MLS: how Zero-Knowledge can secure Updates

Julien Devigne, Céline Duguey, Pierre-Alain Fouque

▶ **To cite this version:**

Julien Devigne, Céline Duguey, Pierre-Alain Fouque. MLS: how Zero-Knowledge can secure Updates. ESORICS 2021, Oct 2021, Darmstadt / Virtual, Germany. hal-03558760

HAL Id: hal-03558760
https://hal.science/hal-03558760

Submitted on 4 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MLS: how Zero-Knowledge can secure Updates

Julien Devigne[1], Céline Duguey[1,2], and Pierre-Alain Fouque[3,2]

[1] DGA Maîtrise de l'information, Bruz, France julien.devigne@intradef.gouv.fr
[2] Irisa, Rennes, France celine.duguey@irisa.fr
[3] Univ Rennes, CNRS , pierre-alain.fouque@irisa.fr

Abstract. The Messaging Layer Security (MLS) protocol currently developed by the IETF aims at providing a group secure messaging solution. The goal is to provide end-to-end security, including Forward Secrecy and Post Compromise Secrecy, properties well studied for one-to-one protocols. MLS proposes a tree-based regular asynchronous update of the group secrets, where a single user can alone perform a complete update. A main drawback is that a malicious user can create a denial of service attack by sending invalid update information.
In this work, we propose a solution to prevent this kind of attacks, giving a checkpoint role to the server that transmits the messages. In our solution, the user sends to the server a proof that the update has been computed correctly, without revealing any information about this update. We use a Zero-Knowledge (ZK) protocol together with verifiable encryption as building blocks. As a main contribution, we provide two different ZK protocols to prove knowledge of the input of a pseudo random function implemented as a circuit, given an algebraic commitment of the output and the input.

Keywords: Cryptographic Protocols · Messaging Layer Security - MLS · Secure Messaging · Zero-Knowledge.

1 Introduction

Secure messaging protocols have been widely adopted over the last few years. The privacy offered by encrypted communication seduces billions of users worldwide. A significant number of application providers have settled their security on the Double Ratchet Algorithm [36], often identified as Signal. This protocol provides End-to-End confidentiality, as well as Forward Secrecy (FS) and Post Compromise Secrecy (PCS). The Double Ratchet however is dedicated to one-to-one communications. Messaging Layer Security (MLS) targets secure *group* messaging and is developed by the Internet Engineering Task Force (IETF). The goal is to obtain similar security properties as those in one-to-one protocols. The group keys are computed and regularly updated in a protocol called TreeKEM, based on a tree structure. In MLS, each member of the group is represented by a tree leaf and the group secret is given by the tree root. Each member can update the group secret as well as its own keys. He can also, if authorized, add or remove members. To perform an update, a user sends to the other leaves

secret information which depend on their position in the tree. In this paper, we are concerned with an open problem identified in the MLS draft: how to be sure that each user receives a valid update information? In other words: how to be sure that the updating user is not cheating? The current protocol provides verification elements for each user to check whether the update he received is valid, but it does not prevent a malicious updater to create denial of service attacks. Such attacks would prevent the updating process, seriously damaging FS and PCS properties, that seduce the users. Consequently, we propose a solution in which the users only receive valid updates: the server which transmits the update messages can check their validity before forwarding them. The server is given a check-point role. If the server were to be dishonest, he could not create a malicious update by himself. Hence we only add a layer of security, through the server, without modifying the core of MLS. A main building block of our solution is a ZK protocol, inspired from the recent multi party computation (MPC) in the head solutions ([29], [25], [33]).

Our contribution. As a first contribution, we show how to combine a ZK protocol with a verifiable encryption solution to solve the open problem identified in the IETF draft for MLS, in a light version of the protocol. The idea is to enable an intermediate server to perform a blind verification (on encrypted and committed data) that each update information sent by the updater is correctly computed. The ZK proof is provided on a statement that mixes a circuit evaluation (an HKDF derivation, defined in [35]) and an algebraic commitment (typically a Pedersen commitment, described in [38]). The verifiable encryption scheme proves to the server (*i.e.* the verifier) that the encrypted data is the one that is committed to and verified in the proof.

As a second contribution, we propose two ZK protocols for statements that compose an algebraic commitment and the circuit computation of a pseudo-random function family (PRF) f. More precisely, we prove the knowledge of a witness x such that public commitments C_x, C_y are algebraic commitments of x and $f(x)$, with $f(x)$ remaining secret. Our approach is based on the MPC in the head paradigm, introduced in [29]. We consider the recent ZK proof system ZKBoo [25] and its improvements ZKBoo++ [16] as well as KKW [33]. Our first approach consists in considering a Boolean circuit that computes two tags of the form $t = ax + b$. One tag is on the secret input x, the second is on the secret output $f(x)$. The proof on this circuit binds the secret input and output to the tag values. In a second step, we use the linear properties of the algebraic commitment to show that the committed values are also bound by the tags. Finally, we invoke properties of the PRF to show that their exists only one solution to the tag equations. Our second approach is directly inspired by a recent work of Backes *et al.* [6]. The first step is to provide commitments to the bits of the secret input and output. Then, we call the algebraic properties of the commitment scheme to link those bits with values used in the circuit proof. Previous works have proposed solutions for such algebraic and circuit composition statements [17] and [1] that we describe in the following. However, the former is inherently interactive and the latter uses SNARKs, where the

burden of the proof is mainly on the prover side, which does not fit our case as smartphones have resources to be saved.

Related work. *Secure Messaging.* Secure messaging, and more particularly Ratcheted Key Exchange (RKE) have been widely studied over the past ten years. We can cite the first analysis of the Signal protocol [20] as well as the following works on RKE ([10], [39], [30], [32], [3]). Literature for the group version is more limited. In [19] Cohn Gordon *et al.* introduced the notion of Asynchronous Ratcheted Trees (ART). These ART are Diffie Hellman based binary trees in which the update process of a node involves entropy coming from both its children. In MLS, the underlying TreeKEM protocol is inspired by ART. A main difference is that a single leaf can generate the update data for each of its ancestor nodes. TreeKEM has been initially formalized in the technical paper [11] and has then evolved to reach the actual description available on the prevailing draft 11 [**?**]. Alwen *et al.* formalize in [4] a Continuous Group Key Agreement (CGKA) derived from the two-party Continuous Key Agreement defined in [3]. They provide a security model for CGKA and show that the protocol TreeKem does not achieve optimal FS and PCS security. However they prove that a better security can obtained by using an updatable public key encryption scheme. Our solution is compatible with this improvement. In [2], Alwen *et al.* focus on the addition and revocation process. Both works consider a partially active attacker, that can leak a user's state, has full control on message delivery but cannot use the knowledge of secret keys to insert his own data.

Zero-Knowledge proofs. ZK proofs have proved to be a powerful tool in cryptography, since their conception in the mid 1980s. It has been shown that ZK proofs exist for any NP language. However, efficient ZK protocols are designed for a small class of language and do not extend to any NP languages. Sigma protocols (Σ-protocols), clearly described in [21], are very efficient for proving algebraic relations, whereas other protocols have been designed for proving statements that can be expressed as a circuit. Among them, Garbled Circuits based schemes, as introduced in [31], which are inherently interactive, and SNARKs, as designed in [24], [27], [34]. SNARKs are non-interactive arguments (with computational soundness) of knowledge with small proofs and light verification: the burden of the proof is on the prover side. They are proven secure in the common reference string (CRS) model: a common trusted public input has to be shared by the prover and the verifier. Practical implementations based on pairings are in use in real life protocols such as cryptocurrencies. More recently, the MPC in the head paradigm, introduced in [29], leads to very efficient proof system without CRS. The seminal paper [25] introducing ZKBoo proposes the first efficient ZK proof of a hash function computation. Further works significantly optimize the efficiency, such as Ligero [5], or ZKBoo++ [16] and KKW [33] that were directly incorporated in the post quantum signature scheme Picnic.
In real life however, many applications need to provide proofs on statements that mix algebraic and non algebraic parts. Expressing the algebraic part as a circuit would considerably increase the circuit size and reduce the efficiency. One can express each gate of a circuit as an algebraic relation that can be proven with a

Σ-protocol, but this solution is clearly non desirable as circuits for hashing may have thousands of gates. Considering this, combining efficiently algebraic and non algebraic proofs has revealed to be an important challenge.
In [17], Chase *et al.* combine for the first time proofs on algebraic and non algebraic statements, solving the open problem described above. They propose two constructions to provide a circuit proof on a committed input. Their two constructions are based on Garbled Circuits, which enable efficient ZK proofs for non algebraic statements but are inherently interactive. Our solutions are close to theirs in the sense that their first proposal uses bit wise commitment on an secret input, and their second proposal includes a one-time mac computation in the circuit to be garbled. However, their proposal heavily relies on the garbling protocol and can not be transposed to the non interactive setting. In terms of efficiency, their second solution augments the cost of garbling in $\mathcal{O}(|x|s + s)$ where s is the security parameter required for the one time mac (which can be chosen less than the size of the witness x). This corresponds to the multiplication on integers ax included in the circuit. In our tag based solution, we show that the multiplication does not significantly increase the proof size.
More recently, Agrawal *et al.* in [1] propose a solution for modular composition of algebraic and non algebraic proofs. Their solution is non interactive, based on Sigma protocols and QAP-based SNARKs. As explained in their work, the "key ingredient we need from a SNARK construction is that the proof contains a multi-exponentiation of the input/output". They compose it with a proof that the exponents in a multi-exponentiation correspond to values committed to in a collection of commitments. From this result, they show how to obtain proofs for AND, OR and composition of two statements, either algebraic or circuit. The authors work with SNARK proofs for the circuit part in order to obtain small proofs and a light verification step. These properties are desirable for their applications such as privacy-preserving credentials or crypto-currencies proofs of solvency. The counterpart is that the prover has to provide a higher computational effort. In our application case - a proof of an honest key update in MLS - the verifier turns out to have a larger computation power than the prover.
Finally, Backes *et al.* propose in [6] an extended version of ZKBoo++ that allows algebraic commitments on the secret input of the circuit. Their protocol is non interactive and the computational cost is balanced between the prover and the verifier. Their solution requires to commit to each bit of the secret input and to commit to internal values of the ZKBoo++ circuit proof. We extend their result to the case of a committed output in our second zero-knowledge solution. We focus on the MPC in the head paradigm in order to provide proofs in which the amount of work is balanced between both parties.

Organization of the paper. In section 2, we quickly recall the definitions concerning ZK proofs, commitment schemes, and verifiable encryption. We detail more particularly ZKBoo, an efficient protocol for large boolean circuits such as hash computation. In section 3, we present the protocol MLS, focusing on the process to update the group secret, and we describe our solution to improve the security of the update mechanism. The section 4 is dedicated to our two proto-

cols, `CopraZK` and (bitwise) `ComInOutZK`, for proving knowledge of the preimage of a PRF function when only commitments of the input and the output are publicly available. Finally, in section 5 we present implementation results concerning our first solution `CopraZK`.

2 Backgrounds

Zero-Knowledge. Consider an NP relation $\mathcal{R}$, *i.e.* given a witness w and an input x, $\mathcal{R}(x, w) = 1$ can be decided in polynomial time. Let $\mathcal{L}$ be the language associated to $\mathcal{R}$, $\mathcal{L} = \{x | \exists w \text{ such that } \mathcal{R}(x, w) = 1\}$. A ZK proof of knowledge for $\mathcal{L}$ allows a prover to convince a verifier (who knows x), that he knows a witness w for $\mathcal{R}$, without revealing any further information on w. It shall be correct (if the prover and the verifier are honest, the verifier always accepts), sound (a corrupted prover can make the verifier accept a false statement only with negligible probability), and zero-knowledge (no information on w leaks from the proof). Following the notation of [13], we write $\mathsf{PK}\{w_1, \dots w_s : \mathcal{R}(w_1, \dots, w_s, x_1 \dots x_t) = 1\}$ to denote the proof of knowledge of the secret witnesses $w_1, \dots w_s$ that satisfy the relation $\mathcal{R}$ with the public values $x_1, \dots, x_t$.

Commitment Schemes. A commitment scheme involves a *Committer* and a *Receiver* who share public parameters. On entrance a message x and an additional opening information r, the commitment protocol produces a value $c = \mathsf{Com}(x, r)$ such that c shall not reveal any information about x; this is the hiding property. The *Committer* can open its commitment c by revealing r and x with the property that only the secret x shall produce a valid opening for c; this is the binding property. For our second ZK protocol, we will require an extra property, called equivocality. Briefly, a commitment is equivocable if there exists a trapdoor T such that, given a commitment C, its opening information, and T, it is possible to open C to any value. Equivocality comes with a specific extractor that, given two different openings (x_1, r_1), (x_2, r_2) to a single commitment C, can extract the trapdoor T. The Pedersen commitment scheme [38] is an equivocable scheme with unconditional hiding and computational binding. It is routinely used because it interacts nicely with linear relations. This scheme is defined as follows: let $\mathbb{G}$ be a cyclic group of prime order q, P a generator and $Q \in \mathbb{G}$ such that the discrete log of Q in base P is unknown. Then, $\mathsf{Com}(x) = xP + rQ$ where r is sampled at random in $\mathbb{Z}_q$. Let C_1, C_2 be commitments to values x_1, x_2. If $a, b \in \mathbb{Z}_q$ are public values, then one can efficiently prove the following : $PK\{x_1, x_2, r_1, r_2 : C_1 = x_1P + r_1Q \wedge C_2 = x_2P + r_2Q \wedge ax_1 + x_2 = b \mod q\}$. The trapdoor for equivocality is given by the discrete log of Q in base P.

MPC in the head. Ishai *et al.* introduced in [29] a new paradigm for ZK proofs, called MPC in the head. This solution reveals to be very competitive in terms of efficiency for ZK proofs performed on circuits. The idea is that the prover performs a virtual MPC and obtains several views. He commits to these views and opens only a sub-part of them required by the verifier. This

Σ-protocol can be turned into a non-interactive proof using Fiat-Shamir transformation [23]. ZKBoo [25] generalizes IKOS to any relation $\mathcal{R}_\phi$ defined by a function $\phi : X \to Y$ ($\mathcal{R}_\phi(y, x) \leftrightarrow y = \phi(x)$), as long as the function ϕ can be computed in a specific manner identified as a (2,3)-decomposition. Given this specific MPC computation, the prover first shares its secret input x into $(x_1, x_2, x_3) = \mathsf{Share}(x)$ such that $x = x_1 \oplus x_2 \oplus x_3$. Then he runs the MPC and obtains three distinct views w_1, w_2, w_3 and from each of this view he gets an output share $y_i = \mathsf{Output}(w_i), i \in \{1, 2, 3\}$ such that $y_1 \oplus y_2 \oplus y_3 = \phi(x)$. A detailed description is given in Appendix A.

Verifiable encryption. Verifiable encryption aims at convincing a verifier that an encrypted data satisfies some properties without leaking any information about the data itself. In such 2-party protocol, a prover and a verifier share in a common input string a public key encryption scheme Enc, a public key pk for Enc, and a public value y. At the end, the verifier either accepts and obtains the encryption of a secret value x under pk such that x and y verify some relation $\mathcal{R}$ or rejects. It is worth noticing that the prover does not need to know the secret key sk, that usually belongs to a third party. Verifiable encryption often appears in the domain of anonymous credentials, fair exchange signatures, or verifiable secret sharing [40]. In [12], Camenish and Damgård describe how to provide a proof that an encrypted value is a valid signature, using any semantically secure encryption scheme. The idea is to take advantage of the Σ-protocol that already exists for a relation $\mathcal{R}(x, y)$, to provide an evidence that an encrypted value is the witness x for this relation. In our application, we need to prove that the encrypted data x, is the one that is linked by a Pedersen commitment $C_x = y$. As a Pedersen commitment comes with an associated Σ-protocol, the Camenish-Damgård scheme applies naturally. There are interesting ways towards more efficient schemes, *e.g.* [18] or [14]. However, the main benefit of the Camenish-Damgård solution is that it does not introduce any change in the original MLS specification, as we can still use the encryption scheme required in MLS draft. More details are given in Appendix B. We denote by $\mathsf{VerifEnc}_{\mathsf{Enc},pk}(m : r)$ the encryption of a message m (using randomness r) under the public key pk with the encryption scheme Enc and the associated proof. We omit the randomness r when it is not necessary to explicitly mention it.

3 MLS Updates

We explain how the MLS update mechanism works and our more secure solution.

3.1 Message Layer Security

MLS is a protocol currently under development by the IETF to provide an End-to-End secure group messaging application. The idea is to enable a group of users to share a common secret that can be updated regularly by any user. One of the open issues in the IETF draft is that the validity of an update message can only be checked *after* it has been received. This open issue was clearly identified

until draft 9 included. In the recent draft 11, all open issues have been removed. However, to our knowledge, there is still no solution to this problem, which can lead to denial of service attacks. Currently in MLS, the authors require an hybrid public key encryption (HPKE) scheme, as designed in [8], which comprises a KEM, an AEAD encryption scheme and a hash function. Briefly, an asymmetric KEM protocol is used to compute and transmit a symmetric key k. Then data are symmetrically encrypted under key k with the AEAD encryption scheme. As symmetric encryption is far more efficient than its asymmetric counterpart, this hybrid method is a common practice. In the rest of this work, we denote by $\mathsf{Enc}_{pk}(m : r)$ the HPKE encryption of a message m under the public key pk using randomness r. The asymmetric part of Enc is based on an elliptic curve E defined on a finite field $\mathbb{Z}/p\mathbb{Z}$ with base point P of order a prime q. MLS also supposes the existence of a broadcast channel for each group, which distributes all the messages to each group member, conserving the order.

MLS tree. MLS is based on a binary tree structure (Figure 1) where users correspond to leaves and each node is associated to a secret value. Each user U has a long term identity signing key and an initial key package for the encryption scheme Enc (both certified by a PKI). We will simply represent the key package as a public/private key pair (pk_U, sk_U) valid for the encryption scheme Enc.

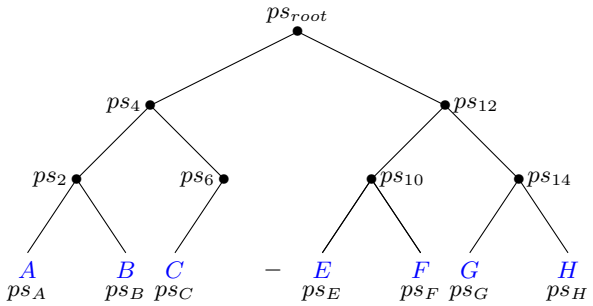

Fig. 1. A view of the MLS tree. Nodes are implicitly numbered from left to right, independently from their height. Leaves are associated to a user represented as a letter. Each node i has a secret ps_i. A leaf secret is indexed with its user name.

The group key is derived from the root secret. Each child node knows the secret of each of its ancestors and only of its ancestors. To each node i corresponds:

- a path secret ps_i ;
- a secret and public key $sk_i, pk_i = \mathsf{deriveKeyPair}(ps_i)$.

The exact derivation depends on the elliptic curve. We define, w.l.o.g, $(sk_i, pk_i) = \mathsf{deriveKeyPair}(ps_i) = (\mathsf{deriveSK}(ps_i), \mathsf{deriveSK}(ps_i)P)$ where $\mathsf{deriveSK}$ is a PRF. A user knows the secret (and so the secret keys) in its direct path, composed of itself and its direct ancestors. Moreover, each user keeps an up-to-date global view of the tree, as a hash value of each node's public information.

Updates. The path secrets and derived keys are regularly updated. Each update gives birth to a new epoch. Each epoch corresponds to a root secret, from which are derived several application keys. We focus on how the root secret is updated, not on how it is used.
To update the tree, a user B generates a new secret ps'_B. The path secrets in the direct path will be successively derived from ps'_B. We note $\mathsf{H}_p(ps_i)$ for the function $\mathsf{HKDF} - \mathsf{expand}(ps_i, \text{“}path\text{”}, \text{“”}, Hash.length)$. The update mechanism is given in Figure 2.

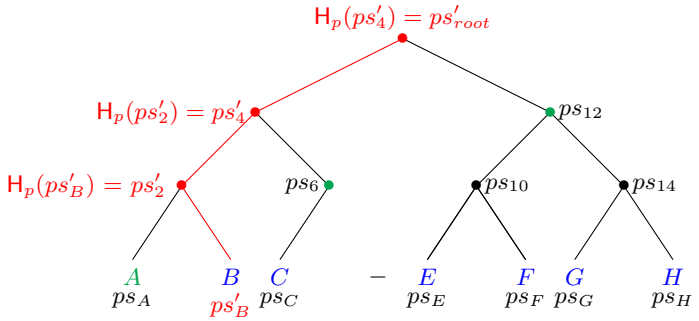

Fig. 2. Update process. User B updates its secrets. Path secrets are updated along its direct path (in red). The update secrets are sent to its copath nodes (in green).

When B updates its secret $ps_B \rightarrow ps'_B$, he first computes the new node data for each node on its path:

- $ps'_2 = \mathsf{H}_p(ps'_B)$, $pk_2 = \mathsf{deriveSK}(ps'_2) \cdot P$
- $ps'_4 = \mathsf{H}_p(ps'_2)$, $pk_4 = \mathsf{deriveSK}(ps'4) \cdot P$
- $ps'_{root} = \mathsf{H}_p(ps'_4)$, $pk_{root} = \mathsf{deriveSK}(ps'_{root}) \cdot P$

Then he sends for each node on its copath the necessary secret material for the users under this node to perform the same update (*i.e.*, to obtain the new secret keys for their updated ancestors). Following our example in Figure 2, B has to send ps'_2 to A, ps'_4 to nodes C and 6 and ps'_{root} to nodes $E, 10, F, 12, G, 14, H$. As a child knows the secret key sk_i for each of its ancestors, B will only have to perform three encryptions, one for each secret. He encrypts ps'_2 under pk_A, ps'_4 under pk_6 and ps'_{root} under pk_{12}.
From ps'_2 (respectively ps'_4), A (resp. C) shall be able to compute the root secret. We recall that from this root secret is derived a new epoch secret S_{E+1}. Before sending his Commit, B computes S_{E+1} and uses it to produce a confirmation key. This value shall enable A and C to check that they have derived the correct root secret and so, that they received a correct update. Other mechanisms such as the transmission of the updated view of the tree, or of intermediate hash values are provided for a user to check that he received a correct update. However, all those mechanisms enable a verification after receiving the update information.

From there, a malicious user can send non valid updates (to everyone or any branch) causing a denial of service. Given the mechanisms for a user to check whether the update is valid, there are two options: either the update is accepted by all only once each user has confirmed that he received a valid update, this can imply a huge latency, if some users are seldom online. Or the update is validated without such a feedback approach. In such a case, the users that received non valid secret values are ejected from the group *de facto*. In both case, this seriously hampers with the security of the service provided by the protocol.

3.2 Securing MLS updates

We now explain how to combine a ZK protocol and a verifiable encryption to secure the update process in MLS. We first focus on a single step of the update process (a user updates his direct parent) and then explain how this solution can be extended to the global tree.

Server-checking in MLS. As described in Figure 2, let assume that B generates a new secret ps'_B and computes:

- $\mathsf{deriveKeyPair}(ps'_B)$ to obtain a new key package;
- $ps'_2 = \mathsf{H}_p(ps'_B)$ the new secret for node 2;
- $(sk'_2, pk'_2) = \mathsf{deriveKeyPair}(ps'_2)$ the new keys for node 2;

Finally he sends $(\mathsf{Enc}_{pk_A}(ps'_2),\ pk'_2)$ so that everyone gets pk'_2 but only A can access ps'_2. Now suppose there exists a ZK protocol which, given public values C_x and C_y, provides the following proof: $\mathsf{PK}\{x, r_x, r_y : C_x = \mathsf{Com}(x, r_x) \wedge C_y = \mathsf{Com}(f(x), r_y)\}$ for any PRF f. Then B can send to the server the public values C_B, C_2, $C_{sk'_2}$, pk'_2 together with a proof $\Pi_2 = \mathsf{PK}\{ps'_B, r_{B'}, r_2, r_{sk'_2} : C_B = \mathsf{Com}(ps'_B, r_{B'}) \wedge C_2 = \mathsf{Com}(\mathsf{H}_p(ps'_B), r_2) \wedge C_{sk'_2} = \mathsf{Com}(\mathsf{deriveSK}(\mathsf{H}_p(ps'_B)), r_{sk'_2}) \wedge pk'_2 = \mathsf{deriveSK}(\mathsf{H}_p(ps'_B))P\}$ (the last part of the proof being a classic discrete log proof). In addition, verifiable encryption (detailed in section 2) allows to link the message encrypted with $\mathsf{VerifEnc}$ with the data committed in C_2. To sum up, B will send for a node update, the public values C_B, C_2, $C_{sk'_2}$, and pk'_2, the proof Π_2 together with $\mathsf{VerifEnc}_{\mathsf{Enc},pk_A}(ps'_2)$. If the server accepts the proof, then he transmits the public key pk'_2 as well as $\mathsf{VerifEnc}_{\mathsf{Enc},pk_A}(ps'_2)$ to A.
To extend the proof to the complete tree, one has to repeat the above steps for each level. To certify the update value ps'_4 corresponding to the parent node 4, B will send the server values C_4, $C_{sk'_4}$, pk'_4, the proof $\Pi_4 = PK\{ps'_2, r_2, r_4, r_{sk'_4}$: $C_2 = \mathsf{Com}((ps'_2), r_2) \wedge C_4 = \mathsf{Com}(\mathsf{H}_p(ps'_2), r_4) \wedge C_{sk_4} = \mathsf{Com}(\mathsf{deriveSK}(\mathsf{H}_p(ps'_2)), r_{sk'_4}) \wedge pk'_4 = \mathsf{deriveSK}(\mathsf{H}_p(ps'_2))P\}$ together with $\mathsf{VerifEnc}_{\mathsf{Enc},pk_6}(ps'_4)$. The crucial point is that, as the commitment C_2 is linked with ps'_B in Π_2, it can be used as a base value for Π_4 and so on. Some special care must be taken as we commit, in a group of order q prime, to an element $sk \in \{0,1\}^{256}$ that does not lie naturally in $\mathbb{Z}/q\mathbb{Z}$. We explain how to handle with this in Appendix C.

About the server. Several reasons appear for calling on a third party. Firstly, this central node with the largest computational power is the one that can discard

invalid updates with the most efficiency. If one relies on users to check for the validity of the data they received, this means that one must wait for each user to process the update and to send back an acknowledgement. As a user can be off-line for a long time, this can be very inefficient. Another solution would be to allow users to adopt the update as soon as they are individually convinced it is correct, while providing a "backup solution". This would probably imply keeping old keys and drastically impoverish FS.
Secondly, in MLS architecture, all the update encrypted messages are gathered and sent as one big message to all the users. It may be of interest to think of a solution where only the needed encryption is sent to a specific user. In this case, only the server will see all the messages together. He is then the only one able to perform a verification on a global proof to see whether all the updates are correctly generated from a single secret seed.

4 ZK for a PRF on committed input and output

In this section, we provide two protocols to prove the knowledge of an input x and randoms r_x, r_y, such that, for a public values C_x, C_y, and a function f evaluated as a circuit, $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(f(x), r_y)$. This goal can be written as an ideal functionality, as in Figure 3.

The *Verifier* inputs C_y, C_x. The *Prover* inputs values x, r_x, r_y. The functionality outputs accept if (x, r_x) opens C_x and $(f(x), r_y)$ opens C_y.

Fig. 3. The ideal functionality $\mathcal{F}_{f,\mathsf{Com}}$.

The efficient ZK proofs for a function evaluation require this function to be evaluated as a circuit, but efficient commitments are algebraic. Consequently, we want to achieve the best of both worlds by combining a proof on a circuit and algebraic commitments.
Our first solution, `CopraZK` (Commitment and PRF alternative ZK), is specific to the case of f being a PRF. The secret x is the PRF key and we evaluate f on a public message m. We consider the circuit that evaluates two equations on x and on another secret input a. We call the results of these equations tag values. The first tag t_1 only depends on $f(x, m)$ and $f(a, m)$. The second tag t_2 depends on x and a. Calling PRF properties, we show that the pair of equations have a single solution. Hence, the input pair (x, a) is bound to the tag pair t_1, t_2. In a second step, we call the homomorphic properties of the commitment to show that the values committed in C_x and C_y also verify the equations. Then it must be that the values committed to corresponds to the values used in the circuit.
Our second solution, `ComInOutZK` (Committed Input and Output ZK) is directly inspired from [6], which provides a proof of a circuit evaluation on a committed input and public output. We extend their work to a committed output. The idea is to commit to the bits of the output and, for each round, to the output shares

given by the circuit decomposition of ZKBoo. Then, calling the homomorphic properties of the commitment scheme, one can bind the bit wise commitments to the share commitments by revealing the difference of randomness between those elements.

`CopraZK` adds a negligible number of algebraic operations. The prover performs around 20 computations on the curve (public key operations) and 8 additions in $\mathbb{Z}/q\mathbb{Z}$ (symmetric operations). For the verifier, 12 additions on the curve and 2 in $\mathbb{Z}/q\mathbb{Z}$ are needed. However, the circuit part of the proof is more than doubled to compute the two tags. Considering ZKBoo, the prover effort is $\mathcal{O}(\sigma|F|)$ symmetric operations, where $|F|$ is the number of AND gates of the circuit and σ the number of rounds. Our solution requires $\mathcal{O}(\sigma(2|F| + |mod|)) + 8$ symmetric operations and 20 public key operations, where $|mod|$ is the size of the circuit for a modular addition. We show in section 5 that $|mod|$ is negligible compared to $|F|$ and finally, our solution requires on the prover side ($\mathcal{O}(2\sigma|F|)$ symmetric + 20 public key operations. The computational cost is dominated by the symmetric part. The size of the proof and the work on the verifier's side are also dominated by the circuit part. One inconvenient is that it is limited to PRF evaluation and that the security proof requires non usual hypothesis on PRF.
On the opposite side, `ComInOutZK` is valid for any circuit, only requires equivocality of the commitment scheme, which is a common hypothesis, and leaves the circuit evaluation untouched. But it requires a non negligible number of algebraic commitments. Considering $|x|$ (respectively $|y|$) the bit size of the input (of the output), we obtain on the prover side $\mathcal{O}(|x| + |y| + 2\sigma)$ public key operations and $\mathcal{O}(\sigma|F|)$ symmetric operations. The verifier's work is equivalent. The proof size of ZKBoo is augmented with $\mathcal{O}(|x| + |y| + 6\sigma)$ curve points which is asymptotically $\mathcal{O}((|x| + |y| + \lambda)\lambda)$ as σ augments with λ. We compare in Table 1 our two solutions with the SNARK based solution of [1].

	Non interactive	No CRS	Prover's work	Verifier's work	Proof size
SNARK based [1]	yes	no	$\mathcal{O}((\|F\| + \lambda) \cdot pub)$	$\mathcal{O}((\|x\| + \|y\| + \lambda) \cdot pub)$	λ
`CopraZK`	yes	yes	$\mathcal{O}(2\|F\|\lambda \cdot sym)$	$\mathcal{O}(2\|F\|\lambda \cdot sym)$	$\mathcal{O}(2\|F\|\lambda)$
`ComInOutZK`	yes	yes	$\mathcal{O}(\|F\|\lambda \cdot sym + (\|x\| + \|y\| + \lambda) \cdot pub)$	$\mathcal{O}(\|F\|\lambda \cdot sym + (\|x\| + \|y\| + \lambda) \cdot pub)$	$\mathcal{O}((\|F\| + \|x\| + \|y\| + \lambda)\lambda)$

Table 1. Comparison of the efficiency of the different solution for Circuit proof on committed input and output. *pub* stands for the cost of a public key operation (multiplication and addition on the curve for instance), while *sym* stands for the cost of a symmetric operation. $|F|$ is the circuit size, $|x|$ the input size and $|y|$ the output size. In most applications, $|F| >> (|x|, |y|, \lambda)$.

On the challenge size When we expose our solutions, in both case we mention a unique challenge, that is used for the algebraic Σ-protocol and for the ZKBoo proof. This means that the challenge space size for the Σ-protocol is 3 and that we shall perform $\lambda/3$ rounds to obtain a soundness error in $2^{-\lambda}$. The σ-protocol can benefit from a larger challenge space, that allows for a single round. As explained in [6], it is possible to define distinct challenges $e_\rho \in \{1, 2, 3\}$ for each ZKBoo round and a global challenge $e = \sum_{i=1}^{\sigma} 3^i e_i$ for the algebraic Σ-protocols, hence the algebraic part of the proof can be performed a single time.

4.1 Our first solution: `CopraZK`

Let us denote by $\mathsf{Func}(\mathcal{D}, \mathcal{R})$ the set of all functions from $\mathcal{D}$ to $\mathcal{R}$ and by $\mathsf{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ the set of all function families with parameter (key) in $\mathcal{K}$, domain $\mathcal{D}$ and range $\mathcal{R}$. We write $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ for a function family in $\mathsf{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ (and call it a function, by ease of language). Let f be a function: $\mathbb{Z}_2^\ell \times \mathbb{Z}_2^* \to \mathbb{Z}_2^\ell$ and m a public input, $m \in \mathbb{Z}_2^*$. Let $\mathbb{G}$ be a group of prime order q, such that $2^\ell \leq q$. There is a natural embedding $\mathbb{Z}_2^\ell \hookrightarrow \mathbb{G}$. Let P be a generator for this group and Q an element of $\mathbb{G}$ such that $\log_P(Q)$ is unknown. Let h be a hash function $\mathbb{Z}_2^* \to \mathbb{Z}_2^{\ell *}$. Let Com be the Pedersen commitment scheme. Those elements are the public parameters of the prover and the verifier.
We use a ZKBoo proof for the circuit part, but the protocol and its proof are valid for any circuit based ZK proof or argument. Let C_x, C_y be public commitments, known to the verifier. The main idea is to consider the circuit two tags $t_1 = f(x, m) + f(a, m)$ and $t_2 = x + a$ where a is a random mask considered as a second secret entry of the circuit. A MPC in the head proof on this circuit ensures that t_1 and t_2 are correctly computed from two secret values x and a known to the prover. The prover also provides commitments C_a and C_b for values a and $f(a, m)$. Considering Pedersen commitments, we complete the circuit proof with an algebraic proof that the committed values in C_a, C_x, C_y, C_b verify the relations t_1 and t_2. These linear relations together with the properties of f defined below, bind the values of C_x and C_y such that the verifier can be convinced that the value committed in C_y is equal to the evaluation of f on the value committed in C_x. The detailed description of the protocol `CopraZK` is given in Figure 4.
The following theorem states the security of `CopraZK`.

Theorem 1. *Let f be a secure special pseudo-random function, h be a secure hash function and Com a homomorphic commitment scheme. Then the* `CopraZK` *protocol described in Figure 4 defines a ZK argument with computational Zero-Knowledge.*

4.2 `ComInOutZK`: a bit-wise solution

In [6], the authors propose a non interactive proof $PK\{x : C_x = Com(x, r_x) \wedge y = f(x)\}$ based on bit commitments and ZKBoo++. Their optimized solution increases the ZKBoo++ prover's and verifier's work with $\mathcal{O}(|x| + \sigma)$ exponentiations and multiplications on the group $\mathbb{G}$ of order q chosen for the commitment,

Let C_y, C_x be public commitments and m a public message.
The *Prover* wants to convince the *Verifier* that he knows x, r_x, r_y
such that $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(f(x,m), r_y)$.

Prover
Commit phase:
1. samples $a, r_a, r_b \leftarrow\!\!\$\ \mathbb{Z}_2^\ell$.
2. computes $C_a = \mathsf{Com}(a : r_a)$, $C_b = \mathsf{Com}(f(a,m) : r_b)$.
3. computes $\alpha = \mathsf{h}(C_x || C_y || C_a || C_b)$.
4. computes $(t_1, t_2) = (f(x,m) - \alpha f(a,m) \bmod q, x - \alpha a \bmod q)$.
5. Evaluates the commit phase output a_Π for the Σ protocol
   $\Pi = PK\{x, r_x, y, r_y, a, r_a, b, r_b : C_x = xP + r_xQ$
   $\wedge\, C_y = yP + r_yQ \wedge C_a = aP + r_aQ \wedge C_b = bP + r_bQ$
   $\wedge\, t_1 = y - \alpha b \wedge t_2 = x - \alpha a\}$

for $\rho \in [1, \sigma]$ (ZKBoo part):
6. samples random tapes $k_1^\rho, k_2^\rho, k_3^\rho$.
7. generates $x_1^\rho, x_2^\rho, x_3^\rho = \mathsf{Share}(x, k_1^\rho, k_2^\rho)$ s.t. $x = x_1^\rho \oplus x_2^\rho \oplus x_3^\rho$.
8. evaluates the MPC protocol on the circuit Circ: on (x, a),
   $(t_1, t_2) = (f(x,m) - \alpha f(a,m) \bmod q, x - \alpha a \bmod q)$, and obtains three views $w_1^\rho, w_2^\rho, w_3^\rho$.
9. obtains the output shares : $o_1^\rho = (t_{1,1}, t_{2,1}), o_2^\rho = (t_{1,2}, t_{2,2})$,
   $o_3^\rho = (t_{1,3}, t_{2,3})$ s.t. $t_1 = t_{1,1}^\rho \oplus t_{1,2}^\rho \oplus t_{1,3}^\rho$, $t_2 = t_{2,1}^\rho \oplus t_{2,2}^\rho \oplus t_{2,3}^\rho$.
10. commits to the views: $c_1^\rho = \mathsf{h}(w_1^\rho, k_1^\rho), c_2^\rho = \mathsf{h}(w_2^\rho, k_2^\rho)$, and $c_3^\rho = \mathsf{h}(w_3^\rho, k_3^\rho)$.

$a = C_a, C_b, C_x, t_1, t_2, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$.

Challenge: $e = h(a)$
Response phase:
1. computes the response z_Π for the proof Π

for $\rho \in [1, \sigma]$:
2. $b^\rho = (o_{e+2}^\rho = (t_{1,e+2}, t_{2,e+2}), c_{e+2}^\rho)$
3. $z^\rho = (w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho)$

return $p = (e, (b^\rho, z^\rho)_\rho, z_\Pi)$

Verifier(a, p)
1. Parse p as e, $(b^\rho, z^\rho)_\rho, z_\Pi$
2. Parse a as $C_a, C_b, C_x, t_1, t_2, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$
3. Computes α
4. Reconstruct the proof Π

for $\rho \in [1, \sigma]$ (Verifying the ZKBoo proof):
5. runs the MPC protocol to reconstruct w_e^ρ from $w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho$
6. gets $t_{1,e}^\rho, t_{2,e}^\rho = \mathsf{Output}(w_e)$, $t_{1,e+1}^\rho, t_{2,e+1}^\rho = \mathsf{Output}(w_{e+1})$
7. computes $t_{1,e+2}^\rho = t_1 \oplus t_{1,e}^\rho \oplus t_{1,e+1}^\rho$, $t_{2,e+2}^\rho = t_2 \oplus t_{2,e}^\rho \oplus t_{2,e+1}^\rho$

Reconstruct a and reject if $e \neq h(a)$

Fig. 4. The `CopraZK` protocol. The reconstruction of the algebraic proof means the verification by reconstruction of the challenge in the Fiat-Shamir version.

where $|x|$ is the number of bits of the input x and σ is the number of rounds in ZKBoo++. The proof size grows by $\mathcal{O}(|x| + \sigma)$ group elements and $\mathcal{O}(|x| + \sigma)$ elements in $\mathbb{Z}/q\mathbb{Z}$. We adapt this strategy in the case of a committed output. As the output of the circuit, y, shall remain secret, we will not be able to call ZKBoo++ as a full black box. This is of prime importance when we prove the zero-knowledge property. Compared to `CopraZK`, the bit-wise commitment strategy does not require to augment the circuit with a second evaluation of f. Another advantage is that we do not require specific hypothesis on f. As a drawback, we add $\mathcal{O}(|x| + |y| + 2\sigma)$ algebraic operations to the basic ZKBoo++ proof.

The work of Backes *et al.* and our extension rely on a result given by the homomorphic property of a commitment scheme such as Pedersen scheme. For any scalar k, and any two commitments $\mathsf{Com}(x, r_x)$, $\mathsf{Com}(y, r_y)$, $k \cdot \mathsf{Com}(x, r_x) + \mathsf{Com}(y, r_y) = \mathsf{Com}(kx + y, kr_x + r_y)$. For any commitment $C_b = \mathsf{Com}(b, r_b)$ to a secret bit b and any public bit β, one can easily compute the commitment of $b \oplus \beta$ as follows: if $\beta = 0$, $C_{b\oplus\beta} = C_b$ and if $\beta = 1$ then $C_{b\oplus\beta} = \mathsf{Com}(1, 0) - C_b = \mathsf{Com}(1 - b, -r_b)$. For any $x = \sum_{i=0}^{|x|-1} 2^i x[i]$, denote $C_{x[i]} = Com(x[i], r_{x[i]})$ a commitment to the i-th bit of x. Then $\sum_{i=0}^{|x|-1} 2^i C_{x[i]}$ is a valid commitment to x with opening randomness $\sum_{i=0}^{|x|-1} 2^i r_{x[i]}$. And one can easily compute a commitment to $x \oplus \beta$ for an element β as $C_{x\oplus\beta} = \sum_{i=0}^{|x|-1} 2^i C_{x[i]\oplus\beta[i]}$, with opening randomness $\sum_{i=0}^{|x|-1} 2^i (-1)^{\beta[i]} r_{x[i]}$.

We describe in Figure 5 the protocol on committed output only, for readability reasons. Combining Backes *et al.* protocol for committed input and ours for committed output leads to the functionality described in Figure 3.

The public parameters are: f a function from $\mathbb{Z}_2^\ell$ to $\mathbb{Z}_2^\ell$ evaluated as a circuit (not necessarily PRF), $\mathbb{G}$ is a prime order q group, such that $2^\ell \leq p$ with a natural embedding $\mathbb{Z}_2^\ell \to \mathbb{G}$, and P be a generator of $\mathbb{G}$ and $Q \in \mathbb{G}$ such that $\log_P(Q)$ is unknown. We consider a hash function $\mathsf{h} : \mathbb{Z}_2^* \to \mathbb{Z}_2^{\ell *}$ and Com a Pedersen commitment scheme.

Theorem 2. *Given that ZKBoo and the Π_j are Σ protocols with 3-special soundness and honest verifier Zero Knowledge property, and Com is a homomorphic and equivocal commitment scheme, then the protocol described in Figure 5 is a Σ-protocol with 3-special soundness and honest verifier property.*

5 Implementation

We now discuss the efficiency of our protocol `CopraZK`, both in terms of time and size of the proof. We focused on `CopraZK` because it was more prone to benefit from optimized MPC protocols such as KKW ([33]), that are designed for large circuits. We detail the results we obtained from a simple implementation of `CopraZK` on top of the existing ZKBoo code. We remind that the efficiency of the MPC in the head protocols is directly related to the number of AND gates in the circuit. In fact, in ZKBoo, ZKBoo++ and KKW, only AND gates

The *Prover* knows x, $y = f(x)$, and r_y such that $C_y = \mathsf{Com}(f(x), r_y)$. The *Verifier* knows the statement C_y.

Prover

Commit phase

1. samples random $r_{y[j]}$ and commits to the bits of y : $C_{y[j]} = \mathsf{Com}(y[j], r_{y[j]})$ for $j \in [0, |y|]$.
2. computes the commit phase a_{Π_j} for the proofs $\Pi_j = PK\{y[j], r_{y[j]} : C_{y[j]} = \mathsf{Com}(y[j], r_{y[j]}) \wedge y_[j] \in \{0,1\}\}$ for $j \in [0, |y|]$.

for $\rho \in [1, \sigma]$:

3. samples random seeds k_1^ρ, k_2^ρ, k_3^ρ.
4. generates the shares $x_1^\rho, x_2^\rho, x_3^\rho = \mathsf{Share}(x, k_1^\rho, k_2^\rho)$ such that $x = x_1^\rho \oplus x_2^\rho \oplus x_3^\rho$.
5. simulates the MPC to obtain three views $w_1^\rho, w_2^\rho, w_3^\rho$.
6. evaluates $y_i^\rho = \mathsf{Output}(w_i^\rho)$, $i \in \{1,2,3\}$.
7. commits to the views : $c_i^\rho = \mathsf{h}(w_i^\rho, k_i^\rho)$, $i \in \{1,2,3\}$.
8. samples random $r_{y_i^\rho}$ and commits to the outputs : $C_{y_i^\rho} = \mathsf{Com}(y_i^\rho, r_{y_i^\rho})$, $i \in \{1,2,3\}$.

$a = ((C_{y_1^\rho}, C_{y_2^\rho}, C_{y_3^\rho}, c_1^\rho, c_2^\rho, c_3^\rho)_\sigma, (C_{y[j]})_{|y|}, (a_{\Pi_j})_{|y|})$

Challenge : $e = \mathsf{h}(a)$

Response phase

1. computes the responses z_{Π_j} for the proofs Π_j

for $\rho \in [1, \sigma]$:

2. $b^\rho = (C_{y_{e+2}^\rho}, c_{e+2}^\rho)$
3. $z^\rho = (w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho, r_{y_e^\rho}, r_{y_{e+1}^\rho})$
4. $\beta^\rho = y_e^\rho \oplus y_{e+1}^\rho$
5. $C_z^\rho = \sum_{i=0}^{|y|-1} 2^i C_{y[i] \oplus \beta^\rho[i]}$
6. $r_z^\rho = r_{y_{e+2}^\rho} - \sum_{i=0}^{|y|-1} 2^i (-1)^{\beta[i]} r_{y[i]}$

return $p = (e, (b^\rho, z^\rho, r_z^\rho)_\rho), (z_{\Pi_j})_j$

. . . . . .

Verifier(a, p)

1. Parse p as e, $(b^\rho, z^\rho, r_z^\rho)_\sigma$
2. Parse a as $(C_{y_1^\rho}, C_{y_2^\rho}, C_{y_3^\rho}, c_1^\rho, c_2^\rho, c_3^\rho)_\sigma, (C_{y[j]})_|y|, a_{\Pi_j})$
3. Reconstruct the proof Π_j
4. Reject if $C_y \neq \sum_{i=0}^{|y|-1} 2^i C_{y[i]}$

for $\rho \in [1, \sigma]$:

5. runs the MPC protocol to reconstruct w_e^ρ from $w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho$
6. obtains $y_e^\rho = \mathsf{Output}(w_e)$, $y_{e+1}^\rho = \mathsf{Output}(w_{e+1})$
7. Computes $\beta^\rho = y_e \oplus y_{e+1}$
8. Computes $C_z^\rho = \sum_{i=0}^{|y|-1} 2^i C_{y[i] \oplus \beta^\rho[i]}$
9. Reject if $C_{y_{e+2}^\rho} \neq \mathsf{Com}(0, r_z) + C_z^\rho$

Reconstruct a and reject if $e \neq h(a)$

Fig. 5. Our second protocol. When the *Verifier* reconstructs the challenge in the final step, he computes the commitments and can check that their openings were correct.

are randomized to provide the Zero-Knowledge property. Hence, the verifier can compute the other gates by himself but he needs the output of the AND gates

to recompose a complete view and check the consistency of the proof. The views sent by the prover then only contain the output of the AND gates.

Our implementation. We implemented the circuit part of our `CopraZK` protocol on top of the ZKBoo code, available at `https://github.com/Sobuno/ZKBoo`. This code provides ZKBoo versions for elementary operations. These functions operate on three views and each binary AND call is randomized as recommended in the ZKBoo description. The code provides ZKBoo versions of operations on 32 bits vectors: addition, XOR, AND, addition with a constant. They also provide a ZKBoo version of a SHA_256 circuit, that comprises around 23300 binary AND gates. We implemented a ZKBoo version for the HMAC function, with two calls to SHA_256, and a 256-bit addition. The function HMAC corresponds to HKDF − expand when the desired output length equals the output length of the underlying hash function. Our final circuit, with input x and a computes $t_1 = x + a$ and $t_2 = \mathsf{HMAC}(x) + \mathsf{HMAC}(a)$ for a total of 93696 AND gates. We did not implement the modular reduction. However, as we expect our entries x and a (similarly $\mathsf{HMAC}(x)$ and $\mathsf{HMAC}(a)$) to be in the cyclic group $\mathbb{Z}/q\mathbb{Z}$ (using solutions described in Appendix C), t_1 and t_2 may only exceed q by one q. Hence modular reduction can be instantiated as a comparison and subtraction if necessary. Using Cingulata (a compiler toolchain for homomorphic encryption, available at `https://github.com/CEA-LIST/Cingulata`), we estimated the number of AND gates for this operation on 8 and 16 bits integers, and obtained respectively 48 and 103 AND gates. From this result, we can expect that a modular reduction on 256 bits integer can be implemented using around 2000 AND gates. This number being far from representative in our circuit, we did not considered this operation in a basic implementation. We consider the running time for a soundness parameter $\sigma = 80$ (corresponding to a soundness error of 2^{-80}), requiring 136 rounds.

Prover (ms)		Verifier (ms)	
Generating random	21	Loading file	1
Sharing secrets	1	Generating challenge	0
Running circuit	534	Verifying	799
Committing	20		
Total generating proof data	578	Total verifying	800
Proof size (MB)	3.3		

Table 2. Running meantime of the *Prover* and the *Verifier* over 1 000 executions for 136 rounds.

Our tests were run on a Dell laptop with Processor IntelCore i7-7600U CPU running a single core at $2,8$ Ghz with 15.5 GB of RAM. Results are given

in Table 2. We see that the prover's running time is better than the verifier's one, which is not the case in [25], running ZKBoo on the mere SHA_256 circuit. Running a proof on a bigger circuit increases the verifier's load more than the prover's one. The running time of the prover is around half a second when the verification takes around 0.8 seconds. These results are better than what can be expected from SNARKs, as we explain in Appendix F. In [25], the authors show that a parallelized implementation can seriously improve those results (with 8 threads, they divide the running time by 3.4 for the prover and by 5 for the verifier). This experimental proof size is larger but in Appendix F that it can be improved with optimized MPC in the head protocols.

6 Conclusion

In this work we provide a concrete solution to a practical problem that appears in the MLS specification. We describe how existing cryptographic tools such as ZK proofs and verifiable encryption can be combined to secure the update process. As the regular update of the group secret is the key to obtain the FS and PCS properties, we think our solution may be of interest.
Additionally, we propose two protocols to obtain ZK proofs on circuit with committed input and output, such that our improvement proposal for MLS is settled on protocols as efficient as possible. Hence, an interesting way for future work is in the optimization of the verifiable encryption. The CL framework, introduced by Castagnos and Laguillaumie in [15] and enriched with Zero-Knowledge properties in [14], that considers a cyclic group $\mathbb{G}$ where the DDH assumption holds together with a subgroup F of $\mathbb{G}$ where the discrete logarithm problem is easy, may provide novel and efficient solutions.

References

1. Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 643–673. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0˙22
2. Alwen, J., Capretto, M., Cueto, M., Kamath, C., Klein, K., Markov, I., Pascual-Perez, G., Pietrzak, K., Walter, M., Yeo, M.: Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement. Cryptology ePrint Archive, Report 2019/1489 (2019), https://eprint.iacr.org/2019/1489
3. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2˙5
4. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 248–277. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2˙9

5. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134104
6. Backes, M., Hanzlik, L., Herzberg, A., Kate, A., Pryvalov, I.: Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: Lin, D., Sako, K. (eds.) PKC 2019, Part I. LNCS, vol. 11442, pp. 286–313. Springer, Heidelberg (Apr 2019). https://doi.org/10.1007/978-3-030-17253-4˙10
7. Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The Messaging Layer Security (MLS) Protocol
8. Barnes, R., Bhargavan, K., Lipp, B., Wood, C.: Hybrid Public Key Encryption
9. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9˙31
10. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 619–650. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9˙21
11. Bhargavan, K., Barnes, R., Rescorla, E.: Treekem: Asynchronous decentralized key management for large dynamic groups
12. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3˙25
13. Camenish, J., Stadler, M.: Proof systems for general statements about discrete logarithms
14. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 191–221. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8˙7
15. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 487–505. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-319-16715-2˙26
16. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133997
17. Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 499–530. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53015-3˙18
18. Chase, M., Perrin, T., Zaverucha, G.: The Signal private group system and anonymous credentials supporting efficient verifiable encryption. Cryptology ePrint Archive, Report 2019/1416 (2019), https://eprint.iacr.org/2019/1416
19. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. Cryptology ePrint Archive, Report 2017/666 (2017), http://eprint.iacr.org/2017/666

20. Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 451–466 (2017)
21. Damgård, I.: On sigma protocols
22. D.J.Bernstein: A State-of-the-art Diffie Hellman Function, `https://cr.yp.to/ecdh.html`
23. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7˙12
24. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9˙37
25. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 1069–1083. USENIX Association (Aug 2016)
26. Goyal, V., O'Neill, A., Rao, V.: Correlated-input secure hash functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 182–200. Springer, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19571-6˙12
27. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8˙19
28. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4˙9
29. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
30. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 33–62. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1˙2
31. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516662
32. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. Cryptology ePrint Archive, Report 2018/954 (2018), `https://eprint.iacr.org/2018/954`
33. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243805
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992). https://doi.org/10.1145/129712.129782
35. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7˙34

36. Marlinspike, M., Perrin, T.: The double ratchet algorithm. Signal's web site (2016)
37. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press (May 2013). https://doi.org/10.1109/SP.2013.47
38. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1˙9
39. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 3–32. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1˙1
40. Stadler, M.: Publicly verifiable secret sharing. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 190–199. Springer, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9˙17

A ZKBoo

We recall here the construction of the protocol ZKBoo. A (2,3)-decomposition is a set of functions that separates the evaluation of a function into three symmetric parts such that, given any two parts, nothing is revealed about the third one, and so on the input.

Definition 1 ((2,3)-decomposition). *A (2,3)-decomposition for the function ϕ is a set of functions $D = (\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}) \cup \mathcal{F}$, such that:*

- Share *is an onto function that splits the input x into 3 shares;*
- $\mathcal{F}$ *is a finite family of efficiently computable functions described as $\{\phi_1^j, \phi_2^j, \phi_3^j\}_{j\in[1,N]}$;*
- Output_i *computes a value y_i called the output share;*
- Rec *computes the final value $y = \phi(x)$ from y_1, y_2 and y_3.*

A (2,3)-decomposition produces three distinct views w_e, each composed of an input share x_e and the output values of the corresponding intermediate functions ϕ_e^j's. Two properties are required from a decomposition: correctness and 2-privacy. The first means that the decomposition allows to correctly evaluate the function. The second guarantees that given any two views, one cannot learn the secret input x. ZKBoo is built over the decomposition. Briefly, the prover commits to the views and only reveals two of them. A soundness error of $2^{-\sigma}$, *i.e.* the probability for a cheating prover not to be caught is less than $2^{-\sigma}$, is obtained by repeating the process $t = \sigma/(\log_2 3 - 1)$ times. The size of the proof is essentially the size of the 2 views, thus depends on the size of the circuit. ZKBoo++ ([16]) improves the original protocol by cutting by half the size of the proof. The authors obtain such a result by avoiding sending in the proof any value that the verifier can compute himself. Figure 6 gives a detailed description of the improved version ZKBoo++.

The *Prover* knows x, such that $y = f(x)$. The *Verifier* knows the statement y.
Prover
Commit phase
1. samples random seeds k_1, k_2, k_3.
2. generates the shares $x_1, x_2, x_3 = \mathsf{Share}(x, k_1, k_2)$ such that $x = x_1 \oplus x_2 \oplus x_3$.
3. simulates the MPC to obtain three views w_1, w_2, w_3.
4. evaluates $y_i = \mathsf{Output}(w_i)$, $i \in \{1, 2, 3\}$.
5. commits to the views : $c_i = \mathsf{h}(w_i, k_i)$, $i \in \{1, 2, 3\}$.

$a = (y_1, y_2, y_3, c_1, c_2, c_3)$

Challenge : $e = \mathsf{h}(a)$ interpreted in $\{1, 2, 3\}$

Response phase
1. $b = (y_{e+2}, c_{e+2})$
2. $z = (w_{e+1}, k_e, k_{e+1})$

return $p = (e, (b, z))$

Verifier(a, p)
1. Parse p as e, (b, z)
2. Parse a as $(y_1, y_2, y_3, c_1, c_2, c_3)$
3. runs the MPC protocol to reconstruct w_e from w_{e+1}, k_e, k_{e+1}
4. obtains $y_e = \mathsf{Output}(w_e)$, $y_{e+1} = \mathsf{Output}(w_{e+1})$
5. Computes $y_{e+2} = y_e \oplus y_{e+1}$

Reconstruct a and reject if $e \neq h(a)$

Fig. 6. The protocol ZKBoo++. Share and Output are functions specific to the (2,3)-decomposition defined in ZKBoo. (We omit the index for the Output.)

Proposition 1 ([25]). *The ZKBoo protocol is a Σ-protocol for the relation $\mathcal{R}_\phi$, with 3-special soundness.*

In [33], a more efficient solution is proposed by considering a MPC solution with n virtual participants, instead of 3 in ZKBoo. The soundness error is also better for one round so that, less rounds are needed to reach the desired security level.

B Verifiable encryption

We recall here the formal definition of verifiable encryption as detailed in [12]. Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a probabilistic public key encryption scheme and $(pk, sk) = \mathsf{KeyGen}(1^\lambda)$ a valid key pair. The verifiable encryption mechanism, attached to an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, to the binary relation $\mathcal{R}$ and to the associated language $L_\mathcal{R} = \{(x, w) : \mathcal{R}(x, w) = 1\}$, is defined as a two-party protocol Π between a prover P (who encrypts the data) and a couple composed of a verifier V on the one hand and a recovery algorithm Rec on the other hand. The protocol Π takes as public parameters a valid public key pk, a statement x and a security parameter λ. Let $V_P(pk, x, \lambda)$ denote the final out-

put of V interacting with P on input (pk, x, λ). The recovery algorithm takes as input the secret key sk and $V_P(pk, x, \lambda)$.

Definition 2 (Secure Verifiable encryption). *The couple protocol/recovery algorithm described above is a secure verifiable encryption scheme if the following holds:*

- *completeness: if P and V are honest, then $V_P(pk, x, \lambda) \neq \perp$ for all (pk, sk) valid key pair for the subsequent encryption scheme and $x \in L_{\mathcal{R}}$;*
- *validity: for all PPT malicious Prover $\tilde{P}$, all valid key pairs (sk, pk), $\Pr[\mathcal{R}(x, \mathsf{Rec}(sk, V_P(pk, x, \lambda))] \neq 1$ and $V_P(pk, x, \lambda) \neq \perp\big)$ is negligible;*
- *computational Zero-Knowledge: for every unbounded malicious Verifier $\tilde{V}$, there exists an expected poly-time Simulator $\mathcal{S}im_{\tilde{V}}$ with black-box access to $\tilde{V}$ such that for all distinguisher A, all positive polynomial $p(\cdot)$, all $x \in \mathcal{L}$ and all sufficiently large λ we have:*

$$\Pr[A(pk, x, \alpha_i] = i : \ (pk, sk) = \mathsf{KeyGen}(1^\lambda), \alpha_0 = V_P(pk, x, \lambda), \\ \alpha_1 = \mathcal{S}im_{\tilde{V}}(pk, x, \lambda), i \in \{0, 1\}\big) \leq \tfrac{1}{2} + \tfrac{1}{p(\lambda)}.$$

Informally, validity ensures that a malicious prover $\mathcal{P}^*$ will always be caught, except with negligible probability, because the recovery algorithm shall be able to compute a witness. The recovery success guaranties that the decryption will be correct. In the soundness property of ZKPoK, one needs a third party, the extractor, to unmask a cheating prover. We also note that the revelation process "kills" the Zero-Knowledge feature of the proof. Verifiable encryption follows the same rules, except that the third party - the recovery algorithm - needs an additional ingredient : the secret key sk. The verifier is not supposed to be honest here.

The Camenish-Damgård verifiable encryption scheme In Figure 7, we describe the verifiable encryption solution given in [12], adapted to the Σ-protocol dedicated to a Pedersen commitment.
As for any cut-and-choose protocols, the probability that a cheating prover wins is $\frac{1}{2}$ for one round. One has to repeat the protocol σ times to obtain a cheating probability (a validity error) of $2^{-\sigma}$. The protocol described in Figure 7 can be optimized by gathering all rounds in a single one as described in the original paper, dropping to $\mathcal{O}(\log(\sigma))$ the number of encryptions to store.

C Key size and group orders in MLS updates

In [**?**], several suitable cipher suites are described. We focus on one of them for a practical example, for a 128-bit security level. This suite uses X25519 for ECDH

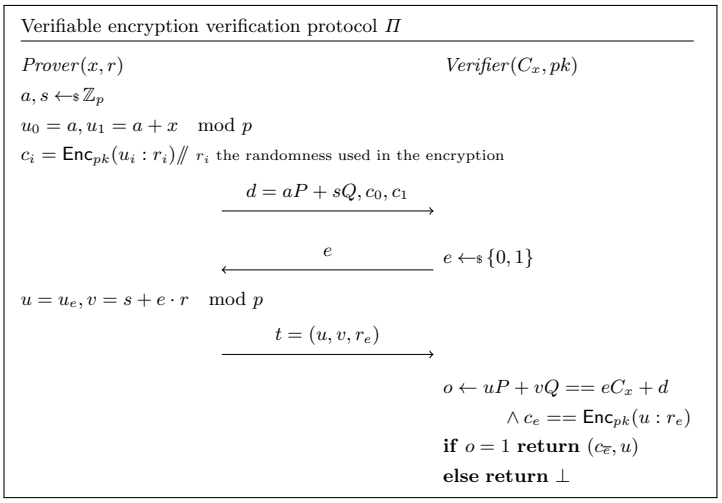

Fig. 7. The simple version of verifiable encryption scheme. The *Verifier* knows a public commitment to x, C_x and a public key pk. The *Prover* proves knowledge of the encrypted message x and of the random r used in the commitment C_x.

computation and SHA256 as a hash function (and base function for HKDF implementation). We detail hereafter the encoding used for private and public key generation for X25519. Following [22], the private key sk is obtained from a 256-bit string of secure random data $(sk[0], sk[1], \ldots, sk[255])$ by applying the following transform : $sk[0]\& = 248$, $sk[31]\& = 127$ and $sk[31]| = 64$. One obtains, when interpreted as an integer value in little endian, a scalar of the form $2^{254} + 8 \cdot \ell, \ell \in [\![0; 2^{251} - 1]\!]$. We design by $\mathsf{deriveSK}$ the application of $\mathsf{SHA256}$ followed by the above transformation such that for any 32-byte sequence of random data X, $\mathsf{deriveSK}(\mathsf{X})$ is a valid secret key for X25519. The public key is obtained by multiplying the secret key by the base point of the curve. Hence, given a 32-byte secret X, $\mathsf{DeriveKeyPair}(X) = (\mathsf{deriveSK}(X), \mathsf{deriveSK}(X)P)$. This last encoding can be integrated in the circuit computing the last derivation. We adopt this notation $\mathsf{DeriveKeyPair}(X) = (\mathsf{deriveSK}(X), \mathsf{deriveSK}(X)P)$ independently from the curve targeted.

Group order and commitments. Considering an elliptic curve E on a base field of order p, together with a base point P of prime order q, the discrete logarithm problem is supposed to be hard in the subgroup $\langle P \rangle$ generated by P. Curves are chosen such that q and p are close, but not equal. From now on, we consider commitments and discrete logarithm proofs in groups of order q. Considering X25519, a random element in $\{0, 1\}^{256}$, interpreted as an integer, will not lie naturally in $\mathbb{F}_q$. We consider that rejection sampling can provide efficiently an element in $\mathbb{F}_q$, without introducing a non negligible bias on the distribution. Let X the element out of HKDF to be committed. The method consists in cancelling the first bits of X, obtaining $\tilde{X}$ such that $\log_2(\tilde{X}) = \log_2(q) + 1$ then discarding X if $\tilde{X} > q - 1$. If the initial number of bits of X

is sufficiently big compared to $\log_2(q)$ ($\log_2(X) > \log_2(q) + 64$ as advised by the NIST for instance), then simply considering $X \mod q$ can be done without introducing a non negligible bias. For all intermediate values in the tree, one of the two above methods (depending on the curve) is available. The last step is the commitment of the secret key : $sk = \mathsf{Encode}(X)$. For this element, we directly consider the encoding provided with the curve. The commitment C_{sk} of sk in a group of order q (the group $\langle P \rangle$) will result in the same implicit reduction modulo q than the computation of the public key. Then we can produce an AND ZK proof that the value committed to in C_{sk} is the discrete log of the pk: $PK\{sk : C_{sk} = skP + rQ \wedge pk = skP\}$ where Q is an element in $\langle P \rangle$ such that its discrete log relatively to P is unknown, and r a random element in $\mathbb{Z}/q\mathbb{Z}$.

D Security definition of PRF

We detail in this Appendix the formal definitions for the security of a PRF. Our security definitions are related to the correlated robustness introduced by Ishai *et al.* [28], and to the related-key attack (RKA) scenario described by Bellare and Kohno [9]. Our notion 1-varCI-ow, formalizes the fact that the equation $eq_1 : t_1 = f(x) - \alpha f(\frac{t_2 - x}{\alpha})$ is hard to solve. This will be of prime importance in the soundness proof. The reduction to Correlated Input one-wayness, accessible in the full version (`shorturl.at/eDJMY`), states that if it is difficult to find a solution only to a simple evaluation $f(\frac{t_2 - x}{\alpha})$, then it should not be easier to find a solution to eq_1. Our 1-varRKA-wPRF notion follows the same path but on the indistinguishability side. It stipulates that the function f does not leak information when evaluated as $f(x) - \alpha f(\frac{t_2 - x}{\alpha})$. This second notion is called when proving the Zero-Knowledge property. The reduction to RKA security, also given in the full version, tells that if f does not leak any information when evaluated on values $f(\frac{t_2 - x}{\alpha})$, then it does not leak more information when evaluated as eq_1. This will be of prime importance in the proof of the Zero-Knowledge property, as the Simulator will not be able to compute t_1 and so will sample it at random. In real life, most of the widely used hash functions are build from compression functions, based on a keyed block cipher (SHA-2 family) or on an unkeyed block defined permutation (Keccak family). PRF are mostly derived from those kind of hash functions. This the case of the HKDF-expand function, built from HMAC and instantiated in MLS with a SHA family. If few literature exists concerning RKA or CI one-wayness for real-life PRF, we state that finding a preimage or correlations on inputs and outputs of a hash function is among the most difficult problem for symmetric cryptography experts.

The 1-varRKA-wPRF experiment $\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}$. Let p be a prime, let f be a public efficiently computable function $\mathcal{K} \times \mathcal{D} \to \mathcal{R}$ such that there exists an embedding $\mathcal{R} \hookrightarrow \mathbb{Z}/p\mathbb{Z}$. Let $\mathcal{A}$ be a PPT adversary. The experiment runs as follows:

SetUp. The Challenger samples a key $k \leftarrow_\$ \mathcal{K}$, a random input $r \leftarrow_\$ \mathcal{D}$ and $b \leftarrow_\$ \{0, 1\}$. He sends $\mathcal{D}$, $\mathcal{R}$ and r to $\mathcal{A}$.

Queries Computation. For $i \in [1, q]$ the adversary computes his queries $q_i = (\alpha_i, t_{1,i}) \in \mathcal{K} \times \mathcal{K}$. He sends those queries to the Challenger.

Answers. The Challenger samples $g \leftarrow_{\$} \mathsf{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$. For $i \in [1, q]$ the Challenger answers to the query q_i with a value $t_{2,i} = f(k, r) - \alpha_i f(\frac{k - t_{1,i}}{\alpha_i}, r)$ if $b = 0$, $t_{2,i} = g(k, r) - \alpha_i g(\frac{k - t_{1,i}}{\alpha_i}, r)$ if $b = 1$.

Guess. $\mathcal{A}$ outputs a guess bit $\hat{b}$. The Challenger accepts if $\hat{b} = b$.

Definition 3 (1-varRKA-wPRF security). *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. f is 1-varRKA-wPRF-secure if, for all adversary $\mathcal{A}$, running in probabilistic polynomial time t and making at most q queries, the quantity $\mathsf{Adv}^{\text{1-varrka-wprf}}_{\mathcal{A},f,p}(t, q)$ defined as:*

$$\Big|\Pr\Big[\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(t, q) = 1 | b = 0\Big] - \Pr\Big[\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(t, q) = 1 | b = 1\Big]\Big|$$

is negligible.

The 1-varCI-ow experiment $\mathsf{Exp}^{\text{1-varCI-ow}}_{\mathcal{A},\mathsf{H},p}$. Let p be a prime number, f be an efficiently computable function $\mathcal{K} \times \mathcal{D} \to \mathcal{R}$, such that there exists an embedding $\mathcal{R} \hookrightarrow \mathbb{Z}/p\mathbb{Z}$ and $\mathcal{A}$ a PPT adversary. The experiment rus as follows:

SetUp. The Challenger selects a random key k and samples a random public input r. He sends r to $\mathcal{A}$.

Queries. For $i \in [1, q]$ the adversary computes his queries $q_i = (\alpha_i, t_{1,i}) \in \mathcal{K} \times \mathcal{K}$. He sends them to the Challenger. He receives a value $t_{2,i} = f(k, r) - \alpha_i f(\frac{k - t_{1,i}}{\alpha_i}, r) \in \mathcal{R}$.

Invert. $\mathcal{A}$ sends a couple (x, j) with $x \in \mathcal{K}$ and j the index of a query. The Challenger accepts if $f(x, r) - \alpha_j f(\frac{x - t_{1,j}}{\alpha_j}, r) = t_{2,j}$.

Definition 4 (1-varCI-ow security). *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. f is said to be (q, t)-1-varCI-ow-secure if, for all adversary $\mathcal{A}$ running in PPT t and making q queries, the following advantage: $\mathsf{Adv}^{\text{1-varci-ow}}_{\mathcal{A},f,p}(t, q) = \Pr\Big[\mathsf{Exp}^{\text{1-varCI-ow}}_{\mathcal{A},f,p}(t, q) = 1\Big]$ is negligible.*

E Security of our zero-knowledge protocols

E.1 Theorem 1

We prove the security of the construction `CopraZK` construction, enunciated in Theorem 2, in a simulation based paradigm. Hence we confront an ideal world, represented by the ideal functionality defined in Figure 3 and the real world corresponding to the protocol. The output distributions of both the functionality and the protocol should be indistinguishable, even in the presence of

an adversary. To prove this, we use a Simulator that operates the transition from the ideal world to the real one. Note that it is possible to directly build an extractor from distinct transcripts, following the same reasoning, but it seemed to us less intuitive to expose. We give a sketch of proof, the detailed proof can be found in the full version (`shorturl.at/eDJMY`). As m is a common public input, we write $f(x)$ for $f(x, m)$.

Corrupted prover P^*: soundness. We first describe a Simulator, interacting with a corrupted prover and having access to all the extractable information, calling as a subroutine the extractors for ZKBoo and for the proof Π on three accepting transcripts (a, e_1, p_1), (a, e_2, p_2), (a, e_3, p_3). This Simulator obtains the desired witnesses and sends them to the functionality only if he is sure that they are correct. Facing the Simulator, P^* has no chance to cheat (no statement without a correct witness can lead to an accept). Then, following a game based reduction, we show that the view of P^* playing with this Simulator is indistinguishable from the view of P^* playing in the real protocol.

The key point is when the Simulator does not check the equality between the values extracted from ZKBoo and the values extracted from the algebraic commitment, and only rely on the relations given by the tag values. There we show that the 1-varCI-ow- security of f induces the equalities. As the Simulator does not use its extraction knowledge, he acts as a real verifier.

Corrupted verifier V^*: Zero-Knowledge. We describe how a Simulator facing a corrupted verifier manages to provide a view that is indistinguishable from a real protocol execution view. The Simulator first acts as if he did not know the witness x and the opening informations r_x, r_y.

Then by successive games we go back to the original protocol. The transition from the Simulation to the real protocol consists in substituting, step by step, the random witness and correlated values used by the Simulator by real values. For each step, we show that there is little chance that the corrupted verifier sees any difference (the probability of distinguishing the two distributions is bounded by a negligible value that depends on the security of the underlying functions). For most of the transcript, the indistinguishability is obtained by calling the hiding property of the commitment scheme and the ZK knowledge property of ZKBoo and Π. However, this is not the case for t_1, t_2. For those elements, we show that 1-varRKA-wPRF security provides us with the desired indistinguishability.

E.2 Theorem 2

Again, we give a sketch of the proof. A detailed proof can be found in the full version (`shorturl.at/eDJMY`). Correctness follows by inspection. **Soundness.** Considering the soundness, an extractor accessing three distinct accepting transcripts works as follows. In a first step it acts as the ZKBoo extractor and uses the transcripts to recompose a value $x^* = x_1 \oplus x_2 \oplus x_3$ and a value $y^* = y_1 \oplus y_2 \oplus y_3$ such that $y^* = f(x^*)$. From the equivocality of the commitment, he also knows that the two randomness given in each transcript only corresponds to 3 random values r_{y_i} such that (y_i, r_{y_i}) opens C_{y_i}. In a second step, the simulator extracts

the opening $(y'[j], r_{y'[j]})$ of the bit commitments $C_{y[j]}$, for $j \in [0, |j| - 1]$. From his knowledge of the $r_{y'[j]}$ and the r_{y_i} he can recompose a valid randomnness r^* such that y^*, r^* opens C_y. Hence x^*, r^* is a valid extraction.

Zero-Knowledge. Again, we cannot directly call the Simulator of ZKBoo as the output of the circuit is not public. Instead we build a simulator that runs as the original ZKBoo simulator, and simulates two views w_e and w_{e+1} for a fixed e and their associated outputs y_e and y_{e+1}. He also generates the associated commitments with random values r_{y_e}, $r_{y[e+1]}$ The simulator also commits to random bit to generates the $C_{y[j]}$ except for the last that he computes as $C_y[0] = C_y - \sum_{j=1}^{|y|-1} C_{y[j]}$ such that the relation is verified. He can produce all the proofs that the $y[j]$ are bits except for $y[0]$ for which he call the Simulator for Π_0. With non negligible probabilities, the simulator obtains a valid transcript for the same challenge e. In a last step, he samples a random r_z and, putting together C_y, the $C_{y[j]}$ and a $\mathsf{Com}(0, r)$ he can produce the missing $C_{y_{e+2}}$. As the only values our Simulator adds to the ZKBoo and the Π_0 simulations are Commitments, the hiding properties of the commitment scheme provides the indistinguishability between the simulation and a real execution of the protocol.

F Implementation optimization

The first improvement, ZKBoo++ is given in [16]. The authors meticulously analyse which data should be sent by the prover and which one can be directly computed by the verifier. They show that one can cut by more than half the size of the proof, at no computational cost. They implemented their solution on an optimized circuit for SHA_256 with 22272 AND gates and obtained a proof size of 618KB for a soundness error of 2^{-128} (48% of ZKBoo proof size on the same circuit). For a soundness parameter $\sigma = 80$ they obtain 385KB. Considering this reduction, our own proof would drop to 1.6MB. Moreover, the optimized SHA_256 circuit enables to save around 4000 AND gates, cancelling the cost of the modular reduction. We now turn to the KKW protocol [33]. The author use another MPC solution to decrease the number of rounds required to reach a desired soundness security. They show that the improvement one can expect on the size of the proof depends on the number of AND gates. Considering an average of 95000 AND gates for our circuit, the proof size drops by 70KB compared to ZKBoo++. Hence we could obtain proofs around 90KB.

F.1 Comparison with SNARKs solutions

Agrawal *et al.* proposed in [1] a ZK protocol mixing algebraic commitments on input and output and circuit evaluation (comIOSnark). It is worth comparing their solution to ours. No implementation is available in [1], however they estimate the prover's work to four exponentiation in addition to the number of exponentiations for computing the SNARK proof when the *Verifier* has to perform 4 exponentiations and 30 pairings. To compare to more practical results,

we consider the protocol Pinocchio [37], on which the protocol of [1] bases its description. The implementation for Pinocchio is performed on a single core of a 2.67 GHz Intel Core i7 with 8 GB of RAM, which is comparable to our test setting. A proof on a SHA_1 evaluation with 23785 multiplication gates requires, first, a public key generation of 11 seconds. The proof computation takes 15.7 seconds. As expected however, the *Verifier*'s running time is only around 10ms and the proof size is 288. Even if optimizations can be performed, the *Prover*'s work is far more important than in our solution. Hence we believe that, depending on the applications targeted, either one or the other solution might be of interest.