftarrow \mathsf{ElGamal.Dec}(dk, c_{\mathrm{root,id}})$ and return 1 if $v = v_{\mathrm{root,id}}$. Return 0 otherwise.

Fig. 7: The leakage a dishonest prover can obtain about the verifier's view in a single run of the protocol.

obtained from dividing the two encryptions. Note that t is used to blind the verifier's Merkle tree root, which would otherwise leak to the prover knowing dk. The prover can then check whether $(v'_{\mathrm{root}} \cdot v_{\mathrm{root}}^{-1})^t = 1$ by raising the first element by the decryption key dk, and inform the verifier accordingly. Observe that since G is of prime order, we have that $x^t = 1$, for $t \in \mathbb{Z}_q^*$, if and only if $x = 1$, and thus the prover's check succeeds if and only if $v'_{\mathrm{root}} = v_{\mathrm{root}}$. If the verifier does not have a file with identifier id, then he chooses $d_0 \in G$ and $t \in \mathbb{Z}_q^*$ uniformly at random and sends $(d_0, (d_0)^t)$ instead, to conceal this fact. With overwhelming probability $t \neq dk$ and, thus, the prover will abort assuming that the Merkle roots don't match.

To protect against dishonest behaviors, during the agreement phase, both parties additionally prove with each message that it has been computed correctly using a NIZK proof for the languages introduced below, which are parametrized in (a description of) the group G, the generator g, the group order q, the file identifier space $\mathcal{ID}$, and the signature scheme Sig including the verification-key space $\mathcal{VK}$ and the signature space Σ.

- For the first message from the prover to the verifier, let NIZK^{dk} be a NIZK proof system for the language $L^{dk} := \{x \mid \exists w\ (x, w) \in R^{dk}\}$, where R^{dk} is defined as follows: for $x = ek \in G$ and a witness $w = dk \in \mathbb{Z}_q$, $R^{dk}(x, w) = 1$ if and only if $ek = g^{dk}$. Hence, the prover shows that he knows the decryption key, and thus also the corresponding plaintext v_{root} of his first message.
- For the message from the verifier to the prover, let NIZK^{con} be a NIZK proof system for the language $L^{con} := \{x \mid \exists w\ (x, w) \in R^{con}\}$, where R^{con} is defined as follows: for $x = \big(c_0, c_1, d_0, d_1, \mathrm{id}, \ell, vk\big) \in G^4 \times \mathcal{ID} \times \mathbb{N} \times \mathcal{VK}$ and a witness $(c'_0, c'_1, t, \sigma) \in G^2 \times \mathbb{Z}_q \times \Sigma$, $R^{con}(x, w) = 1$ if and only if

$$\Big((d_0, d_1) = \big((c'_0 \cdot c_0^{-1})^t, (c'_1 \cdot c_1^{-1})^t\big) \wedge \mathsf{Sig.Vrf}(vk, \sigma, (\mathrm{id}, c'_0, c'_1, \ell))\Big) \vee (d_0)^t = d_1.$$

– For the second message from the prover to the verifier, let NIZK^{eq} be a NIZK proof system for the language $L^{eq} := \{x \mid \exists w\ (x,w) \in R^{eq}\}$, where R^{eq} is defined as follows: for $x = \big(ek, d_0, d_1\big) \in \mathbb{Z}_q \times G^2$ and a witness $w = dk \in \mathbb{Z}_q$, $R^{eq}(x,w) = 1$ if and only if

$$(ek, d_1) = (g^{dk}, d_0^{dk}).$$

Finally, in the prove-phase, the server selects again a number of leaf indexes and the prover replies with the encrypted siblings path together with NIZK's to prove that the path is correctly built, defined as follows.

– Let the language $L^{mt,h} := \{x \mid \exists w\ (x,w) \in R^{mt,h}\}$ be defined via the following relation $R^{mt,h}$: for $x = \big(ek, n_0, n_1, l_0, l_1, r_0, r_1\big) \in G^7$ and a witness $w = dk \in \mathbb{Z}_q$, $R^{mt,h}(x,w) = 1$ if and only if

$$ek = g^{dk} \wedge \mathsf{ElGamal.Dec}(dk, (n_0, n_1)) = \\ h\big(\mathsf{ElGamal.Dec}(dk, (l_0, l_1)), \mathsf{ElGamal.Dec}(dk, (r_0, r_1))\big).$$

The verifier furthermore checks that in each path, the ciphertext of the root is the one the prover sent in the agreement phase.

Analysis. The described agree-and-prove protocol achieves the same level of security as the plain Merkle-Tree based protocol, analyzed in the last section, but additionally provides the described level of privacy. This is summarized in the following theorem which is proven in [1].

Theorem 3. *The above described agree-and-prove scheme, for the agree-and-prove scenario consisting of the setup functionality from Figure 5 and the relations from equations* (3) *and* (4)*, is secure up to knowledge error* $p(\kappa) := (1-\alpha)^{n(\kappa)}$*. Furthermore, it is verifier zero-knowledge up to* $\mathcal{L}_P$ *and prover zero-knowledge up to* $\mathcal{L}_V$*, where* $\mathcal{L}_P$ *and* $\mathcal{L}_V$ *are defined as above and specified as pseudo-code in Figures 6 and 7, respectively.*

4 Application to Client Authentication

Client authentication has gained a lot of attention from the security community and plenty of client authentication protocols have been proposed and studied over the years, such as [6, 21, 22, 13]. Those works however phrase security in a property based manner with rather particular attack models making them not directly applicable in an overall cryptographic analysis based on explicit hardness assumptions and reduction proofs. Multi-factor authentication has gotten some attention in the cryptographic community by Shoup and Rubin on session-key distribution with smart cards [15], which however does not reflect the usual password plus second-factor based setting we observe in practice for which no formal model exists. In the following, we show how the agree-and-prove notion can be used to formalize the above mentioned properties in a sound and thorough manner. We focus here just on the 2-FA case and refer to [1] for the full treatment.

Two-Factor Setup $\mathcal{F}_{\text{2-FA}}$

- The setup is parameterized a user-administration mechanism UAdmin and a PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.
- init: $db \leftarrow \mathsf{UAdmin.Init}(1^\kappa)$; initialize PW, and Keys to empty maps, and $\mathsf{Assigned}$ to a map pre-initialized to false.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{I}, \texttt{SetPassword}, un, pw)$:
  1: If $\mathsf{Keys}[un]$ is not defined yet, sample $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and store $\mathsf{Keys}[un] \leftarrow (pk, sk)$.
  2: Set $db \leftarrow \mathsf{UAdmin.Set}(db, un, pw)$ and $\mathsf{PW}[un] \leftarrow pw$.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{I}, \texttt{Assign}, un)$: Set $\mathsf{Assigned}[un] \leftarrow \mathsf{true}$.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{I}, \texttt{GetUsers}, -)$: Return PW.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{i}, \texttt{GetDB}, -)$: If $i \in \{\mathsf{I}, \mathsf{V}\}$, return db. Otherwise, return $\perp$.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{i}, \texttt{TokenEval}, un, x)$: If $i = \mathsf{I}$ and $\mathsf{Keys}[un]$ is defined, or $i = \mathsf{P}$ and $\mathsf{Assigned}[un]$, then let $(pk, sk) \leftarrow \mathsf{Keys}[un]$ and return $\mathsf{Dec}(sk, x)$. In any other case, return $\perp$.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{P}, \texttt{IsAssigned}, un)$: If $\mathsf{Assigned}[un]$ then return 1, otherwise 0.
- $\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\mathsf{i}, \texttt{GetPublicKey}, un)$: If $i \in \{\mathsf{I}, \mathsf{V}\}$ and $\mathsf{Keys}[un]$ is defined, then let $(pk, sk) \leftarrow \mathsf{Keys}[un]$ and return pk. Otherwise, return $\perp$.

Fig. 8: The description of the concrete setup functionality for 2-FA.

Two-Factor Authentication. The scheme we consider in this section combines both factors password and token. We consider the following type of hardware token, analogous to [15]: upon producing the token a public/secret key pair of a PKE scheme is chosen. The secret key is then securely embedded into the token—that provides a decryption oracle—and the public key is stored for verification. The setup is parametrized by what we call a user-administration mechanism UAdmin that allows a user to register to the service with a given password or update its password (algorithm Set) whose definition we defer to [1] due to space constraints. The input-generation algorithm can *assign* a username to the user, thereby granting him access to the token. In addition, the input generation algorithm gets query access to all the tokens—modeling that the prover might have had temporary access to those tokens in the past. The verifier can access the password database, as well as all public keys corresponding to the secret keys embedded in the tokens. The description can be found in Figure 8.

The agreement condition requires that the parties either have to agree on a valid username x or abort. For correctness, we require that the honest parties only agree on a username if the prover possesses the corresponding token.

$$C^{\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\cdot,\cdot,\cdot)}(1^\kappa, aux_p, aux_v, x) = \begin{cases} 1 & \text{if } x = \perp \vee (\mathsf{PW}[x] = pw \wedge aux_p = (x, pw) \wedge \mathsf{Assigned}[x]) \\ * & \text{if } x = \perp \vee (\mathsf{PW}[x] = pw \wedge aux_p \neq (x, pw) \wedge \mathsf{Assigned}[x]) \\ 0 & \text{otherwise,} \end{cases} \quad (5)$$

which can be efficiently implemented using oracle access to $\mathcal{F}_{\text{2-FA}}$. The proof relation for two-factor authentication checks two conditions: it checks *knowledge* of the password and *access* to the token. Knowledge of the password is as usually phrased as the witness w which the knowledge extractor has to extract. Access, or possession, of the token on the other hand cannot be phrased as a witness extraction problem—in the end we do not want to require the extractor to extract the internal state of a secure hardware token. Rather, it is simply a property of the setup that is checked by the relation. We thus can define the relation via the condition $\mathcal{R}^{\mathcal{O}_{\mathcal{F}_{\text{2-FA}}}(\cdot,\cdot,\cdot)}(1^\kappa, x, w) = 1 :\leftrightarrow \mathsf{PW}[x] = w \wedge \mathsf{Assigned}[x]$.

An agree-and-prove scheme for the above relation can be built in a black-box way from a secure password-based authentication scheme that has to check that the prover has access to the token by requesting to decrypt the encryption of a random challenge. The details are deferred to the full version [1].

References

[1] Christian Badertscher, Daniel Jost, and Ueli Maurer. Generalized proofs of knowledge with fully dynamic setup. Cryptology ePrint Archive, Report 2019/662, 2019. `https://ia.cr/2019/662`.

[2] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, 2020.

[3] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 390–420, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[4] David Bernhard, Marc Fischlin, and Bogdan Warinschi. Adaptive proofs of knowledge in the random oracle model. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 629–649, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[5] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized schnorr proofs. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 425–442, 2009.

[6] Hung-Yu Chien, Jinn-Ke Jan, and Yuh-Min Tseng. An efficient and practical solution to remote authentication: Smart card. *Computers & Security*, 21(4):372 – 375, 2002.

[7] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.

[8] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, Jun 1988.

[9] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA, 2006.

[10] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[11] Lorena González-Manzano and Agustin Orfila. An efficient confidentiality-preserving proof of ownership for deduplication. *J. Netw. Comput. Appl.*, 50(C):49–59, April 2015.

[12] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 491–500, New York, NY, USA, 2011. ACM.

[13] I-En Liao, Cheng-Chi Lee, and Min-Shiang Hwang. A password authentication scheme over insecure networks. *Journal of Computer and System Sciences*, 72(4):727 – 740, 2006.

[14] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 5–5, Berkeley, CA, USA, 2011. USENIX Association.

[15] Victor Shoup and Avi Rubin. Session key distribution using smart cards. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 321–331, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[16] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 472–482, Washington, DC, USA, 1987. IEEE Computer Society.

[17] Thomas Vidick and Tina Zhang. Classical proofs of quantum knowledge. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 630–660. Springer, 2021.

[18] Douglas Wikström. *On the security of mix-nets and hierarchical group signatures*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2005.

[19] Jia Xu and Jianying Zhou. Leakage resilient proofs of ownership in cloud storage, revisited. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, pages 97–115, Cham, 2014. Springer International Publishing.

[20] Guomin Yang, Duncan S. Wong, Huaxiong Wang, and Xiaotie Deng. Formal analysis and systematic construction of two-factor authentication scheme (short paper). In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security*, pages 82–91, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[21] Eun-Jun Yoon, Eun-Kyung Ryu, and Kee-Young Yoo. Efficient remote user authentication scheme based on generalized elgamal signature scheme. *IEEE Transactions on Consumer Electronics*, 50(2):568–570, May 2004.

[22] Eun-Jun Yoon and Kee-Young Yoo. New authentication scheme based on a one-way hash function and diffie-hellman key exchange. In Yvo G. Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *Cryptology and Network Security*, pages 147–160, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.