Updatable Privacy-Preserving Blueprints

Bernardo David[1], Felix Engelmann[2][0000−0001−9356−0231], Tore Frederiksen[3], Markulf Kohlweiss[4][0000−0002−8660−9663], Elena Pagnin[5][0000−0002−7804−6696], and Mikhail Volkhov[4]

[1] ITU Copenhagen beda@itu.dk
[2] Lund University fe-research@nlogn.org *
[3] Zama tore.frederiksen@zama.ai **
[4] University of Edinburgh, IOG {mkohlwei,mikhail.volkhov}@ed.ac.uk
[5] Chalmers, elenap@chalmers.se

Abstract. Privacy-preserving blueprints enable users to create escrows using the auditor's public key. An escrow encrypts the evaluation of a function $P(t, x)$, where t is a secret input used to generate the auditor's key and x is the user's private input to escrow generation. Nothing but $P(t, x)$ is revealed even to a fully corrupted auditor. The original definition and construction (Kohlweiss et al., EUROCRYPT'23) only support the evaluation of functions on an input x provided by a *single* user.

We address this limitation by introducing *updatable* privacy-preserving blueprint schemes (UPPB), which enhance the original notion with the ability for multiple parties to non-interactively update the private value x in a blueprint. Moreover, a UPPB scheme allows for verifying that a blueprint is the result of a sequence of valid updates while revealing nothing else.

We present uBlu, an efficient instantiation of UPPB for computing a comparison between private user values and a private threshold t set by the auditor, where the current value x is the cumulative sum of private inputs, which enables applications such as privacy-preserving anti-money laundering and location tracking. Additionally, we show the feasibility of the notion generically for all value update functions and (binary) predicates from FHE and NIZKs.

Our main technical contribution is a technique to keep the size of primary blueprint components independent of the number of updates and reasonable for practical applications. This is achieved by elegantly extending an algebraic NIZK by Couteau and Hartmann (CRYPTO'20) with an update function and making it compatible with our additive updates. This result is of independent interest and may find additional applications thanks to the concise size of our proofs.

Keywords: Updatable NIZKs · Privacy-Preserving Blueprints.

1 Introduction

Data protection often demands that the actions and personal information of individual users be kept private. At the same time, regulatory organizations should be able to learn about potential violations evidenced by the combined actions of multiple users and track down the misbehaving parties. In such scenarios, cryptographic techniques such as multiparty computation come to mind as potential solutions. While such techniques may trivially reconcile the paradox of privacy vs. accountability, they often impose performance overheads and trust assumptions that are incompatible with the original scenario.

Multiparty computation (MPC) [43, 24] is a natural solution for auditing private data without undermining the privacy of users or jeopardizing the audit process by revealing sensitive audit parameters. An auditor could execute an efficient MPC protocol (*e.g.*, [31]) together with users to audit their data while only revealing the final audit result. However, traditional MPC requires *all* participating parties to interact in *each* computation. It is unrealistic to require all users to interact with each other at once, as opposed to only interacting when generating the audit data, as parties may simply be unavailable or technically incapable of executing long-running complex protocols. Moreover, processing large volumes of audit data via traditional MPC would prove prohibitively expensive due to the network interaction required from *all* participating parties.

* Part of this work was done while at the ITU Copenhagen.
** Part of this work was done while at the Alexandra Institute.

Privacy-Preserving Blueprints: Adding Updates. A privacy-preserving blueprint scheme [52] allows a user to create an escrow that only reveals the function $P(t, x)$ where t is a private input from the auditor and x is a private value from the user. This notion allows an auditor who sees a blueprint to non-interactively learn $P(t, x)$ at any point in time, allowing for asynchronous audits. However, $P(t, x)$ can only be computed on an input x that is fully known to a single party who generates the blueprint. This is clearly insufficient when inputs come from multiple mutually distrusting users and the original constructions of blueprints cannot be efficiently extended to this case. We focus on overcoming this limitation with the notion of updatable privacy-preserving blueprints, which allow a party departing from an initial blueprint for $P(t, x)$ to obtain an updated blueprint for $P(t, x')$, where x' is an output of a function of the current x and the party's private input x_i, without learning anything about t, and nothing besides this functional relation about x or x'. Moreover, our notion allows for parties to check that a sequence of blueprints has been obtained by successive updates done by different parties using their respective private inputs, without learning anything else. Our main construction supports evaluation predicates of the form "output 1 if $x_1 + \cdots + x_n \geq t$ and 0 otherwise", where each x_i is added to the blueprint as an update by a different party. Further, we show how updatable blueprints can work with other predicates, expressible as a special class of multivariate polynomials, and non-binary outputs, allowing more elaborate applications including privacy-preserving location testing.

Applications of Updatable Privacy-Preserving Blueprints. Our new notion can be seen as a type of MPC and therefore can be used in different areas, such as privacy-preserving auditing, financial accountability, reputation systems, private auctions, or voting. The common pattern in these applications is that they involve both sensitive audit parameters provided by authorities (or a collective of users) and the private data of multiple individual parties. Current approaches, especially in financial regulation-compliance, sacrifice either privacy (when legally possible) or accuracy, which is lost as a result of auditing incomplete data.

While the notion of updatable privacy-preserving blueprints is generic, our running motivational example is implementing anti-money laundering (AML) policies. In decentralized finance, AML schemes for privacy-preserving cryptocurrencies either only support auditing users in isolation [59, 33] or rely on revocable anonymity [2, 49, 29], requiring the auditor to learn all information about users' activities, severely undermining their privacy. Updatable blueprints can address these issues in both scenarios by enabling privacy-preserving AML audits on joint data from multiple users.

Intuitively, an updatable blueprint scheme can be used to securely compute and validate a suspiciousness score for the users' bank accounts based on the features of transactions between multiple banks — and the private information these banks hold about their customers. A blueprint is attached to each account to accumulate a score that is based on the reputation of customers at other banks transferring funds into the account. Before such a transfer the sending bank must receive an up-to-date blueprint with the suspiciousness score from the receiving bank. It then augments the transfer with an updated blueprint with an update value x_i based on the value of the transaction and the reputation of the sending account owner. Upon receiving the transfer the receiving bank records the updated blueprint as the new up-to-date blueprint. At any moment, an auditor (*e.g.* a tax authority) can validate the correctness of this procedure and check if a user's score exceeds a certain threshold without learning the score. If so, the auditor can further request the opening of related commitments held by other banks. All of these operations do not require the users nor the banks to perform any extra rounds of communication and only add minimal overhead to the existing communication and computation involved in transfers. A similar idea can be used to enforce limits on the amount of funds transferred in decentralized finance applications using cryptocurrencies.

1.1 Our Contributions

In this work, we introduce the notion of *updatable* privacy-preserving blueprints, constructions, and applications. Our main results are summarized as follows:

Updatable Privacy-Preserving Blueprint: In Section 4, we introduce our new notion, extending privacy-preserving blueprints [52] to allow for an auditor to learn a predicate $P(t, x)$ of their own private input t and users' private inputs x such that users can update x in an existing blueprint.

Efficient Construction for Range Predicates: In Section 5, we present uBlu, an efficient realization of UPPB for a comparison predicate between private user values and a range defined by the auditor's private threshold. This includes an update mechanism that allows users to update the current input value.

Succinct Updatable Algebraic NIZK: As a core technique in our construction, we show in Section 3 and Appendix B that we can extend the NIZK proof system of [26] to allow for updating the witness and the instance of an existing NIZK while maintaining its efficiency and security.

Applications to AML and Extensions: In Section 7, we use our efficient uBlu instantiation to implement a privacy-preserving AML algorithm. In Section 6, we discuss how updatable blueprints can be used to evaluate more general predicates and additional applications including privacy-preserving location-based services and enforcing limits on funds transferred in decentralized cryptocurrencies.

We remark that the main technical contribution of this work is a mechanism for updatable range predicates. In the core of the algorithm, we employ a certain type of more standard ElGamal-inspired homomorphic computation for polynomial update and evaluation. However, our technique allows us to construct *sound* blueprints that *do not grow with the number of updates*, and are reasonably sized for practical applications. By investigating and employing the updatability of CH20 NIZK [26], we show that a given ElGamal ciphertext contains a correct and consistent evaluation of a polynomial whose coefficients are in other ElGamal ciphertexts, and the evaluation point is in a Pedersen commitment. Furthermore, when x is updated, the NIZK is updated too, while staying concise and efficient when used in generation and verification algorithms.

1.2 Overview of our Techniques

The Notion of Updatable Privacy-Preserving Blueprints. A UPPB scheme is defined in terms of a base commitment scheme and a public predicate $P(t, x)$ taking two private inputs: a fixed auditor value t and an aggregate value $x = \mathsf{fold}(x_1, \ldots, x_n)$ where fold is an online streaming algorithm and each x_i is provided by a different user in committed form (we focus on $\mathsf{fold}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$). The updatable functionality for blueprints requires several components: escrows esc, tags tag, hints hint, and commitments $\mathfrak{C}$. Next, we elaborate on how these components interact.

Given an escrow esc for a corresponding history of commitments $\mathfrak{C}_i$ to the values x_i, the auditor learns the value of $P(t, \mathsf{fold}(x_1, ..., x_n))$ but nothing else, and third parties learn nothing. The auditor generates a key pair $(\mathsf{sk}, \mathsf{pk})$ and a blueprint hint_0 for predicate $P(t, 0)$, where sk and t remain private but pk is published, while hint_0 is passed to the first user. Blueprint hint values store the aggregated x so far and can be verified with respect to a history of base commitments. Specifically, using pk and hint_{i-1}, a user can extend the history with their value x_i, each time deriving an updated hint_i. In our construction, hint_{i-1} will contain an aggregate value $\sum_{j=1}^{i-1} x_j$ that users can update via homomorphic operations to add their private input x_i. A hint_i can be transformed into an escrow, esc, which reveals nothing to the users but can be used by an auditor who knows sk to learn *only* whether $P(t, \sum_{j=1}^{i} x_j) = 1$, while keeping the x_j values private. When a user updates hint_{i-1} to hint_i, they obtain an update tag_i for verifying the validity and consistency of the commitment history by running $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{j=1}^{i})$, of the hint by running $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_i, \mathsf{tag}_i)$, and of esc by running $\mathsf{VfEscrow}_{\mathsf{pk}}(\mathsf{esc}_i, \mathsf{tag}_i)$. We have the following three properties: (i) tag_i can only verify for a single history, *i.e.*, valid histories do not collide in their tags; (ii) we can extract the openings x_j of all base commitments from a valid history; (iii) if esc verifies with respect to tag_i, then it indeed is an escrow of predicate value $P(t, \sum_{j=1}^{i} x_j)$ for the openings x_j of these base commitments.

We require hints, escrows, and tags derived from valid hints to be hiding. However, as hints contain the aggregate value, $\sum_{j=1}^{i} x_j$, hiding for hints only holds against adversaries who do not know sk. Thus, hints should only be used for updates between users who cooperate to gain privacy, but must not be given to the auditor as this would leak the current aggregate value. Finally, our scheme preserves the hiding and binding properties of the base commitment scheme.

A Generic but Inefficient Construction. Departing from circuit private FHE and generic NIZKs for NP, we can construct a UPPB scheme for arbitrary predicates and updates. The auditor generates an FHE key pair $(\mathsf{pk}, \mathsf{sk})$ and publishes pk along with an encryption of t under pk as the auditor's public key. Users

encrypt their private inputs x_i under pk and use the resulting ciphertext to obtain an updated hint_i containing $\mathsf{fold}(x_1, \ldots, x_i)$ by homomorphically evaluating the update function corresponding to the fold algorithm on this ciphertext and hint_{i-1}. The update tag tag_i can be obtained by generating a NIZK showing that hint_i was correctly computed by evaluating the update function on hint_{i-1} and the ciphertext containing x_i . To generate a predicate escrow esc, a user homomorphically evaluates the predicate $P(t, \mathsf{fold}(x_1, \ldots, x_i))$ using hint_i and the ciphertext containing t (obtained from the public key) and computes the corresponding tag as a NIZK showing that the predicate was correctly evaluated. Using sk the auditor can decrypt esc and learn only the output of $P(t, \mathsf{fold}(x_1, \ldots, x_i))$ but nothing else, while the validity of esc can be checked by verifying the tag NIZK. This scheme provides support for arbitrary predicates and updates (beyond additive ones) but clearly has a very high concrete computational complexity due to the use of heavy primitives such as FHE and NIZKs for homomorphic computation on FHE ciphertexts. This is a serious caveat in a setting such as that of AML and location privacy, where hundreds of thousands of updates must be processed per second.

An Efficient Construction for Range Predicates. Range predicates $P(t, X)$ are evaluated over ordered sets and output 1 iff $X \geq t$. To construct a concretely efficient UPPB scheme called uBlu, we will consider the subclass of range predicates $P_d(t, X)$ that can be expressed as a degree d polynomial over $\mathbb{Z}_q$ (for an opportune module q and degree d) with roots at positions $[t, \ldots, t + d - 1]$. Here d is the size of the range for which $P(t, X) = 1$ and t is the auditor's secret threshold. The core of our scheme is a mechanism that relies on Pedersen commitments, ElGamal ciphertexts, and NIZKs with updatable witness/instances for predicate evaluations in the multi-user scenario, guaranteeing the soundness of the updates.

In detail, the auditor ElGamal encrypts the powers $(-t)^i$ for $i \in [d]$, in the exponent. The ciphertexts are included in the initial hint hint_0. Users can combine the ciphertexts to homomorphically evaluate the polynomial $P_d(t, x)$ at their input x, in the exponent. Moreover, given hint_{i-1} (containing an encryption of $P_d(t, x)$) a user with input x' can update the hint to hint_i containing an encryption of $P_d(t, x + x')$. This is achieved by an algebraic transformation on the polynomial representation, as described in Section 5.1.

When converting hints to escrows esc, users homomorphically reconstruct the ElGamal ciphertext for the value $P_d(t, x)$, and then exponentiate it by a random β obtaining an encryption of $\beta \cdot P_d(t, x)$. If the polynomial evaluates to 0, the randomization has no effect and the auditor is able to decrypt the message G^0 (this is the case where the predicate $P(T, X) = [P_d(T, X) \stackrel{?}{=} 0]$ outputs 1). Otherwise, if the polynomial evaluation is not a root, the auditor learns nothing.

The main technical achievement of this work is the design of hints that do not grow with the number of updates, and are reasonably sized for practical applications. We make hints linear in the degree d of the polynomial and endow them with *updatable* proofs that attest to hint consistency. Intuitively, this is achieved by including witness-products in the witness and checking the consistency of those products w.r.t. the minimal witness values by using Pedersen commitments and adding extra checks in the relation to ensure input consistency based on these commitments. Thanks to the homomorphic property of Pedersen commitments, our witness and instance are updatable: randomnesses and x values accumulate additively (see Section 5.3 for further details).

For the many use cases which *do require* a linear history of proofs-of-updates, we provide a very concise "update history" consisting of update tags tag, which naturally grows in the number of updates, but allows enforcing update accountability. In addition, as we use Pedersen as base commitment, it is easy to integrate our uBlu construction with applications such as credentials and private payment systems: one simply proves *extra* statements about base commitments, *e.g.*, that their values are equal to a credential attribute or a payment transaction value.

1.3 Related Work

Homomorphic commitments [28, 19] and functional commitments [20] could potentially support general updates and predicate evaluations for UPPB schemes. However, these operations would require knowledge of individual commitment openings, making such schemes unfit for our multi-user setting, where users'

inputs must be kept private. The threshold predicate for which we present an efficient uBlu scheme is closely related to Yao's Millionaire's Problem, *i.e.* performing secure comparison. While there are many protocols for secure comparison (*e.g.*, [30, 40]) that could be used in this setting, they would require continuous online involvement of parties and many rounds of interaction. In our setting, no interaction is required from the auditor or users (after they update commitments).

Updatable NIZKs, such as CH20 [26] used as our prime technical tool, have been previously investigated in [8, 21, 23, 48]. While recursive approach to NIZK updatability becomes more practical over time [11, 9, 22, 13, 18, 16, 17, 56], direct malleability without recursion is more lightweight and thus more suitable for tailored application, such as various signature schemes [32, 12, 37, 48], anonymous credentials [1], scalable mix-nets [47] etc. The malleability of CH20 was observed in [25] to build anonymous credentials and structure-preserving signatures on equivalence classes. It is also worth noting that RO-based NIZKs are essentially non-malleable unless recursively [34, 54, 39]. This is why the CH20 NIZK combining the simplicity of a Schnorr-like proof, together with a bilinear setup avoiding the ROM limitation, stands out as a natural candidate for direct updatability.

A series of recent papers explored accountable law enforcement access system [44, 36, 66, 45, 6, 53]. A different approach to privacy-preserving AML securely computing similarity scores of transaction graphs [38, 61]. However, as computing similarity among many accounts is expensive, the authors of [61] suggest that banks pre-select accounts, which can be done utilizing our techniques for computing aggregate suspiciousness scores. Our applications are most closely related to a subcategory of these works on abuse-resistant regulation compliance [36, 6, 53] where users have full privacy against a malicious auditor. A consequence of full privacy for users is that the auditor's detection policy must be kept private since users can adapt their behavior to avoid detection. Hence, it is necessary to securely compute a function on private inputs from the user and the regulator.

2 Preliminaries

Notation. We will use $\lambda \in \mathbb{N}$ as a security parameter, often implicitly. PPT stands for "probabilistic polynomial time". The dollar-arrow combination $\xleftarrow{\$}$ denotes uniform sampling, e.g., $x \xleftarrow{\$} S$ means that x is picked according to the uniform distribution over the set S. By extension, we denote probabilistic executions as $x \xleftarrow{\$} \mathsf{Alg}(y)$ where we implicitly run the algorithm Alg with uniform randomness; when needed we specify the random coins r to be used for the execution as $x \leftarrow \mathsf{Alg}(y; r)$. Whenever we do not explicitly use one output of an algorithm, we denote the unspecified output place with $\cdot$, e.g., $(x, \cdot) \leftarrow \mathsf{Alg}(y)$ defines the first output to be x and leaves undefined the second output (well-determined by the input y). Polynomials and variables are written in capital letters, e.g. $F(X, Y) \in \mathbb{Z}[X, Y]$, while instantiated values are in small letters, e.g. $F(x, y) \in \mathbb{Z}$ for $x, y \in \mathbb{Z}$. Square braces generically denote predicate evaluation, e.g. $[a = b]$ is a boolean value. We use pairs like $1/0$ or $\top/\bot$ for readability as aliases for $1/0$. In pseudocode, $\mathsf{assert}b$ is equivalent to "if not b, then return 0". For two positive integers $m < n$, we denote discrete intervals of the form $\{m, m+1, \ldots, n\}$ as $[m, n]$; in the special case $m = 1$, we compact to $[n]$. Within mathematical expressions, the symbols "|" "." and ":" are used interchangeably as "such that" or "given that". The equality-question-mark combination $\stackrel{?}{=}$ denotes the boolean result of an equality check between the expressions surrounding the symbol, while $\neq$ denotes the negation of $\stackrel{?}{=}$.

We will use type-III bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3)$. We will denote group elements by their exponents using square brackets: $[a]_b \in \mathbb{G}_b$ for $b \in \{1, 2, 3\}$; when we have a vector of $\mathbb{Z}$ element, denoted by either $\vec{x}$ or $\boldsymbol{x}$, then $[\vec{x}]_b$ is a vector of $\mathbb{G}_b$ elements. In our protocols, we will mostly work in $\mathbb{G}_1$; whenever clear from the context, we will drop the index and just write $\mathbb{G}$ for this group, and G for its generator.

Secure Commitments. Following [53] we use non-interactive commitments to bind inputs to externally committed values.

Definition 1 (Statistically Hiding Non-Interactive Commitment). *A pair of algorithms* $(\mathfrak{Setup}, \mathfrak{Commit})$*, defined over message space* $\mathbb{V}$ *and randomness space* $\mathbb{R}$*, constitutes a statistically hiding non-interactive commitment scheme if it satisfies:*

- Statistical hiding*, i.e., for any* $\mathfrak{pp}$ *output by* $\mathfrak{Setup}(1^\lambda)$*, for any* $m_0, m_1 \in \mathbb{V}$*, the distributions* $D(\mathfrak{pp}, m_0)$ *and* $D(\mathfrak{pp}, m_1)$ *are statistically close, where* $D(\mathfrak{pp}, m) = \{r \xleftarrow{\$} \mathbb{R} : \mathfrak{Commit}_{\mathfrak{pp}}(m; r)\}$*; and*
- Computational binding*, i.e. for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that* $\Pr[\mathfrak{Commit}_{\mathfrak{pp}}(m_0; r_0) = \mathfrak{Commit}_{\mathfrak{pp}}(m_1; r_1) \wedge m_0 \neq m_1 : \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda); (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\mathfrak{pp})] = \mathsf{negl}(\lambda)(\lambda)$.

Our main instantiation will use the Pedersen commitment scheme over $\mathbb{V} = \mathbb{R} = \mathbb{Z}_q$ in a group $\mathbb{G}_1$ of prime order q, which exists as part of a bilinear group setup $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3)$.

3 Updatable NIZK Proof Systems

Zero-knowledge proofs enable a prover to convince a verifier of the validity of a statement, without revealing any additional information beyond the truth of the statement itself. Further, non-interactive zero-knowledge proofs (NIZK) remove the need for interaction between prover and verifier: the prover simply outputs a publicly verifiable proof generated from her (secret) witness and a common reference string (crs) [63].

We first discuss NP languages, relations, and their properties. Notation-wise, we will denote languages and relations interchangeably in the following manner: $\mathcal{L}_\mathcal{R} = \{\mathsf{x} \mid \exists\ \mathsf{w}.\, (\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}\}$, where $\mathsf{x} \in X$ is a (public) instance, $\mathsf{w} \in W$ is a (secret) witness, and the relation $\mathcal{R}_\mathcal{L}$ is a subset of $X \times W$. Defining $\mathcal{R}$ also determines $\mathcal{L}_\mathcal{R}$ uniquely; furthermore we implicitly assume $\mathcal{L}$ uniquely defines $\mathcal{R}_\mathcal{L}$.

Definition 2 (Updatable Language). *A language* $\mathcal{L}$ *is* updatable *w.r.t. the class of transformations* $\mathcal{T}$ *if for all* $T \in \mathcal{T}$*,* $T = (T_\mathsf{x}, T_\mathsf{w})$*, and for all* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$ *it holds that* $(T_\mathsf{x}(\mathsf{x}), T_\mathsf{w}(\mathsf{w})) \in \mathcal{R}_\mathcal{L}$*. We call such* T valid transformations *for* $\mathcal{L}$*.*

Note that the functions $T_\mathsf{x} : X \to X, T_\mathsf{w} : W \to W$ are defined independently of any particular instance and witness, i.e. in T_x the symbol "x" is only used as a label. All the relations and functions we consider can be evaluated in PPT in the security parameter.

We recall the definitions of standard and updatable NIZK proofs. Updatable NIZKs are known as malleable proofs [8, 21] and allow transforming a proof for x into a proof for $T_\mathsf{x}(\mathsf{x})$. We prefer the term updatable and make this feature explicit in our syntax.

Definition 3 (Standard and Updatable NIZKs). *An* updatable *NIZK proof system for a language* $\mathcal{L}$ *and a set of transformations* $\mathcal{T}$ *is defined by:*

- $\mathsf{Setup}(\lambda) \xrightarrow{\$} (\mathsf{crs}, \mathsf{td})$*: generates a common reference string* crs *and a trapdoor* td *which is used for security definitions;*
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}) \xrightarrow{\$} \pi$*: produces a proof for* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$*;*
- $\mathsf{Verify}(\mathsf{crs}, \pi, \mathsf{x}) \to 0/1$*: verifies* π *w.r.t. the public instance* x*;*
- $\mathsf{Update}(\mathsf{crs}, \pi, \mathsf{x}, T) \xrightarrow{\$} \pi'$*: updates the proof* π *for* x *into* π' *for* $T_\mathsf{x}(\mathsf{x})$*.*

A standard *non-updatable NIZK proof system is defined only by the first three algorithms. For conciseness, in what follows we drop* crs *from the explicit inputs and separate the proof and instance by a semicolon, e.g.,* $\mathsf{Verify}(\pi; \mathsf{x})$*.*

Our NIZKs need to satisfy the standard security definitions — completeness, soundness, and zero-knowledge, given e.g., in [46] — which we defer to Appendix A. An updatable NIZK must additionally satisfy the following two properties. First, the updated proof must be valid for the updated instance:

Definition 4 (Update Completeness). *An updatable NIZK proof system for* $\mathcal{L}$ *satisfies* update completeness *w.r.t. a set of transformations* $\mathcal{T}$*, if given* $(\mathsf{crs}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, for all* x, π *such that* $\mathsf{Verify}(\pi, \mathsf{x}) = 1$*, and all* $T = (T_\mathsf{x}, \cdot) \in \mathcal{T}$ *it holds that:* $\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{Update}(\mathsf{crs}, \pi, \mathsf{x}, T), T_\mathsf{x}(\mathsf{x})) = 1\right] = 1$.

Second, derivation privacy states that updated proofs are distributed similarly to fresh proofs for the new instance.

Definition 5 (Derivation Privacy). *An updatable NIZK proof system for $\mathcal{L}$ satisfies* derivation privacy *w.r.t. $\mathcal{T}$, if given $(\mathsf{crs}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$, all π such that $\mathsf{Verify}(\mathsf{crs}, \pi, \mathsf{x}) = 1$, and all $T = (T_\mathsf{x}, T_\mathsf{w}) \in \mathcal{T}$ it holds that: $\left\{\mathsf{Update}(\mathsf{crs}, \pi, \mathsf{x}, T)\right\} \stackrel{p}{=} \left\{\mathsf{Prove}(\mathsf{crs}, T_\mathsf{x}(\mathsf{x}), T_\mathsf{w}(\mathsf{w}))\right\}$.*

Because updated proofs are distributed as fresh ones, they can be simulated using the standard simulator guaranteed by zero-knowledge; therefore transformed proofs are also zero-knowledge. This property is inspired by derivation privacy in [21].

NIZKs Used in This Work. We use two NIZK proof systems that work with the same class of languages.

- Π: The standard non-updatable Σ-protocol proof system for equality of discrete logarithm relations [67, 57]. It is assumed to be straight-line knowledge-sound after non-interactive transformation, e.g. by encrypting witnesses or using Fischlin's technique [35].
- Π_u: The CH20 NIZK [26], which is Σ-like, but is updatable, works in the bilinear setting, and has a uniform CRS. We discuss CH20 and investigate its updatability in Appendix B. Although the malleability of CH20 was used in [25], we believe that we provide a more focused analysis of it, which can be of independent interest.

The Common Algebraic Language. Let $\mathbb{G}$ be a prime ordered group. Define the set of linear polynomials $\mathcal{P} \subset \mathbb{G}[X_1 \dots X_l]$ in l variables with coefficients in $\mathbb{G}$ as $\mathcal{P} = \{a_0 + \sum_{i=1}^{l} a_i X_i \mid a_0 \in \mathbb{G}, a_1 \dots a_l \in \mathbb{Z}_q\}$. Both Π and Π_u work with the so-called *algebraic* language[6] $\mathcal{L}_M$:

$$\mathcal{L}_M = \left\{ \vec{\mathsf{x}} \in \mathbb{G}^l \mid \exists\, \vec{\mathsf{w}} \in \mathbb{Z}_p^t : M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \vec{\mathsf{x}} \right\}, \text{ where } M(\vec{X}) \in \mathcal{P}^{l \times t}.$$

In other words, it is a set of DLOG-like linear equations with a common instance and bases in $M(\vec{X})$ that can potentially depend on the instance x itself. We define the corresponding relation $\mathcal{R}_M$ to be the set $\{(\vec{\mathsf{x}}, \vec{\mathsf{w}}) \in \mathbb{G}^l \times \mathbb{Z}_p^t \mid M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \vec{\mathsf{x}}\}$.

Updatability for algebraic languages $\mathcal{L}_M$, due to their group structure, means that there exist four matrices $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ such that for all $(\vec{\mathsf{x}}, \vec{\mathsf{w}}) \in \mathcal{R}_M$ it holds that:

$$(T_{\mathsf{xm}} \cdot \vec{\mathsf{x}} + T_{\mathsf{xa}}, T_{\mathsf{wm}} \cdot \vec{\mathsf{w}} + T_{\mathsf{wa}}) \in \mathcal{R}_M.$$

The functions $T_\mathsf{x}, T_\mathsf{w}$ required in Definition 2 are defined as follows: $T_\mathsf{x}(\mathsf{x}) := T_{\mathsf{xm}} \cdot \vec{\mathsf{x}} + T_{\mathsf{xa}}$ and $T_\mathsf{w}(\mathsf{w}) := T_{\mathsf{wm}} \cdot \vec{\mathsf{w}} + T_{\mathsf{wa}}$. We will show that the algebraic languages we define in this work are updatable by explicitly providing the matrices and proving they satisfy the equation. For more details, including an illustration of updatability for a simple language, see Appendix B.

4 Updatable Privacy-Preserving Blueprints

Updatable privacy-preserving blueprints are like privacy-preserving blueprints [51, Definition 3], in the sense that they are defined for a given function (in our case an efficiently computable predicate) and a given non-interactive commitment scheme. In addition to the basic algorithms, updatable privacy-preserving blueprints have three new algorithms that are necessary for our functionality: Update, $\mathsf{VfHistory}$, and VfHint. Finally, to enable proving external properties about update values we require the commitment scheme to be statistically hiding and computationally binding (see Definition 1). For integration, the commitment scheme additionally has to be compatible with our updatable NIZK system. We also introduce hints and tags which repackage and augment some of the variables used in the original syntax of [51, Definition 3]. For convenience, we will refer

[6] For simplicity, and contrast with [26], we do not consider arbitrary $\Theta(\vec{\mathsf{x}})$ such that $M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \Theta(\vec{\mathsf{x}})$.

to updatable privacy-preserving blueprint schemes as *updatable blueprints* (UPPB), and to the statistically hiding non-interactive NIZK-friendly commitment scheme as *the base commitment scheme* and denote it as $\mathsf{BC} = (\mathfrak{Setup}, \mathfrak{Commit})$ (with gothic font). We will also employ binary predicates, i.e. functions of the form $P(T, X) : \mathbb{V} \times \mathbb{V} \to \{0, 1\}$.

For a concrete example, consider Pedersen commitments, the CH20 NIZK, $\mathbb{V} = \mathbb{Z}_q$, and the predicate family $\mathcal{P}_{\mathbb{V}}$ of range checks, i.e., binary predicates that are parameterized by a public distance value $d \in \mathbb{Z}_q$ that return 1 if $x \in [t, t + d - 1]$, and 0 otherwise.

Definition 6 (Updatable Privacy-Preserving Blueprints). *Let* $\mathsf{BC} = (\mathfrak{Setup}, \mathfrak{Commit})$ *be a statistically hiding non-interactive commitment scheme defined over* $(\mathbb{V}, \mathbb{R})$*, and* $P(T, X) \in \mathcal{P}_{\mathbb{V}}$ *be an efficiently computable binary predicate defined over* $\mathbb{V}$*. An updatable blueprint scheme (UPPB) for* $(\mathsf{BC}, P(T, X))$ *is defined by the following set of PPT algorithms:*

$\mathsf{Setup}(1^\lambda, \mathfrak{pp}) \xrightarrow{\$} (\mathsf{pp}, \mathsf{td})$**:** *the setup algorithm is randomized,takes as input the security parameter* λ*, and base commitment parameters* $\mathfrak{pp}$ *output by* $\mathfrak{Setup}(1^\lambda)$*. It returns public parameters that contain at least a description of a special value denoted by 0. It also returns a trapdoor* td *that is only used in security definitions.* pp *and* $\mathfrak{pp}$ *are implicit inputs to all other algorithms.*

$\mathsf{KeyGen}(t) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0)$**:** *the key generation algorithm is randomized, ittakes as input a threshold value* $t \in \mathbb{V}$*. It outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$*; a hint* hint_0 *implicitly encoding 0 — this first hint is also the only public one.*

$\mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) \to 1/0$: *the verify key generation algorithm is deterministic, itverifies the validity of the public output of* KeyGen*.*

$\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag}, x, \mathfrak{r}) \xrightarrow{\$} (\mathsf{hint}', \mathsf{tag}')$**:** *the update algorithm is randomized, ittakes as input a hint and its tag, a value, and external base commitment randomness. When* $\mathsf{hint} = \mathsf{hint}_0$*,* $\mathsf{tag} = \perp$*. It returns an updated* hint' *and the new update* tag'*. To keep track of the update history (also called trace) of a commitment we introduce an epoch index* $\iota \geq 1$*, e.g.,* $\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, \mathsf{tag}_{\iota-1}, x_\iota, \mathfrak{r}_\iota) \xrightarrow{\$} (\mathsf{hint}_\iota, \mathsf{tag}_\iota)$*.*

$\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) \to 1/0$: *the verify history algorithm is deterministic, ittakes as input an ordered sequence of update tags and base commitments and verifies the consistency of the update history (also called trace, see Definition 7).*

$\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag}) \to 1/0$: *the verify hint algorithm is deterministic, ittakes as input a hint and the last corresponding update tag. It returns* 1 *if the inputs are deemed to be consistent (see Definition 7), and* 0 *otherwise.*

$\mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}) \xrightarrow{\$} \mathsf{esc}$**:** *the convert algorithm is randomized, ittakes as input a hint for the last tag of a history. It returns a predicate escrow* esc *that prepares the history for audit evaluation.*

$\mathsf{VfEscrow}_{\mathsf{pk}}(\mathsf{esc}, \mathsf{tag}) \to 1/0$**:** *the verify escrow algorithm is deterministic, ittakes as input a predicate escrow* esc*, and the update tag* tag *used in the last update. It returns* 1 *if the inputs are consistent, and* 0 *otherwise.*

$\mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{esc}) \to \top/\perp$**:** *the decrypt algorithm is deterministic, ittakes as input the auditor's secret key* sk *and a predicate escrow. It returns* $\top$ *if the escrow contains values that satisfy the predicate* $P(t, \cdot)$*, where* t *is determined by* sk*.*

The notion of *correctness* covers the honest execution of the protocol. It ensures that: (1) the honestly generated key always verifies, (2) honestly updated histories of base commitments verify, and (3) the result of decrypted escrow is consistent with the evaluation of the predicate on the sum of update values.

Definition 7 (Correctness). *Let* BC *and* $P \in \mathcal{P}_{\mathbb{V}}$ *be as in Definition 6, and* $\lambda \in \mathbb{N}$*. A UPPB scheme for* (BC, P) *is* correct *if the following statements hold for all* $\mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda)$*,* $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp})$ *(remember these are implicit in all the algorithms):*

– *Full correctness: for all $t \in \mathbb{V}$, all poly-sized sequences of values $x_1, \ldots, x_n \in \mathbb{V}$ and $\mathfrak{r}_1, \ldots, \mathfrak{r}_n \in \mathbb{R}$:*

$$\Pr\left[\begin{array}{l|l} \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = 1 \wedge & (\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t) \\ \text{for all } i \in [n]: & \text{for } i \in [n]: \\ \quad \mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_i, \mathsf{tag}_i) = 1 \wedge & \quad (\mathsf{hint}_i, \mathsf{tag}_i) \upharpoonleft_{\$} \\ \quad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{i}) = 1 \wedge & \quad\quad \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{i-1}, \mathsf{tag}_{i-1}, x_i, \mathfrak{r}_i) \\ \quad \mathsf{VfEscrow}_{\mathsf{pk}}(\mathsf{esc}_i, \mathsf{tag}_i) = 1 \wedge & \quad \mathsf{esc}_i \xleftarrow{\$} \mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_i) \\ \quad \mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{esc}_i) = P(t, \sum_{j=1}^{i} x_j) & \quad \mathfrak{C}_i \leftarrow \mathfrak{Commit}(x_i; \mathfrak{r}_i) \end{array}\right] = 1$$

– *Update correctness: for all $\mathsf{pk}, \mathsf{hint}_0$ s.t. $\mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = 1$, and all $\mathsf{hint}_n, \{\mathsf{tag}_j, \mathfrak{C}_j\}_{i=1}^{n}$ such that $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_n, \mathsf{tag}_n) = 1$ and $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{n}) = 1$, and for all $x \in \mathbb{V}, \mathfrak{r} \in \mathbb{R}$:*

$$\Pr\left[\begin{array}{l|l} \mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_{n+1}, \mathsf{tag}_{n+1}) = 1 \wedge & (\mathsf{hint}_{n+1}, \mathsf{tag}_{n+1}) \upharpoonleft_{\$} \\ \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{n+1}) = 1 \wedge & \quad \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_n, \mathsf{tag}_n, x, \mathfrak{r}) \\ \mathsf{VfEscrow}_{\mathsf{pk}}(\mathsf{esc}_{n+1}, \mathsf{tag}_{n+1}) = 1 & \mathsf{esc}_{n+1} \xleftarrow{\$} \mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_{n+1}) \\ & \mathfrak{C}_{n+1} \leftarrow \mathfrak{Commit}(x; \mathfrak{r}) \end{array}\right] = 1$$

In both statements, the probability is taken over the random coins internally sampled by the randomized algorithms of UPPB.

4.1 Security Properties

We say that a history and predicate escrows are valid if they verify under a verifying public key. Valid histories and escrows must satisfy two properties – history binding and soundness.

History binding enforces that for any valid history, its prefix must also be valid, and no alternative prefix can ever be valid. This means that after verifying a commitment history one can use the last tag as a commitment to the whole history and is guaranteed the validity of each step in the history.

Definition 8 (History Binding). *A UPPB scheme for (BC, P) is* history binding *if for all PPT $\mathcal{A}$, it holds that $\Pr[\mathcal{G}_{\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$, where game $\mathcal{G}_{\mathcal{A}}(1^\lambda)$ is as follows:*

$\mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda)$; $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp})$
$(\mathsf{pk}, \mathsf{hint}_0, \{\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota}\}_{b \in \{0,1\}}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp})$
return $\mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = 1 \wedge$
  $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota}) = 1 \wedge$
  $\Big(\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota-1}) \neq 1 \vee$
    $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)}\}_{i=1}^{\iota}) = 1 \wedge$
    $\mathsf{tag}_\iota^{(0)} = \mathsf{tag}_\iota^{(1)} \wedge \exists i.\, (\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}) \neq (\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)})\Big)$

In practice, history binding prevents history manipulation: assuming an updater that produced tag as a "receipt" of their update is later approached by the regulator for presenting their esc, the updater will not be able to deceive the regulator by saying "this tag I produced for a different history". So history binding is crucial for "tracking back" the history of changes done to the esc; it enforces history linearity.

Soundness focuses on what VfEscrow and VfHistory functions mean *together*: (1) any verifying history "contains" a set of update values, and (2) if esc verifies w.r.t. the last tag of this history, it must decrypt to the value of predicate P evaluated on t and the sum of committed values in the history.

Definition 9 (Soundness). *A UPPB scheme for (BC, P) is* sound *if there exists a deterministic poly-time black-box extractor* Ext*, such that for all PPT $\mathcal{A}$:*

1. *Valid history can be explained in terms of base commitments: for all* $\iota > 0$,

$$\Pr\left[\begin{array}{l|l} \mathsf{VfKeyGen}(\mathsf{pk},\mathsf{hint}_0)=1 \wedge & \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda) \\ \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i,\mathfrak{C}_i\}_{i=1}^{\iota})=1 \wedge & (\mathsf{pp},\mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\mathfrak{pp}) \\ \mathfrak{C}_\iota \neq \mathfrak{Commit}(x_\iota,\mathfrak{r}_\iota) & (\mathsf{pk},\mathsf{hint}_0,\{\mathsf{tag}_i,\mathfrak{C}_i\}_{i=1}^{\iota}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ & (x_\iota,\mathfrak{r}_\iota) \leftarrow \mathsf{Ext}(\mathsf{td},\mathsf{tag}_\iota) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

2. *Decryption always reveals the predicate computed for the sum of update values: for all* $t \in \mathbb{V}$, $\iota > 0$,

$$\Pr\left[\begin{array}{l|l} \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i,\mathfrak{C}_i\}_{i=1}^{\iota})=1 \wedge & \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda) \\ \mathsf{VfEscrow}_{\mathsf{pk}}(\mathsf{esc}^\star,\mathsf{tag}_\iota)=1 \wedge & (\mathsf{pp},\mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\mathfrak{pp}) \\ \mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{esc}^\star) \neq P(t,\sum_{i=1}^{\iota} x_i) & (\mathsf{sk},\mathsf{pk},\mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t) \\ & (\mathsf{esc}^\star,\{\mathsf{tag}_i,\mathfrak{C}_i\}_{i=1}^{\iota}) \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{pk},\mathsf{hint}_0) \\ & \text{for } i \in [1,\iota]: \\ & \quad (x_i,\mathfrak{r}_i) \leftarrow \mathsf{Ext}(\mathsf{td},\mathsf{tag}_i) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

The extractor is the same in both clauses of the definition and works the same given the same inputs. This means that the two parts are composable: the extracted value in the second part satisfies $\forall i.\, \mathfrak{C}_i = \mathfrak{Commit}(x_i,\mathfrak{r}_i)$ with overwhelming probability. Together with the binding property of BC, this guarantees that any values x_ι that are opened, by revealing $\mathfrak{r}_\iota$ or that are used externally in proofs of knowledge about $\mathfrak{C}_\iota$, must be the same as that used to evaluate P.

The first part of soundness considers dishonest keys (emulating a view of a third party observing the history, e.g. on the bulletin board), while the second part has honest keys because it is viewed from the honest regulator's perspective.

Our *hiding* definitions provide privacy guarantees, capturing the following properties: (1) output of KeyGen does not leak the threshold value t (Definition 10), (2) tags do not leak the update value (Definition 11), (3) hints do not leak the update value, without sk (Definition 12), and (4) escrow values only leak the result of the evaluated predicate (Definition 13).

Threshold hiding states that it is computationally impossible to determine the threshold value t chosen upon key generation from the public key, without the secret key.

Definition 10 (Threshold Hiding). *A UPPB scheme for* (BC,P) *is* threshold hiding *if for all PPT* $\mathcal{A}$ *it holds that:*

$$\Pr\left[b \stackrel{?}{=} b^\star \;:\; \begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda);\ (\mathsf{pp},\cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\mathfrak{pp}) \\ (t_0,t_1) \leftarrow \mathcal{A}(\mathsf{pp}),\ b \xleftarrow{\$} \{0,1\} \\ (\cdot,\mathsf{pk},\mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t_b) \\ b^\star \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{hint}_0) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Tag hiding states that tags do not reveal any additional information than already revealed by $\mathfrak{C}$ itself.

Definition 11 (Tag Hiding). *A UPPB scheme for* (BC,P) *is* hiding in tags *if, for* $(\mathsf{pp},\mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\mathbb{V},\mathcal{P})$ *all* $t \in \mathbb{V}$, *all* pk, *all pairs* $(\mathsf{hint},\mathsf{tag})$ *such that* $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint},\mathsf{tag})=1$, *and for all* $x \in \mathbb{V}$, $\mathfrak{r} \in \mathbb{R}$, *there exists a PPT* $\mathcal{S}$ *such that:*

$$\left\{\mathsf{tag}' \mid (\cdot,\mathsf{tag}') \xleftarrow{\$} \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint},\mathsf{tag},x,\mathfrak{r})\right\} \stackrel{p}{=} \left\{\mathcal{S}(\mathsf{td},\mathsf{pk},\mathsf{tag},\mathfrak{C} := \mathfrak{Commit}(x,\mathfrak{r}))\right\}$$

where distributions are over the internal randomness of the Update *algorithm and the simulator. For the first update, this holds conditioned on* $\mathsf{hint} := \mathsf{hint}_0$, $\mathsf{tag} := \bot$.

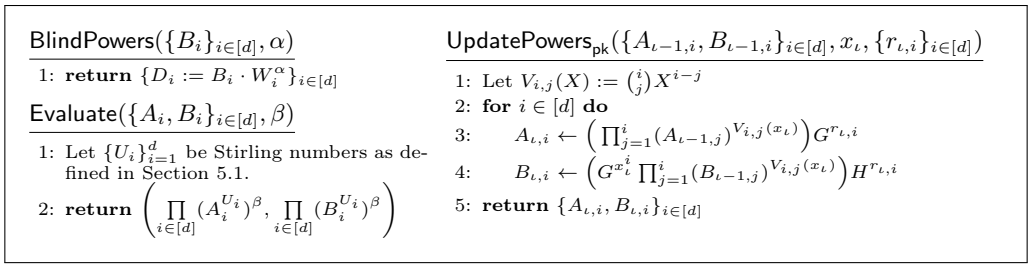

Fig. 1. Helper functions for the main uBlu protocol. The values $\{W_i\}_{i\in[d]}$ are independent bases, being part of the public parameters, and $\{V_{i,j}\}, \{U_i\}_{i\in[d]}$ are public values as defined in Section 5.1.

Note that the simulation-style definition here is dictated by tags being verifiable w.r.t. base commitments in histories. This allows composable reasoning: tags are hiding regardless of the base commitments hiding property; whereas IND-style definition would imply that the base scheme needs to be hiding which we avoid. Tag hiding also implies hiding for any *sequence* of tags, and thus for any history $\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}$: using $\mathcal{S}$ and $\{\mathfrak{C}_i\}$ we can simulate all the tags one by one, without any hints.

Hint hiding states that without the secret key, hints do not leak information update values.

Definition 12 (Hint Hiding). *A UPPB scheme for* (BC, P) *is* (computationally value) hiding in hints *if, for all* $t \in \mathbb{V}$ *and all PPT* $\mathcal{A}$*, it holds that* $\Pr[\mathcal{G}_{\mathcal{A}}(1^\lambda) = 1] \le 1/2 + \mathsf{negl}(\lambda)$*, where game* $\mathcal{G}_{\mathcal{A}}(1^\lambda)$ *is as follows:*

$\mathsf{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda)$; $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathsf{pp})$
$(\cdot, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t)$; $b \xleftarrow{\$} \{0,1\}$
$(\mathsf{hint}^\star, \mathsf{tag}^\star, x^{(0)}, x^{(1)}, \mathfrak{r}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \mathsf{hint}_0)$
$(\mathsf{hint}, \cdot) \xleftarrow{\$} \mathsf{Update}(\mathsf{hint}^\star, \mathsf{tag}^\star, x^{(b)}, \mathfrak{r})$
$b^\star \xleftarrow{\$} \mathcal{A}(\mathsf{hint})$
return $b^\star \stackrel{?}{=} b \wedge \mathsf{VfCommit}(\mathsf{hint}^\star, \mathsf{tag}^\star) \stackrel{?}{=} 1$

Blueprint hiding models that even with the knowledge of the secret key, the escrow esc does not leak anything about the values inside the history besides the predicate result itself.

Definition 13 (Blueprint Hiding). *A UPPB scheme for* (BC, P) *is* blueprint hiding *if there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT* $\mathcal{A}$*, it holds that* $\Pr[\mathcal{G}_{\mathcal{A}}(1^\lambda) = 1] \le 1/2 + \mathsf{negl}(\lambda)$*, where game* $\mathcal{G}_{\mathcal{A}}(1^\lambda)$ *is as follows:*

$(\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathbb{V}, \mathcal{P})$.
$(t, \mathsf{pk}, \mathsf{hint}_0, \{\mathsf{tag}_i, x_i, \mathfrak{r}_i\}_{i=1}^{\iota}, \mathsf{hint}_\iota) \xleftarrow{\$} \mathcal{A}(\mathsf{pp})$
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{esc} \leftarrow$ **if** $b = 0$ **then** $\mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_\iota)$ **else** $\mathcal{S}(\mathsf{td}, \mathsf{pk}, P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota)$
$b^\star \xleftarrow{\$} \mathcal{A}(\mathsf{esc})$
return $b^\star = b \wedge$
$\quad \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = 1 \wedge$
$\quad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{Commit}(x_i, \mathfrak{r}_i)\}_{i=1}^{\iota}) = 1 \wedge$
$\quad \mathsf{VfHint}(\mathsf{hint}_\iota, \mathsf{tag}_\iota) = 1$

5 uBlu: Efficient Realization of Updatable Blueprints

Our efficient uBlu construction is presented in Figures 2 (main algorithms), 1 (helper functions), and 3 (verification algorithms). The construction is instantiated with Pedersen commitment scheme PedersenC as a base commitment, and the predicate $P_d(T, X)$ that returns 1 if and only if x is in the range $\{t, t+1, \ldots, t+d-1\}$.

$\underline{\mathsf{Setup}(1^\lambda, \mathsf{pp})}$

% To ensure $\mathbb{G}_1$ is the same for Pedersen BCS and the pairing system
parse pp as $(\mathbb{G}_1, G, \mathfrak{H}, P_d)$
$\mathsf{pp}_{\mathsf{BLG}} \leftarrow \mathsf{BLG.Setup}(1^\lambda; \mathbb{G}_1, G)$
% Blinding factors for $\{D_i\}_{i=1}^d$
% d comes from the predicate P_d in pp
$\{W_i\}_{i\in[d]} \xleftarrow{\$} \mathbb{G}_1$
$(\mathsf{crs}_\Pi, \mathsf{td}_\Pi) \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{BLG}})$
$(\mathsf{crs}_{\Pi_\mathsf{u}}, \mathsf{td}_{\Pi_\mathsf{u}}) \leftarrow \Pi_\mathsf{u}^{\mathcal{L}_\mathsf{c}}.\mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{BLG}})$
$\mathsf{pp} \leftarrow (\mathsf{pp}, \mathsf{pp}_{\mathsf{BLG}}, \{W_i\}_{i\in[d]}, 0, P_d, \mathsf{crs}_\Pi, \mathsf{crs}_{\Pi_\mathsf{u}})$
$\mathsf{td} \leftarrow (\mathsf{td}_\Pi, \mathsf{td}_{\Pi_\mathsf{u}})$
return $(\mathsf{pp}, \mathsf{td})$

$\underline{\mathsf{KeyGen}(t)}$

$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$, $H \leftarrow G^{\mathsf{sk}}$
$\{r_{0,i}\}_{i=1}^d, r_\mathsf{t} \xleftarrow{\$} \mathbb{Z}_q$
for $i \in [d]$ **do**
% ElGamal encryptions of t^i
$A_{0,i} \leftarrow G^{r_{0,i}}$, $B_{0,i} \leftarrow G^{((-t)^i)} H^{r_{0,i}}$
$\mathfrak{T} \leftarrow \mathfrak{Commit}(t; r_\mathsf{t})$ % Pedersen $\mathfrak{T} = G^t \mathfrak{H}^{r_\mathsf{t}}$
$\mathfrak{X}_0 \leftarrow \mathfrak{Commit}(0; 0)$ % $\mathfrak{X}_0 = 1_{\mathbb{G}_1}$
$\mathfrak{A}_0 \leftarrow \mathfrak{Commit}(0; 0)$
$\mathsf{x_c} \leftarrow (H, \{A_{0,i}, B_{0,i}\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}_0, \mathfrak{A}_0)$
$\mathsf{w_c} \leftarrow \begin{pmatrix} t, r_\mathsf{t}, \{r_{0,i}\}_{i\in[d]}, \hat{x} := 0, \\ \hat{r}_\mathsf{x} := 0, \{r_{0,i} \cdot (0-t)\}_{i\in[d]}, \\ \alpha := 0, r_\alpha := 0 \\ \alpha \cdot (\hat{x} - t) := 0, r_\alpha(\hat{x} - t) := 0 \end{pmatrix}$
$\pi_\mathsf{c} \xleftarrow{\$} \Pi_\mathsf{u}^{\mathcal{L}_\mathsf{c}}.\mathsf{Prove}(\mathsf{x_c}, \mathsf{w_c})$
$\pi_{\mathsf{pk}} \xleftarrow{\$} \Pi^{\mathcal{L}_{\mathsf{pk}}}.\mathsf{Prove}((H, B_{0,1}, \mathfrak{T}), (\mathsf{sk}, t, r_{0,1}, r_\mathsf{t}))$
$\mathsf{pk} \leftarrow (H, \mathfrak{T}, \pi_{\mathsf{pk}})$
$\mathsf{hint}_0 \leftarrow (\{A_{0,i}, B_{0,i}\}_{i\in[d]}, \mathfrak{X}_0, \pi_\mathsf{c})$
return $(\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0)$

$\underline{\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, \mathsf{tag}_{\iota-1}, x_\iota; \mathfrak{r})}$

parse $\mathsf{tag}_{\iota-1}$ as $(\pi_{\mathsf{t},\iota-1}, \mathfrak{X}_{\iota-1})$
$r_{\mathsf{x},\iota} \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{hint}_\iota \leftarrow \mathsf{UpdateHint}(\mathsf{hint}_{\iota-1}, x_\iota, r_{\mathsf{x},\iota})$
$\mathfrak{C}_\iota \leftarrow \mathfrak{Commit}(x_\iota, \mathfrak{r})$
parse $\mathsf{hint}_{\iota-1}$ as $(\cdot, \mathfrak{X}_{\iota-1}, \cdot)$
parse hint_ι as $(\cdot, \mathfrak{X}_\iota, \cdot)$
$\pi_{\mathsf{t},\iota} \xleftarrow{\$} \Pi^{\mathcal{L}_\mathsf{t}}.\mathsf{Prove}(\begin{pmatrix} H, \\ \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \\ \mathfrak{C}_\iota, \pi_{\mathsf{t},\iota-1} \end{pmatrix}, \begin{pmatrix} x_\iota, \\ r_{\mathsf{x},\iota}, \\ \mathfrak{r}_\iota \end{pmatrix})$
$\mathsf{tag}_\iota \leftarrow (\pi_{\mathsf{t},\iota}, \mathfrak{X}_\iota)$
return $(\mathsf{hint}_\iota, \mathsf{tag}_\iota)$

$\underline{\mathsf{UpdateHint}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, x_\iota, r_{\mathsf{x},\iota})}$ (Helper)

parse $\mathsf{hint}_{\iota-1}$ as $\begin{pmatrix} \{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, \\ \mathfrak{X}_{\iota-1}, \pi_{c,\iota-1}, \end{pmatrix}$
$\{r_{\iota,i}\}_{i\in[d]} \xleftarrow{\$} \mathbb{Z}_q$
% $r_{\mathsf{x},\iota}$ is the input to UpdateHint
$\mathfrak{X}_\iota \leftarrow \mathfrak{X}_{\iota-1} \cdot \mathfrak{Commit}(x_\iota; r_{\mathsf{x},\iota})$
% $= \mathfrak{Commit}(\sum_{i\in[\iota]} x_i; \sum r_{\mathsf{x},i})$
$\{A_{\iota,i}, B_{\iota,i}\}_{i\in[d]} \leftarrow \mathsf{UpdatePowers}\begin{pmatrix} \{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, \\ x_\iota, \{r_{\iota,i}\}_{i\in[d]} \end{pmatrix}$
$\mathsf{x_c} \leftarrow \begin{pmatrix} H, \{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, \\ \mathfrak{T}, \mathfrak{X}_{\iota-1}, \mathfrak{A} := 1 \end{pmatrix}$
$\mathsf{w}_{\mathsf{upd,c}} \leftarrow \begin{pmatrix} x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \\ \alpha := 0, r_\alpha := 0 \end{pmatrix}$
$\pi_{\mathsf{c},\iota} \xleftarrow{\$} \Pi_\mathsf{u}^{\mathcal{L}_\mathsf{c}}.\mathsf{Update}(\pi_{\mathsf{c},\iota-1}; \mathsf{x_c}, \vec{T}_{\mathsf{upd}}(\mathsf{w}_{\mathsf{upd,c}}))$
$\mathsf{hint}_\iota \leftarrow (\{A_{\iota,i}, B_{\iota,i}\}_{i\in[d]}, \mathfrak{X}_\iota, \pi_{\mathsf{c},\iota})$
return hint_ι

$\underline{\mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_\iota)}$

% Partially rerandomize the hint
$(\{A_i, B_i\}_{i\in[d]}, \mathfrak{X}, \pi_\mathsf{c}) \leftarrow \mathsf{UpdateHint}_{\mathsf{pk}}(\mathsf{hint}_\iota, 0, 0)$
$\alpha, \beta, r_\alpha, r_\beta \xleftarrow{\$} \mathbb{Z}_q^\star$
$\mathfrak{A} \leftarrow \mathfrak{Commit}(\alpha; r_\alpha)$
$\mathfrak{B} \leftarrow \mathfrak{Commit}(\beta; r_\beta)$
$\{D_i\}_{i\in[d]} \leftarrow \mathsf{BlindPowers}(\{B_i\}_{i\in[d]}, \alpha)$
$(E_1, E_2) \leftarrow \mathsf{Evaluate}(\{A_i, B_i\}_{i\in[d]}, \beta)$
$\mathsf{x_c} \leftarrow (H, \{A_i, B_i\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}, \mathfrak{A} := 1)$
$\mathsf{w}_{\mathsf{upd,c}}\begin{pmatrix} x_\iota := 0, \{r_{\iota,i} := 0\}_{i\in[d]}, \\ r_{\mathsf{x},\iota} := 0, \alpha, r_\alpha \end{pmatrix}$
$\pi'_\mathsf{c} \xleftarrow{\$} \Pi_\mathsf{u}^{\mathcal{L}_\mathsf{c}}.\mathsf{Update}(\pi_\mathsf{c}; \mathsf{x_c}, T_{\mathsf{upd}}(\mathsf{w}_{\mathsf{upd,c}}))$
$\mathsf{w_e} \leftarrow (\alpha, r_\alpha, \beta, r_\beta)$
% U_i are the Stirling numbers
$\pi_\mathsf{e} \leftarrow \Pi^{\mathcal{L}_\mathsf{e}}.\mathsf{Prove}\left(\begin{pmatrix} E_1, E_2, \mathfrak{B}, \mathfrak{A}, \\ \prod A_i^{U_i}, \prod D_i^{U_i} \end{pmatrix}, \mathsf{w_e}\right)$
$\mathsf{esc} \leftarrow \begin{pmatrix} E_1, E_2, \pi_\mathsf{e}, \pi'_\mathsf{c}, \mathfrak{X}, \\ \{A_i, D_i\}_{i\in[d]}, \mathfrak{A}, \mathfrak{B} \end{pmatrix}$
return esc

$\underline{\mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{esc})}$

parse esc as $(E_1, E_2, \cdot)$
$M \leftarrow E_1^{-\mathsf{sk}} * E_2$
% ElGamal decryption $M = G^{\beta P(t,\hat{x})}$
return $[M \stackrel{?}{=} 1_{\mathbb{G}_1}]$

Fig. 2. Our uBlu protocol for $\mathsf{BC} = \mathsf{PedersenC} = (\mathfrak{Setup}, \mathfrak{Commit})$, the predicate $P_d(T, X) = \left[\prod_{\delta=0}^{d-1}(X - T - \delta) \stackrel{?}{=} 0\right]$, and CH20 updatable NIZK. Main Algorithms.

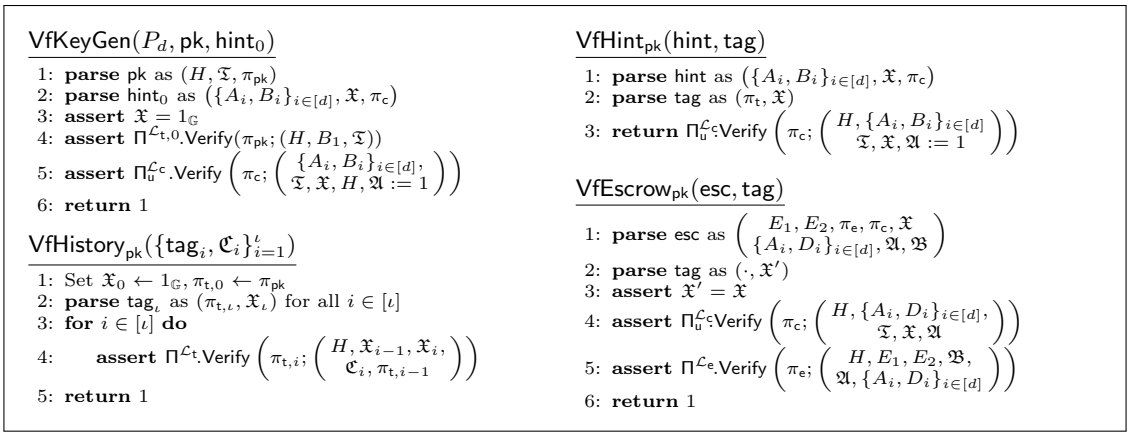

Fig. 3. Verification Algorithms for the uBlu protocol. Continuation of Fig. 2.

It also uses the updatable proof system Π_{u} instantiated by CH20, and a straight-line simulation-extractable Π (instantiated by Fiat-Shamir transformed Σ-protocols for proofs of equality of discrete logarithm representations with witness encryption). Next, we proceed with an overview that gradually builds intuition on the techniques employed in our construction, and conclude with how to achieve privacy and soundness.

5.1 Achieving Updatable Functionality

The Predicate. We consider the predicate that returns $\top$ if the value x concealed in the escrow is *above* the given threshold t. For efficiency, we limit the check to a reasonable interval, i.e. the escrow is only decryptable if the value x is in $[t, t+d-1]$ for a small value d: for generic X and T the predicate is defined as

$$P_d = P_d(T, X) = \left[\prod_{\delta=0}^{d-1}(X - T - \delta) \stackrel{?}{=} 0\right] \in \{0, 1\}. \tag{1}$$

Clearly, when evaluated, the predicate $P_d(t, x)$ returns 1 if and only if x is in the "critical range" $[t, t+d-1]$ on which the core polynomial evaluates to 0. The polynomial is built in such a way as to allow for efficient updates as discussed next.

Setup and Key Generation. The setup takes as input a group $\mathbb{G}$ generated by $\mathfrak{Setup}$ of the base commitment scheme (Pedersen), together with generator $G_1 = G$ and $\mathfrak{H}$. It finishes the bilinear group setup, creating $\mathsf{pp}_{\mathsf{BLG}}$ w.r.t. $\mathbb{G}_1$. It also sets up common reference strings and trapdoors for the NIZK proofs (more on this in Section 5.3). Most importantly, it generates d random masking values $W_1, \ldots, W_d \stackrel{\$}{\leftarrow} \mathbb{G}$ which are needed for blinding ElGamal ciphertexts in the hints.

To run the key generation process, the regulator needs to choose a threshold value t. In a nutshell, KeyGen samples $\mathsf{sk} \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and computes its corresponding DH public key $H \leftarrow G^{\mathsf{sk}}$. The public key consists of H and some additional data to prove consistency. The key generation process additionally returns a hint consisting of: 1) a sequence of d ElGamal ciphertexts encrypting the powers of $t, t^2, \ldots, t^d$ under the public key H; 2) a NIZK consistency proof $\pi_{\mathsf{c},0}$ for these powers; and 3) additional dummy information for correct hint formatting. This allows public verifiability of the correctness of the key generation procedure and of the hints in epoch $\iota = 0$.

Hints and Updatability. Hints are used to update the value concealed in an updatable blueprint. In our construction, updates perform the addition of a new known value x_ι to the already concealed one (which is possibly unknown).

At each epoch ι hints have two main components: 1) a sequence of d ElGamal ciphertexts in the exponent for a public key H (see Equation (5) for definition); and 2) a base Pedersen commitment $\mathfrak{X}$ for the accumulated value $\hat{x} = \sum_{i\in[\iota]} x_i$ embedded in the ciphertexts (the meaning of "embedded" will become clear in a moment).

To understand the mechanics behind updatability, we need to look back at the polynomial in our predicates P_d (see Equation (1)). This is a product of expressions consisting of a sum of a private/unknown value $(X - T)$ and a public value δ. Hence, the polynomial in Equation (1) can be written as a linear combination of powers of $(X - T)$, namely:

$$\prod_{\delta=0}^{d-1}((X-T)-\delta) = \sum_{i=0}^{d} U_i (X-T)^i \tag{2}$$

where all U_i are well-determined public coefficients that depend solely on i and d.[7] By the binomial theorem, it is possible to rewrite terms on the right side of Equation (2) as:

$$(X - T + Y)^i = \sum_{j=0}^{i} \left(\binom{i}{j} Y^{i-j} \right) \cdot (X-T)^j. \tag{3}$$

Equation (3) shows that we can always build $((x + y) - t)^i$ linearly from (lower powers) $(x - t)^j$ and y. This property is exploited by the UpdatePowers helper function (Fig. 1) to compute hints for $x + y$ as a linear combination of the old hints values (dependent only on x and t), and values dependent only on the new (known) input $y = x_\iota$. For easy reference, we define the y-dependent values as

$$V_{i,j}(y) := \binom{i}{j} y^{i-j} \in \mathbb{Z}_q. \tag{4}$$

Recall that hints contain a sequence of ElGamal ciphertexts, in our construction the initial hint, produced at epoch $\iota = 0$ during key generation, contains encryptions of powers of $-t^i$, i.e. $\{A_{0,i} = G^{r_{0,i}}, B_{0,i} = G^{(-t)^i} H^{r_{0,i}}\}_{i\in[d]}$ where the $\{r_{0,i}\}_{i\in[d]}$ are the random values and t is the regulator's secret threshold for the predicate. Hints get progressively updated (as we show momentarily) into the following form:

$$\{A_{\iota,i} = G^{\hat{r}_{\iota,i}}, B_{\iota,i} = G^{(\hat{x}-t)^i} H^{\hat{r}_{\iota,i}}\}_{i\in[d]}, \tag{5}$$

where $\hat{r}_{0,i}$ denotes accumulated randomness, $\hat{x} = \sum_{i\in[\iota]} x_i$ is the committed value accumulated at epoch ι. (Note that when $\iota = 0$ the ciphertexts conceal the value $(0 - t)^i$).

By linear homomorphism, it is possible to add a new known value y to the quantity $(x - t)$ that is concealed in the hints of the previous epoch via the expression:

$$B_{\iota+1,i} = \prod_{j=0}^{i} B_{\iota,j}^{V_{i,j}(y)} = G^{\sum_{j=0}^{i}(x-t)^j \cdot V_{i,j}(y)} \cdot (H^{r_{\iota,i}})^{V_{i,j}(y)} = G^{(x+y-t)^i} H^{r_{\iota,i} V_{i,j}(y)}$$

where the last equality comes for Equation (3) and the definition in Equation (4) and $B_{0,0} = G^{t^0} H^0$. Noting that $V_{i,0}(y) = y^i$, each $B_{\iota+1,i}$ can be computed solely from hints of epoch ι with $j > 0$ in the following way:

$$B_{\iota+1,i} = G^{y^i} \prod_{j=1}^{i} B_{\iota,j}^{V_{i,j}(y)}$$

where we isolate the $j = 0$ term $G^{y^i} = B_{\iota,0}^{V_{i,0}(y)}$ to the left.

[7] To be precise, the U_i are the Stirling coefficients, i.e., Stirling numbers of the first kind are defined as the coefficients in the expansion of the falling factorial polynomial $(x)_n = \prod_{i=0}^{n-1}(x - i) = \sum_{k=0}^{n} s(n, k)x^k$, and have closed form $s(n, k) = \begin{bmatrix} n \\ k \end{bmatrix} = (-1)^{n-k} \cdot \sum_{1\le i_1<\ldots<i_{n-k}\le n-1} (\prod_{j=1}^{n-k} i_j)$ [55].

Preparing Hints for Escrow. This procedure is performed by the Escrow algorithm (Fig. 2). Intuitively, the ElGamal ciphertexts are extracted from the hint, and "evaluated". The Evaluate algorithm raises both ciphertext components to $U_i \cdot \beta$, where β is a random non-zero value (used for masking non-escrow data), and the U_i are the Stirling coefficients described in Section 5.1. Specifically:

$$E_1 = \prod_{i\in[d]} (A_i^{U_i})^{\beta} = G^{\beta\cdot(\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i)}$$

$$E_2 = \prod_{i\in[d]} (B_i^{U_i})^{\beta} = \left(G^{\sum\limits_{i\in[d]} U_i(x-t)^i} H^{\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i} \right)^{\beta} = G^{\beta\cdot(\prod\limits_{\delta=0}^{d-1}(x-t-\delta))} H^{\beta\cdot(\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i)}$$

where the last equality comes for Equation (2). As a result, the holder of the ElGamal secret key cannot efficiently decrypt the evaluated ciphertext. Decryption corresponds to solving the discrete logarithm problem since β is random (and unknown to the authority) unless the ciphertext encrypts the value "0". Note that we built the predicate in such a way that the ciphertext encodes 0 only on the roots of the polynomial, which correspond to values in the "critical range". The Escrow procedure outputs the predicate escrow which, in addition to the evaluated ElGamal ciphertext, contains additional components needed to prove consistency and verify the correctness of the procedure.

Testing the Predicate. This procedure is run by the regulator and simply attempts to decrypt the ciphertext (E_1, E_2) using the secret key sk corresponding to the ElGamal encryption public key H. This entails computing $M = E_2 \cdot (E_1)^{-\mathsf{sk}}$, which by construction is $M = G^{\beta\cdot(\prod_{\delta=0}^{d-1}(X-T-\delta))}$, where the reader should recognize the core polynomial of the predicate (see Equation (1)). Note that β (unknown to the regulator) acts as a random mask that prevents efficient decryption whenever the polynomial evaluates to a value other than 0. This makes M gibberish unless the predicate P_d evaluates to 1 (the polynomial evaluates to 0), which yields $M = G^0 = 1_{\mathbb{G}}$.

5.2 Achieving Privacy

Up to this point, we discussed the correctness of our construction. Now we focus on how to achieve privacy, i.e., the authority only learns $P_d(t, x)$ and nothing else, and updaters learn nothing about the concealed value.

The IND-CPA property of ElGamal ciphertexts $\{A_i = G^{r_i}, B_i = G^{(x-t)^i} H^{r_i}\}_{i\in[d]}$ prevents updaters from seeing the concealed value. The regulator however can obtain $\{G^{(x-t)^i}\}_{i\in[d]}$ by decrypting the ciphertexts, and even though encoding values in the exponent makes generic decryption inefficient, it does not prevent the regulator from obtaining x by when the encoded values are in small, predictable ranges (which is the setting of our application). To hide x properly, updaters will *blind* hints before sending them to the regulator with the escrow.

The blinding is performed by BlindPowers and consists of multiplying each B_i component by a value W_i^{α}, where the $\{W_i\}_{i=1}^{d}$ are public group elements generated upon system setup, and α is a freshly sampled random value. Specifically, blinded ciphertexts are of the form: $\{A_i, D_i := B_i \cdot W_i^{\alpha}\}_{i\in[d]}$. To achieve efficient updatable proofs, the α component will be zero while the hints are updated, which means parties will exchange unblinded hints; and α will only be set while the hints are converted to an escrow (more details on this in the upcoming description of updatable proofs).

5.3 Achieving Soundness Using NIZKs

Intuitively speaking, soundness means then whenever the data (primarily hints and escrows) is valid, it must be "good" – bind to the history, contain only updates that are relevant to commitments, etc. As of now, hints and escrows can be malformed and lack these guarantees. We overcome these issues by employing NIZKs to ensure data correctness.

Our construction employs four kinds of proofs. The *key proof* (π_{pk}) will show that the public key was built correctly. The *consistency proof* (π_{c}) will show the consistency of all the components in a hint, and it will be updatable (details of which are the main technical contribution of the construction). The *trace proof* (π_{t}) will show that the new hint — obtained updating a hint from the previous epoch — is computed correctly and that the update value is the same as in the external commitment $\mathfrak{C}$. Trace proofs are included in tags tag and form the trace. The *escrow proof* (π_{e}) will show that the escrow esc was produced correctly from a tag and its hint.

Key Proof (π_{pk}). During the key generation phase, the regulator produces the ElGamal encryptions of the powers of the threshold (as explained in Section 5.1), in particular, we will use the first ciphertext $(A_{0,1}, B_{0,1}) := (G^{r_{0,1}}, G^t H^{r_{0,1}})$ that is a standard ElGamal encryption of t for the auditor's public key H. In addition, the regulator computes a Pedersen commitment to the threshold $\mathfrak{T} = G^t \mathfrak{H}^{r_{\mathsf{t}}}$ (which is included in the public key pk). The public key additionally contains π_{pk} that proves knowledge of the sk corresponding to the ElGamal public key $H = G^{\mathsf{sk}}$, and knowledge of a threshold value t and randomnesses that realize the public components $B_{0,1}$ (contained in hint_0) and $\mathfrak{T}$. Specifically, the language $\mathcal{L}_{\mathsf{pk}}$ is defined by:

$$\begin{array}{c} \mathsf{x} = (H, B_{0,1}, \mathfrak{T}) \in \mathbb{G}^3 \\ \mathsf{w} = (\mathsf{sk}, t, r_{0,1}, r_{\mathsf{t}}) \in \mathbb{Z}_q^4 \end{array} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 & 0 & 0 \\ 0 & G & H & 0 \\ 0 & G & 0 & \mathfrak{H} \end{bmatrix}.$$

where witness here and in the following is highlighted in gray.

Consistency Proof (π_{c}). This proof shows that all the hint values (including the $\{A_{0,i}, B_{0,i}\}$ produced by KeyGen) are indeed encodings of $(\hat{x} - t)^i$, where $\hat{x}$ and t are the current accumulated value and the original threshold selected by the regulator. It works for both unblinded (B_i) and blinded (D_i) hints. This proof is produced (1) originally by the regulator in KeyGen to prove the consistency of powers in $\mathsf{hint}_{\mathsf{pk}}$, and (2) by updating parties to prove that the hints they send further are still consistent, (3) by updating parties to prove to the regulator that the blinded hints in the escrow are consistent. The consistency proof will always refer to $\mathfrak{T}$ (the Pedersen commitment to the threshold included in pk) to make sure that the witness t used in all proof iterations is the same as the initial one.

At epoch ι, the consistency proof π_{c} proves the following statement: for an instance

$$\mathsf{x} = (H, \{A_{\iota,i}, D_{\iota,i}\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}_\iota, \mathfrak{A}) \in \mathbb{G}^{2d+4}$$

there exists a witness

$$\mathsf{w} = \left(\begin{array}{c} t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}, \alpha, r_\alpha, \{\hat{r}_{\iota,i}\}_{i\in[d]}, \\ (\hat{x}_\iota - t), \{r_{\iota,i}(\hat{x}_\iota - t)\}_{i\in[d-1]}, \alpha(\hat{x}_\iota - t), r_\alpha(\hat{x}_\iota - t) \end{array} \right) \in \mathbb{Z}_q^{2d+8}$$

such that the following relations are satisfied:

1. $\mathfrak{T} = G^{t} \mathfrak{H}^{r_{\mathsf{t}}}$ (r_{t} is the randomness used to create $\mathfrak{T}$, the Pedersen commitment to the threshold)
2. $\mathfrak{X}_\iota = G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$ (Pedersen commitment to $\hat{x}_\iota$, the accumulated value)
3. $\mathfrak{A}_\iota = G^{\alpha} \mathfrak{H}^{r_\alpha}$ (Pedersen commitment to the randomness for blinding factors)
4. $A_{\iota,1} = G^{\hat{r}_{\iota,1}}$ ($r_{\iota,i}$ is the randomness used to create the ElGamal ciphertext)
5. $D_{\iota,1} = G^{\hat{x}_\iota - t} H^{\hat{r}_{\iota,1}} W_1^{\alpha}$ (the blinded ciphertext encrypts $(\hat{x} - t)$)
6. $\forall\ i \in [2, d]$:
   (a) $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$
   (b) $D_{\iota,i} = (D_{\iota,i-1})^{\hat{x}_\iota - t} (H^{-1})^{\hat{r}_{\iota,i-1}(\hat{x}_\iota - t)} H^{\hat{r}_{\iota,i}} (W_{i-1}^{-1})^{\alpha(\hat{x}_\iota - t)} W_i^{\alpha}$
7. *Witness products* (needed for step 6):
   (a) $1 = G^{\hat{x}_\iota} (G^{-1})^{t} (G^{-1})^{\hat{x}_\iota - t}$

(b) $1 = \mathfrak{A}_{\iota}^{\hat{x}_\iota - t}(G^{-1})^{\alpha(\hat{x}_\iota - t)}(\mathfrak{H}^{-1})^{r_\alpha(\hat{x}_\iota - t)}$

(c) $1 = A_{\iota,i}^{\hat{x}_\iota - t}(G^{-1})^{\hat{r}_{\iota,i}(\hat{x}_\iota - t)}$, for $i \in [d-1]$:

The complexity of this formula is due to the fact that we need to prove the relationship between the powers of $(\hat{x} - t)^i$, which we do recursively. Note that we do not store powers as additional witnesses; the only witness is the first power $(\hat{x}_\iota - t)$.

When simplified, the recursive formulas reduce to the following four relations:

$$\mathfrak{T} = G^{t}\mathfrak{H}^{r_{\mathsf{t}}} \qquad \mathfrak{X} = G^{\hat{x}}\mathfrak{H}^{\hat{r}_{\mathsf{x}}}$$
$$\mathfrak{A} = G^{\alpha}\mathfrak{H}^{r_\alpha} \qquad (A_i, D_i) = (G^{\hat{r}_i}, G^{(\hat{x}-t)^i}H^{\hat{r}_i}W_i^{\alpha}) \text{ for } i \in [d]$$

As briefly mentioned before, we use α in two different ways depending on the scenario: (1) while updating the hints blinding is disabled: users will set $\alpha = r_\alpha = 0$, and thus $\mathfrak{A} = 1$; thus D_i will be actually just B_i; (2) while creating escrow, the blinding values α, r_α will be introduced, $\mathfrak{A} \neq 1$ will be sent to the regulator, but α, r_α will not, which will ensure hiding of the blinding approach.

Therefore, to verify the consistency proof, the party (user or regulator) needs an instance x, which consists of the original $D_{0,1}$ produced during key generation; a collection of ciphertexts $\{(A_{\iota,i}, D_{\iota,i})\}_{i\in[d]}$ (unblinded or blinded); a tag tag_ι containing a commitment $\mathfrak{X}$ to the accumulated $\hat{x}_\iota$; and a special commitment $\mathfrak{A}$ to the blinding randomness α (either trivial $\mathfrak{A} = 1$ for users, or nontrivial for regulator).

Consistency proofs are instantiated by Π_{u}, which is linear in the size of the hints but is also updatable, meaning that the proof for new hints is a transformation of the previous consistency proof. Practically, this is quite efficient, since otherwise consistency proofs would need to be aggregated, and hints would thus grow in size; this is especially expensive given that the consistency language is linear in d. Because of updatability, no updating party (except for the regulator, who creates the initial proof) ever knows the whole witness "contained" in the proof.

Updating Hints and Consistency Proof. The consistency proof language $\mathcal{L}_{\mathsf{c}}$ is structured in such a way, that it supports a transformation that we will call T_{upd}, which can change all the necessary witnesses, including our target aggregated commitment value $\hat{x}_\iota$. To fit within the algebraic language updatability framework, we must be able to represent the new instance and witness as a linear combination of the old instance and witness values correspondingly.

We first start with the instance, which implicitly defines $T_{\mathsf{xm}}, T_{\mathsf{xa}}$:

- Using the (plaintext) update value x_ι, sample rerandomisation factors $r_{\iota,i}$, and compute the new hints:
  - $A_{\iota,i} = \Big(\prod_{j=1}^{i}(A_{\iota-1,j})^{V_{i,j}(x_\iota)}\Big)G^{r_{\iota,i}}$.
  - $B_{\iota,i} = G^{x_\iota^i}\Big(\prod_{j=1}^{i}(B_{\iota-1,j})^{V_{i,j}(x_\iota)}\Big)H^{r_{\iota,i}}$, where $G^{x_\iota^i}$ covers the role of implicit $(B_{\iota-1,0})^{V_{i,0}(x_\iota)}$.
- Sample $r_{\mathsf{x},\iota}$, update the tag commitment $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1}G^{x_\iota}\mathfrak{H}^{r_{\mathsf{x},\iota}}$.
- Sample α, r_α, create the special commitment $\mathfrak{A} = G^{\alpha}\mathfrak{H}^{r_\alpha}$ (optionally, or still assume $\mathfrak{A} = 1$).

Next, we show how to update the witness, which implicitly defines $T_{\mathsf{wm}}, T_{\mathsf{wa}}$:

$$\hat{x}_\iota := \hat{x}_{\iota-1} + x_\iota \qquad \hat{x}_\iota - t := \hat{x}_{\iota-1} - t + x_\iota$$

$$\hat{r}_{\iota,i} := \sum_{j=1}^{i}\hat{r}_{\iota-1,i}\cdot V_{i,j}(x_\iota) + r_{\iota,i}$$

$$\hat{r}_{\iota,i}(\hat{x}_\iota - t) := \sum_{j=1}^{i}\hat{r}_{\iota-1,i}(\hat{x}_{\iota-1} - t)\cdot V_{i,j}(x_\iota) + \sum_{j=1}^{i}\hat{r}_{\iota-1,i}\cdot x_\iota \cdot V_{i,j}(x_\iota) + r_{\iota,i}\cdot(\hat{x}_{\iota-1} - t) + r_{\iota,i}x_\iota$$

$$\hat{r}_{\mathsf{x},\iota} := \hat{r}_{\mathsf{x},\iota-1} + r_{\mathsf{x},\iota}$$

$$\hat{\alpha} := \alpha \qquad \hat{\alpha}(\hat{x}_\iota - t) := \alpha\cdot(\hat{x}_{\iota-1} - t) + \alpha\cdot x_\iota$$

$$\hat{r}_\alpha := r_\alpha \qquad \hat{r}_\alpha(\hat{x}_\iota - t) := r_\alpha\cdot(\hat{x}_{\iota-1} - t) + r_\alpha\cdot x_\iota$$

The language transformation T_{upd} is formally a set of matrices $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ as implicitly defined above, that is parameterized by a vector of update values $\mathsf{w}_{\mathsf{upd,c}} = (x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \alpha, r_\alpha)$, where all the "product witnesses" can be defined in terms of this tuple.

Note that we do not describe the last four witnesses as "accumulatable" — if we try to update w with (α, r_α) more than once, $\hat{\alpha}$ will not be equal to the sum of previous α unlike e.g. $\hat{r}_{\mathsf{x},\iota}$. This is due to our setup: (1) we apply T_{upd} incrementally parameterized with $(x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \alpha = 0, r_\alpha = 0)$ with *blinding turned off*; (2) and then, given $\alpha = r_\alpha = 0$ and $\mathfrak{A} = 1$, we can *introduce blinding*, applying T_{upd} parameterized with $\alpha, r_\alpha \neq 0$ only once. This separation is a result of a deeper limitation of Π_{u}, discussion of which we defer to Appendix B:

Theorem 1 (Validity of T_{upd} (Informal)). *The transformation T_{upd} is valid w.r.t. $\mathcal{L}_{\mathsf{c}}$, and the Π_{u} proof system for $\mathcal{L}_{\mathsf{c}}$ satisfies update completeness and derivation privacy w.r.t. T_{upd} when applied according to the two distinct parametrizations described above.*

Proof. See Definition 18, Theorem 3, and Lemma 1 in Appendix B. □

Trace Proof ($\boldsymbol{\pi_{\mathsf{t}}}$). Trace proofs are small aggregatable proofs that allow parties to linearise their updates. At epoch ι, the party performing an update with local value x_ι will prove the following statement. For an instance $\mathsf{x} = (H, \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \mathfrak{C}_\iota, \pi_{\mathsf{t},\iota-1})$ (where $\mathfrak{C}_\iota = G^{x_\iota}\mathfrak{H}^{\mathfrak{r}}$), there exists a witness $\mathsf{w} = (x_\iota, r_{\mathsf{x},\iota}, \mathfrak{r}_\iota)$ such that:

1. $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} \cdot G^{x_\iota}\mathfrak{H}^{r_{\mathsf{x},\iota}}$ (the new tag is computed the form the previous one, and the updating information is completely known to the updater).
2. $\mathfrak{C}_\iota = \mathfrak{Commit}(x_\iota, \mathfrak{r}_\iota)$(the updated value x_ι is the same as in $\mathfrak{C}_\iota$).

Note that the value $\pi_{\mathsf{t},\iota-1}$ is in the instance, and thus bound by the NIZK being a signature of knowledge, but it does not appear in any equations. In practice, this translates with hashing the additional value when computing a Fiat-Shamir challenge, but not using it otherwise. This proof will be instantiated with a standard non-updatable Π Σ-protocol.

As a potential future-work extension of our scheme, one can consider parties including their signatures on these elements, to sign the update act, which can be used for extending updater accountability w.r.t. the regulator.

Escrow Proof ($\boldsymbol{\pi_{\mathsf{e}}}$). This proof is produced upon conversion of a hint into an escrow esc. The esc contains an ElGamal encryption $E = (E_1, E_2)$ of $\beta \cdot P_d(\hat{x}, t)$ for some masking value β (random), an escrow proof, a consistency proof (rerandomized, and with α introduced), and information needed to check the proofs: a commitment $\mathfrak{X}$ to the accumulated value, a commitment $\mathfrak{B}$ to the randomness β, a commitment $\mathfrak{A}$ to the introduced accumulated blinding exponent α, and, most importantly, *blinded* ElGamal ciphertexts $\{A_i, D_i\}_{i\in[d]}$ (with α).

The escrow proof for $\mathcal{L}_{\mathsf{e}}$ proves the following statement. For an instance $\mathsf{x} = (E_1, E_2, \mathfrak{B}, \mathfrak{A}, \prod A_i^{U_i}, \prod D_i^{U_i})$, there exists a witness $\mathsf{w} = (\alpha, r_\alpha, \beta, r_\beta, \beta\alpha, r_\beta\alpha)$ such that the following conditions are satisfied:

1. $\mathfrak{A} = G^{\alpha}\mathfrak{H}^{r_\alpha}$
2. $\mathfrak{B} = G^{\beta}\mathfrak{H}^{r_\beta}$
3. $1 = \mathfrak{B}^{\alpha}(G^{-1})^{\beta\alpha}(\mathfrak{H}^{-1})^{r_\beta\alpha}$
4. $E_1 = \prod_i (A_i^{U_i})^{\beta}$
5. $E_2 = \prod_i (D_i^{U_i})^{\beta} \cdot \prod_i (W_i^{-U_i})^{\beta\alpha}$

The language is compact, so the proof π_{e} can be created from scratch, and since it doesn't need to be updatable performance-wise we can also use standard Π as a proof system.

5.4 Security of the uBlu Construction

The main security statement of our construction can be summarized as follows.

Theorem 2. *The* uBlu *protocol w.r.t.* $(\mathsf{PedersenC}, P_d(T, X))$ *introduced in Section 5 is secure according to the security definitions 8-13 (all in Section 4.1) under: hiding and binding of* PedersenC*; DDH in* $\mathbb{G}_1$ *that in particular implies ElGamal IND-CPA; completeness, strong simulation-extractability, and ZK of* Π*; and (update) completeness, soundness, derivation privacy, and ZK of* Π_{u}.

Proof (Summary). Due to the lack of space, the theorems and their corresponding proofs are deferred to Appendix C, while here we present their summary and main intuition.

History Binding (Theorem 5) is proven by unfolding simulation-extractability of Π, instantiating trace proofs, along the history.

Soundness (Theorem 6) reduces to KS of Π, soundness of Π_{u}, and binding of PedersenC. The first part of soundness is a trivial application of Π KS, while for the second part, we need to unpack all the NIZKs, using the fact that they are "connected" by binding commitments, to arrive at the statement about correct decryptability of esc.

Threshold Hiding (Theorem 7) is, first, by ZK of both Π and Π_{u} — and after proofs in $\mathsf{pk}, \mathsf{hint}_0$ are simulated, we reduce the property directly to IND-CPA of ElGamal, holding under DDH in $\mathbb{G}_1$. *Hint hiding* (Theorem 9) is very similar to threshold hiding: it holds by IND-CPA of Pedersen (A_i, D_i are hiding), ZK of the NIZKs, but also by derivation privacy of Π_{u}, since Update uses Π_{u}.Update internally. Similarly, *Tag hiding* (Theorem 8) is a direct consequence of ZK of trace proof and hiding of PedersenC.

Blueprint Hiding (Theorem 10) holds under DDH in $\mathbb{G}_1$, security of PedersenC, ZK of Π, Π_{u}, KS of Π, and soundness of Π_{u}. The hardest part in simulating esc is arguing that encryption (E_1, E_2) has a "correct" form, which is similar to the soundness proof.

□

5.5 Adding Range Proofs for Improved Accuracy

For efficiency, construction approximates the predicate $[x > t]$ with a polynomial that has roots in the range $[t, t+d-1]$, for a reasonably small value d. For the approximation to be accurate, however, we need to ensure that an update does not jump over the range or cause an overflow. In other words, we need an extra range proof ensuring that the value x_ι added during the update is not too large.

Range proofs can be conveniently integrated into our protocol which already exposes $\mathfrak{C}_i = G^{x_\iota}\mathfrak{H}^{\mathfrak{r}_i}$ for exactly this purpose. Among the existing approaches to range proofs on Pedersen commitments, we recall Bulletproofs [15] or "adjusted" Pedersen commitments and square decomposition [27]. These are efficient, with the latter only requiring a constant amount of exponentiations and group elements in the proof.

Here we present a much simpler approach than the aforementioned: committing to bits of x_ι, and using Π to prove that (1) $\mathfrak{C}$ contains bit-reconstructed values; and (2) commitments are actually to the bit values $\in \{0, 1\}$. The latter can be done as follows: given $C = G^x H^r$ for H being chosen uniformly at random, note that condition $x \in \{0, 1\}$ is equivalent to $(x-1)x = 0$, therefore it is enough to prove that $C^{x-1} = H^{r'}$ for some (known to the prover but private) r'. This requires $O(\log(d))$ exponentiations and group elements, which is practically efficient for our choice of d.

With this in mind, every updater can only "adjust" the aggregated rating $\hat{x}$ by $x_\iota \in [0, d]$, which makes our uBlu construction a proper score aggregation system. Other more complicated predicates can be proven similarly about x_ι; commitment $\mathfrak{C}_i$ is used precisely for this kind of external integration of a uBlu scheme with other applications.

6 Scheme Extensions

The uBlu protocol we presented in this section is designed for a limited class of predicates (range predicates) to keep its construction simple. In this section we show how to generalize our construction to support arbitrary polynomial predicates and non-binary escrow values. We also discuss additional features such as the distributed decryption of predicate escrows.

Disjoint Ranges Polynomial Predicates. Our uBlu construction can easily be adapted to the case of polynomials $P(T, X)$ that capture $r > 1$ disjoint ranges (e.g. $[t_1, t_1 + d_1]$ *and* $[t_2, t_2 + d_2]$). The cost is r sets of hints that are linear in the corresponding range length.

Arbitrary Polynomial Predicates. Our uBlu protocol targets the "range polynomial" $P_d(T, X)$, that is zero in $[t, t+d-1]$. We leverage the special structure of P_d to design hints linear in d. The algebraic trick used in our protocol to construct and update hints can be generalized to *any* polynomial predicate $P(T, X)$ at the cost of a *quadratic* number of hints, each encoding $\{x^i t^j\}_{i,j:i+j\leq d}$, which are updatable in a similar way our linear hints are. An example of using non-interval predicates could be designing P to encode a certain few excluded revealing points, for example representing a certain blocklist of public key hashes.

In detail, assuming $P(T, X) = \sum_{i,j} C_{i,j} X^i T^j$, we can always construct the evaluation of the updated polynomial $P(T, X+Y) = \sum_{i,j} C_{i,j}(X+Y)^i T^j$ from the hints, if we can transform old hints $\{(x^i t^j)\}_{i,j}$ into the new ones $\{(x+y)^i t^j\}_{i,j}$. The latter is always possible since $(x+y)^i t^j = \sum_{k=0}^{i} \binom{i}{k} y^{i-k}(t^j x^k)$ is a linear combination of the previous $(t^j x^k)$, which are known. Quadratic number of hints makes many algorithms much less efficient and generally imposes much stricter upper bounds on d.

Multi-Variate Polynomial Predicates. It is possible to extend updatable blueprints to support evaluations of predicates described by multi-variate polynomials, by repeating the protocol in parallel independently for each dimension i (as if it were n protocol instances).

For example, assume the polynomial is of the form $P(T, X_1, \ldots, X_n) = \sum_i^n P_i(T, X_i)$, and we are running uBlu in parallel for each P_i independently, however with the same starting commitment to t. Updates and the evaluation can still be done in a very similar manner. The only significant difference now is the escrow proof created during Escrow will now have to prove the evaluation of joint $P(T, X_1, \ldots, X_n)$ instead of independent $P_i(T, X_i)$ as in the basic scheme. This, however, can still be encoded as an algebraic relation — now it will take $\{\mathfrak{A}_i\}_{i=1}^n$ (one element per each "parallel" run), which will lead to the introduction of $\{\beta\alpha_i\}_{i=1}^s$ witnesses and the statement for E_2 will have to change to $E_2 = (\prod_{i=1}^d \prod_{j=1}^n D_{j,i}^{U_i})^{\beta} \cdot (\prod_{i=1}^d W_i)^{\sum_{j=1}^n \beta\alpha_j}$.

As a slightly different but instructive example, let us consider evaluating the polynomial corresponding to the computation of the squared Euclidean distance. In this case, we manage the two-variate polynomial $P(T_X, T_Y, X, Y) = (X - T_X)^2 + (Y - T_Y)^2$, where (T_X, T_Y) are the coordinates of a departure point, and X, Y are updates on the respective latitude and longitude deviation from the previous (or initial) position. The regulator needs to commit to T_X, T_Y separately. In order to evaluate $P(t_X, t_Y, x + x', y + y')$ it is sufficient to notice that:

$$\begin{aligned} &((x' + x) - t_X)^2 + ((y' - y) - t_y)^2 = \\ =\ & x'^2 + x^2 + t_X^2 + 2(x'(x + t_X) + xt_X) + y'^2 + y^2 + t_Y^2 + 2(y'(y + t_Y) + yt_Y) \\ =\ & x'^2 + 2x'(x + t_X) + (x^2 + t_X^2 + 2xt_X) + y'^2 + 2y'(y + t_Y) + (y^2 + t_Y^2 + 2yt_Y) \end{aligned}$$

where in the last line, all terms in parenthesis are computable from the previous hints.

Non-binary Predicate Value. In some applications, *e.g.*, when users are anonymous, it is desirable for $\mathsf{Escrow}_{\mathsf{pk}}$ to return not only a binary value but also information about the user (*e.g.* their identity) to enable further investigations. Our uBlu scheme can be modified to achieve this functionality. The core idea is to give $\mathsf{Escrow}_{\mathsf{pk}}$ a piece of extra information y, so that instead of returning an encryption of $\beta P(x)$, the algorithm returns $(\beta_1 P(x), \beta_2 P(x) + y)$ for random β_1, β_2, which in case $P(x) \neq 0$ produces two random points and in the case $P(x) = 0$ (*i.e.* if the escrow is decryptable) returns $(0, y)$. The extra information y then can be proven to be added correctly by integrating a pre-computed commitment $\mathfrak{C}_y$ to y (e.g. coming from an external identity scheme) into the escrow proof, which now will not only attest to the correctness of the evaluation w.r.t. (x, y) but also that y is coming from the designated commitment $\mathfrak{C}_y$.

Another simple but useful variation of this idea is to return y not based on $\mathfrak{C}_y$, but as a function of x. Consider two polynomials P_1, P_2, where polynomial P_1 is binary and defining the decryptability as

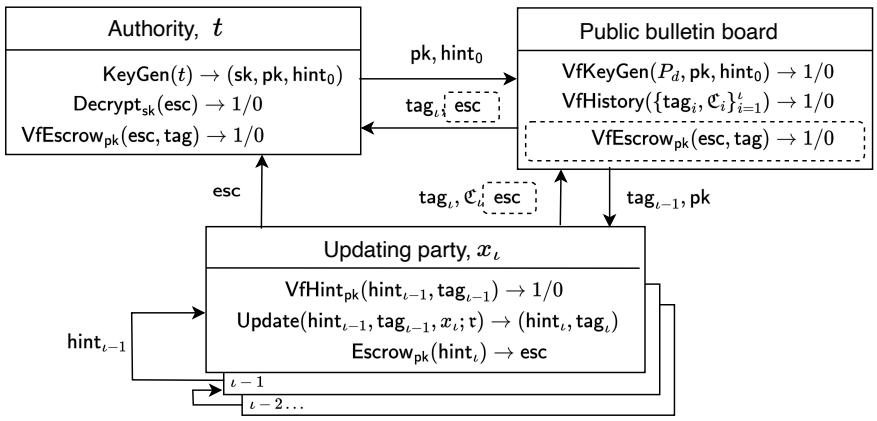

Fig. 4. Illustration of a potential usage of updatable blueprints in the blockchain setting. Dotted boxes illustrate optional communication and computation depending on the scenario.

before, and $P_2(x) = y$ is defining what the result in case of decryption will be. Then $\mathsf{Escrow}_{\mathsf{pk}}$ can return $(\beta_1 P_1(x), \beta_2 P_1(x) + P_2(x))$, revealing the non-binary y that depends on x. Since P_1 and P_2 are different predicates, they will have different sets of hints; but assuming that $\deg P_2$ is low, the previous paragraph explained how we can extend the uBlu scheme to support polynomial reconstructions for arbitrary P_2. Again, here the escrow proof must be modified to attest to the correct evaluation of the escrow.

Distributed Decryption. We observe that while we explicitly consider the KeyGen and Decrypt algorithms to be run by a single party, these steps could be distributed or thresholded using standard techniques [41]. This would have the advantage of both, separating the attack surface for the secret decryption key sk and of requiring the explicit consensus of decryption by a quorum of different key share holders. That is, it would only make it possible for the auditors to discover exactly which update caused the change in the predicate value (by running Decrypt after each update) if sufficiently many of them agree that they should learn this.

7 Applications of Updatable Privacy-Preserving Blueprints

In this section we present applications for uBlu schemes such as accountable privacy-preserving blockchains, anti-money laundering (AML), and privacy-preserving proximity testing for unknown locations.

We believe that updatable blueprints can find multiple uses in ensuring accountability for privacy-preserving blockchain applications beyond cryptocurrencies. The security properties of uBlu and their flexibility as a building block in modular constructions make them easy to integrate into various applications without undermining existing privacy properties.

In this setting, we assume access to a public append-only ledger with support for a Turing complete scripting language (*e.g.* Ethereum or Cardano). We consider a model with an auditor who ensures the setup of the system and who is able to decrypt the escrow. Observe that such an auditor could be distributed as discussed in Sec. 6. With such a ledger, any auditor can run Setup and deploy a smart contract that embeds the verification functions: VfKeyGen, VfHint, VfHistory and VfEscrow. The auditor can then run KeyGen for a specific choice of predicate threshold t, and send the public key and hint_0 to the initial 0-commitment hint_0 to the smart contract, which validates these values using VfKeyGen and stores them if they are valid. External parties can now interact off-chain, constructing updates by running the Update algorithm and sending the resulting hint to the next updating party, after validating the last hint using VfHint. When mandated by the application (potentially after every update), the updating party will publish its tag to the ledger, which will form a consistent history that will be validated by the smart contract using VfHistory. When necessary,

authority can request a party to Escrow a certain hint corresponding to a point in history, obtain escrow esc off-chain, and check if the corresponding update has caused a predicate evaluation of 1; this would also obviate any need for off-chain communication between authority and updating parties. Alternatively, esc can be sent to the smart contract directly, which would be more expensive in terms of escrow validation but can be used to prove to a third party that a certain escrow evaluated to 1. We illustrate this in Fig. 4.

This approach is generic and allows for many application-specific variations. For example, it is possible the augment the smart contract logic to ensure that only a permissible set of parties are able to update the blueprint. Or, one could limit the quantity of updates any specific updating party is allowed to make; etc.

In case one wishes to simplify the smart contract and save gas, it is also possible to have the smart contract *only* store protocol elements but not run any of the verification algorithms, instead optimistically assuming them to be correct and putting the responsibility of validation off-chain, to the auditor or any other observer. That is, an approach similar to optimistic roll-ups in the blockchain space. Next, observe that each call to a smart contract is signed by the caller and stored on the ledger, linking it to the caller's public key. Thus, by augmenting the smart contract with Decentralized Identifiers [68] it would also be possible to publicly audit which parties correctly follow the protocol.

7.1 Traditional AML

Money laundering is the process of concealing the origins of funds obtained illegally by changing their origin to one considered legitimate and is estimated to constitute 2-3% of the GDP in the US alone [64, Chap. 2]. Traditional banks mostly do AML by manually inspecting an account if its suspiciousness score, expressing how suspicious or risky the account is, is above a certain threshold [7]. The score is first computed from private account metadata and updated based only on local account activity. As this precludes using valuable information held by other banks, e.g., the reputation of parties transferring money into the account, this results in a very high false-positive rate.

The updatable blueprint scheme can be used in traditional AML by allowing banks to secretly communicate the user's suspiciousness score, or credit score, from the sending bank to the receiving bank (or vice versa). Concretely, by having the *sending* bank use the score of the *sending* account to compute a value with which the *recipient's* score should be increased. That is, an auditor generates keys (KeyGen) for an updatable blueprint for each account in each bank. Then each bank checks these public keys using VfKeyGen and uses Update to add each account's *base* score to the corresponding hint_0 (initially containing 0). When an account holder makes a transaction, the *sending* bank updates the hint of the *receiving* account holder with an amount computed from the quantity of the transaction and the base score of the sender. The receiving bank then runs VfHint and VfHistory to validate the update. At certain time intervals, each bank runs Escrow to create the predicate escrow esc to be shared with the auditor along with the tags that have been constructed as part of the updates since the last time the auditor did a check. The authority can then run Decrypt (along with VfEscrow) to check if an account should be flagged. We illustrate the overall flow of this in Fig. 7.1.

We observe that the scheme will allow the banks to get a much more accurate suspiciousness score on each account without any bank leaking the base score of their account holders. When manual inspection of a flagged account results in the bank needing to report the customer to the authorities, it is possible for the bank to also share all the hints used to compute the updates. This allows the authority to validate the entire history of transactions. Finally, we also note that this can be enhanced using the idea of distributing the secret key in Sec. 6 to make the powers of the authority distributed.

7.2 Blockchain AML

The previously discussed approach to privacy-preserving AML based on updatable blueprints can be generalized to the blockchain setting. Most decentralized cryptocurrencies (*e.g.* Bitcoin, Ethereum, and Cardano) allow anyone to perform transactions with no privacy guarantees, publicly revealing the transaction graph and transferred amounts. While this seemingly makes AML easy, it is cheap and easy to create new accounts,

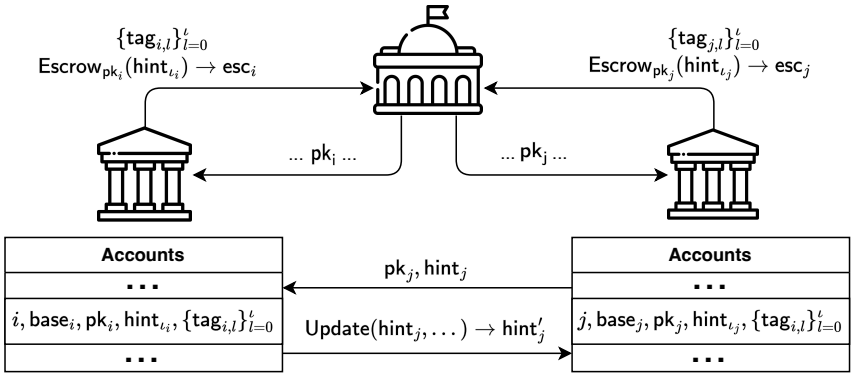

Fig. 5. Illustration of the interaction between parties in the AML use-case.

and it is easy to perform layering through many accounts and various "mixing services", mixing tokens from many different sources [62]. Hence, although the transaction graph is public, AML is not simple. This issue is exacerbated by privacy-preserving cryptocurrencies such as Zether [14], which is built on top of a smart contract system, or ZCash [10], Monero, and Dash, which incorporate privacy in their basic design. These schemes aim at hiding all transaction data, making AML even harder.

In order to reach a compromise between privacy and AML, several different ideas have been proposed by authors in recent years. Some authors [65, 3, 29] suggest an escrow system where anonymity and privacy can be broken if suspicious or illegal activities occur. Another approach is to specify a small spending limit per client which they can use every month for anonymous payments. After the client has made more transactions than covered by this budget, any future transactions can be traced [71, 70].

We now discuss how updatable blueprints could be used to achieve private spending limits on ledgers with Turing complete smart contracts supporting private transactions.

Private Spending Limit. In this setting, we assume that the underlying ledger supports private transactions and the goal of an authority is to be able to find out which (if any) users go over their private spending limit. This is achieved by having a smart contract store an updatable blueprint hint associated with each user and validate that each private transaction is supported by a Update to the user's updatable blueprint history consistent with the transaction amount, followed by a Escrow. If an update causes the escrow to be decryptable, it leaks the user's real identity to the auditor via the extension discussed in Sec. 6 that allows for decryption to output specific messages.

With this in mind, we describe the use case step by step: An authority starts by setting up new public and private keys and an initial hint_0 using KeyGen for an updatable blueprint scheme. Public key pk and hint_0 are going to be "shared" by *all* users with a private spending limit. Next, we require each user to get their real-world identity y validated by the auditor and linked to their blockchain account. This is done by having the user post a commitment $Y = G^y H^{r_y}$ to the smart contract and prove to the auditor that the committed message is indeed their identity y. The commitment from each user will then get aggregated into a Merkle tree, and stored in the smart contract. Next, each user gets a copy of hint_0 from the auditor which they will store, and use for their first Update. For each private transaction, the user runs Update and Escrow to make a new hint, tag, and esc which will open to y in case of successful decryption. The latter is based on the extension of Sec. 6. They then construct a zero-knowledge proof that the value y that esc will open to, if decrypted, corresponds to the value of a leaf commitment on a path in the smart contract's Merkle tree (without revealing the path). Finally, the user also constructs a proof of equality between the value in the base commitment of the update and the value of the private transaction they wish to carry out. The smart contract now validates the proofs, the private transaction, and whether the updates are valid ($\mathsf{VfHistory}$, $\mathsf{VfEscrow}$).

7.3 Privacy-Preserving Geofencing for Unknown Locations

As seen in Sec. 6 updatable blueprints can also be used to evaluate the (squared) Euclidean distance. This is relevant in privacy-preserving location-based services, and updatability provides the feature to update one's location according to movement w.r.t. a departure point or a previous location. This might be of use for military training, where soldiers are dropped in unknown territories and can track their own movements relative to the starting point, without knowing the exact coordinates of their location. Another use case could be proximity-based testing w.r.t. a moving object, here instead of sending the new location at every time interval, one sends relative update values, which are much shorter (in size) and may increase privacy should the device sending updates be compromised.

8 Instantiation and Performance

In this section, we summarise the implementation and performance details of the uBlu construction. We consider an instantiation of our updatable blueprints in the "standard model", with a caveat that we apply the Fiat-Shamir heuristic for the non-updatable proof system Π. The updatable proof system Π_{u}, which is instantiated by the CH20 [26] NIZK, is non-interactive by design.

We realize the base commitment BC using Pedersen's scheme [60] over $\mathbb{G}_1$ of a practical type III pairing friendly elliptic curve, where source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ have no efficiently computable isomorphism. While we do not consider a specific curve, we remark that several curves could be used, such as Barreto-Naehrig [5] or BLS12-381 [4]. For more details on choice of curves, and performance estimates, especially w.r.t. NIZKs, see Appendix D.

Table 1. Complexity of our uBlu construction in terms of the number P of pairing operations and E_1/E_2 the number of multiplicative group scalar exponentiations in $\mathbb{G}_1/\mathbb{G}_2$. Constant d is defining the "explosion" range $[t, t+d-1]$. Value ι_{cur} stands for the current epoch (history length).

Algorithm	$\#P$	$\#E_1$	$\#E_2$
Setup	0	$O(1)$	$O(1)$
KeyGen	0	$9d + O(1)$	$4d + O(1)$
Update	0	$4d^2 + O(d)$	$1.5d^2 + O(d)$
Escrow	0	$14d + O(1)$	$4d + O(1)$
Decrypt	0	$O(1)$	0
VfKeyGen	$2d + O(1)$	$10d + O(1)$	$6d + O(1)$
VfHint	$2d + O(1)$	$10d + O(1)$	$6d + O(1)$
VfHistory	0	$O(\iota_{\mathsf{cur}})$	$O(\iota_{\mathsf{cur}})$
VfEscrow	$2d + O(1)$	$10d + O(1)$	$6d + O(1)$

Table 2. Size complexity of the different components of our uBlu construction in terms of the number of group elements (or elements of equivalent size).

Object	pp	sk	pk	hint	tag	esc
$\#\mathbb{G}_1$	$O(d^2)$	1	$O(1)$	$4d + 5$	$O(1)$	$4d + O(1)$
$\#\mathbb{G}_2$	$O(1)$	0	0	$2d + 8$	0	$2d + 8$

Asymptotic Summary. We summarize the computational complexity of the different algorithms in Table 1 and the sizes of the different components in Table 2. We assume that the public combinatorial values (binomial coefficients for $V_{i,j}$ and Stirling coefficients for U_i) are pre-computed, which requires total auxiliary storage of $2d^2$ elements (added into the cost for Setup).

Given that most constants, as we discussed in the previous paragraphs, are quite low (for UpdatePowers it is 1, for updating the proofs it is ≈ 3), and having a slower BLS12-381 curve in mind, we roughly estimate

that the conservative choice of d could be about 200-400, in order to achieve a latency in running Update of at most 1 second. When proof verification time is a concern, as in the blockchain environments, it makes sense to consider lower degrees, e.g. pairings in VfHint for $d = 45$ will take about 100 ms. These being rough estimates, in practice, many optimizations are possible, such as parallelizing the (row-independent) quadradic computations and using efficient Multi-Scalar-Multiplication (MSMs), so we expect concrete performance to be significantly better. Nevertheless, the estimates clearly show that the system is practical, the NIZK overhead is comparably low, and given that hints are not aggregated due to updatability of Π_{u}, the NIZK-induced communication overhead is quite optimal w.r.t. what is absolutely necessary to produce a hint update.

References

[1] T. Acar and L. Nguyen. "Revocation for Delegatable Anonymous Credentials". In: *PKC 2011*. Ed. by D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi. Vol. 6571. LNCS. Springer, Heidelberg, Mar. 2011, pp. 423–440. DOI: 10.1007/978-3-642-19379-8_26.

[2] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann. "Privacy-preserving auditable token payments in a permissioned blockchain system". In: *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*. ACM, 2020, pp. 255–267. DOI: 10.1145/3419614.3423259. URL: https://doi.org/10.1145/3419614.3423259.

[3] A. Barki and A. Gouget. *Achieving privacy and accountability in traceable digital currency*. Cryptology ePrint Archive, Report 2020/1565. https://eprint.iacr.org/2020/1565. 2020.

[4] P. S. L. M. Barreto, B. Lynn, and M. Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees". In: *SCN 02*. Ed. by S. Cimato, C. Galdi, and G. Persiano. Vol. 2576. LNCS. Springer, Heidelberg, Sept. 2003, pp. 257–267. DOI: 10.1007/3-540-36413-7_19.

[5] P. S. L. M. Barreto and M. Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order". In: *SAC 2005*. Ed. by B. Preneel and S. Tavares. Vol. 3897. LNCS. Springer, Heidelberg, Aug. 2006, pp. 319–331. DOI: 10.1007/11693383_22.

[6] J. Bartusek, S. Garg, A. Jain, and G. Policharla. "End-to-End Secure Messaging with Traceability Only for Illegal Content". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by C. Hazay and M. Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 35–66. DOI: 10.1007/978-3-031-30589-4_2. URL: https://doi.org/10.1007/978-3-031-30589-4_2.

[7] C. Baum, J. H. yu Chiang, B. David, and T. K. Frederiksen. *SoK: Privacy-Enhancing Technologies in Finance*. Cryptology ePrint Archive, Report 2023/122. https://eprint.iacr.org/2023/122. 2023.

[8] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. "Randomizable Proofs and Delegatable Anonymous Credentials". In: *CRYPTO 2009*. Ed. by S. Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 108–125. DOI: 10.1007/978-3-642-03356-8_7.

[9] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *CRYPTO 2014, Part II*. Ed. by J. A. Garay and R. Gennaro. Vol. 8617. LNCS. Springer, Heidelberg, Aug. 2014, pp. 276–294. DOI: 10.1007/978-3-662-44381-1_16.

[10] E. Ben-Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[11] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive composition and bootstrapping for SNARKS and proof-carrying data". In: *45th ACM STOC*. Ed. by D. Boneh, T. Roughgarden, and J. Feigenbaum. ACM Press, June 2013, pp. 111–120. DOI: 10.1145/2488608.2488623.

[12] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud. "Signatures on Randomizable Ciphertexts". In: *PKC 2011*. Ed. by D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi. Vol. 6571. LNCS. Springer, Heidelberg, Mar. 2011, pp. 403–422. DOI: 10.1007/978-3-642-19379-8_25.

[13] S. Bowe, J. Grigg, and D. Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. https://eprint.iacr.org/2019/1021. 2019.

[14] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. "Zether: Towards Privacy in a Smart Contract World". In: *FC 2020*. Ed. by J. Bonneau and N. Heninger. Vol. 12059. LNCS. Springer, Heidelberg, Feb. 2020, pp. 423–443. DOI: 10.1007/978-3-030-51280-4_23.

[15] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[16] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. *Proof-Carrying Data without Succinct Arguments*. Cryptology ePrint Archive, Report 2020/1618. https://eprint.iacr.org/2020/1618. 2020.

[17] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *CRYPTO 2021, Part I*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[18] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Recursive Proof Composition from Accumulation Schemes". In: *TCC 2020, Part II*. Ed. by R. Pass and K. Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 1–18. DOI: 10.1007/978-3-030-64378-2_1.

[19] I. Cascudo, I. Damgård, B. David, N. Döttling, R. Dowsley, and I. Giacomelli. "Efficient UC Commitment Extension with Homomorphism for Free (and Applications)". In: *ASIACRYPT 2019, Part II*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11922. LNCS. Springer, Heidelberg, Dec. 2019, pp. 606–635. DOI: 10.1007/978-3-030-34621-8_22.

[20] D. Catalano, D. Fiore, and I. Tucker. "Additive-Homomorphic Functional Commitments and Applications to Homomorphic Signatures". In: *ASIACRYPT 2022, Part IV*. Ed. by S. Agrawal and D. Lin. Vol. 13794. LNCS. Springer, Heidelberg, Dec. 2022, pp. 159–188. DOI: 10.1007/978-3-031-22972-5_6.

[21] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. "Malleable Proof Systems and Applications". In: *EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 281–300. DOI: 10.1007/978-3-642-29011-4_18.

[22] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. "Malleable Signatures: New Definitions and Delegatable Anonymous Credentials". In: *CSF 2014 Computer Security Foundations Symposium*. Ed. by A. Datta and C. Fournet. IEEE Computer Society Press, 2014, pp. 199–213. DOI: 10.1109/CSF.2014.22.

[23] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. "Succinct Malleable NIZKs and an Application to Compact Shuffles". In: *TCC 2013*. Ed. by A. Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 100–119. DOI: 10.1007/978-3-642-36594-2_6.

[24] D. Chaum, C. Crépeau, and I. Damgård. "Multiparty Unconditionally Secure Protocols (Abstract) (Informal Contribution)". In: *CRYPTO'87*. Ed. by C. Pomerance. Vol. 293. LNCS. Springer, Heidelberg, Aug. 1988, p. 462. DOI: 10.1007/3-540-48184-2_43.

[25] A. Connolly, P. Lafourcade, and O. Perez-Kempner. "Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes". In: *PKC 2022, Part I*. Ed. by G. Hanaoka, J. Shikata, and Y. Watanabe. Vol. 13177. LNCS. Springer, Heidelberg, Mar. 2022, pp. 409–438. DOI: 10.1007/978-3-030-97121-2_15.

[26] G. Couteau and D. Hartmann. "Shorter Non-interactive Zero-Knowledge Arguments and ZAPs for Algebraic Languages". In: *CRYPTO 2020, Part III*. Ed. by D. Micciancio and T. Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 768–798. DOI: 10.1007/978-3-030-56877-1_27.

[27] G. Couteau, M. Klooß, H. Lin, and M. Reichle. "Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments". In: *EUROCRYPT 2021, Part III*. Ed. by A. Canteaut and F.-X. Standaert. Vol. 12698. LNCS. Springer, Heidelberg, Oct. 2021, pp. 247–277. DOI: 10.1007/978-3-030-77883-5_9.

[28] I. Damgård, B. M. David, I. Giacomelli, and J. B. Nielsen. "Compact VSS and Efficient Homomorphic UC Commitments". In: *ASIACRYPT 2014, Part II*. Ed. by P. Sarkar and T. Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 213–232. DOI: 10.1007/978-3-662-45608-8_12.

[29] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi. "Balancing Privacy and Accountability in Blockchain Identity Management". In: *CT-RSA 2021*. Ed. by K. G. Paterson. Vol. 12704. LNCS. Springer, Heidelberg, May 2021, pp. 552–576. DOI: 10.1007/978-3-030-75539-3_23.

[30] I. Damgard, M. Geisler, and M. Kroigard. "Homomorphic encryption and secure comparison". In: *International Journal of Applied Cryptography* 1.1 (2008), pp. 22–31.

[31] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO 2012*. Ed. by R. Safavi-Naini and R. Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[32] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. "Cryptography against Continuous Memory Attacks". In: *51st FOCS*. IEEE Computer Society Press, Oct. 2010, pp. 511–520. DOI: 10.1109/FOCS.2010.56.

[33] M. F. Esgin, R. K. Zhao, R. Steinfeld, J. K. Liu, and D. Liu. "MatRiCT: Efficient, Scalable and Post-Quantum Blockchain Confidential Transactions Protocol". In: *ACM CCS 2019*. Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM Press, Nov. 2019, pp. 567–584. DOI: 10.1145/3319535.3354200.

[34] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. "On the Non-malleability of the Fiat-Shamir Transform". In: *INDOCRYPT 2012*. Ed. by S. D. Galbraith and M. Nandi. Vol. 7668. LNCS. Springer, Heidelberg, Dec. 2012, pp. 60–79. DOI: 10.1007/978-3-642-34931-7_5.

[35] M. Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *CRYPTO 2005*. Ed. by V. Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 152–168. DOI: 10.1007/11535218_10.

[36] J. Frankle, S. Park, D. Shaar, S. Goldwasser, and D. J. Weitzner. "Practical Accountability of Secret Processes". In: *USENIX Security 2018*. Ed. by W. Enck and A. P. Felt. USENIX Association, Aug. 2018, pp. 657–674.

[37] G. Fuchsbauer. "Commuting Signatures and Verifiable Encryption". In: *EUROCRYPT 2011*. Ed. by K. G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 224–245. DOI: 10.1007/978-3-642-20465-4_14.

[38] N. Gama et al. "Detecting money laundering activities via secure multi-party computation for structural similarities in flow networks". In: *Real World Cryptography*. 2020. URL: https://www.youtube.com/watch?v=4hryY6cMPaM&t=2558s.

[39] C. Ganesh, H. Khoshakhlagh, M. Kohlweiss, A. Nitulescu, and M. Zając. "What Makes Fiat–Shamir zkSNARKs (Updatable SRS) Simulation Extractable?" In: *International Conference on Security and Cryptography for Networks*. Springer. 2022, pp. 735–760.

[40] J. A. Garay, B. Schoenmakers, and J. Villegas. "Practical and Secure Solutions for Integer Comparison". In: *PKC 2007*. Ed. by T. Okamoto and X. Wang. Vol. 4450. LNCS. Springer, Heidelberg, Apr. 2007, pp. 330–342. DOI: 10.1007/978-3-540-71677-8_22.

[41] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20.1 (Jan. 2007), pp. 51–83. DOI: 10.1007/s00145-006-0347-3.

[42] *Gnark Benchmarks*. https://hackmd.io/@gnark/eccbench. Accessed: 2023-07-25.

[43] O. Goldreich, S. Micali, and A. Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th ACM STOC*. Ed. by A. Aho. ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[44] S. Goldwasser and S. Park. *Public Accountability vs. Secret Laws: Can They Coexist?* Cryptology ePrint Archive, Report 2018/664. https://eprint.iacr.org/2018/664. 2018.

[45] M. Green, G. Kaptchuk, and G. V. Laer. "Abuse Resistant Law Enforcement Access Systems". In: *EUROCRYPT 2021, Part III*. Ed. by A. Canteaut and F.-X. Standaert. Vol. 12698. LNCS. Springer, Heidelberg, Oct. 2021, pp. 553–583. DOI: 10.1007/978-3-030-77883-5_19.

[46] J. Groth and A. Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *EUROCRYPT 2008*. Ed. by N. P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24.

[47] C. Hébant, D. H. Phan, and D. Pointcheval. "Linearly-Homomorphic Signatures and Scalable Mix-Nets". In: *PKC 2020, Part II*. Ed. by A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas. Vol. 12111. LNCS. Springer, Heidelberg, May 2020, pp. 597–627. DOI: 10.1007/978-3-030-45388-6_21.

[48] M. Khalili, D. Slamanig, and M. Dakhilalian. "Structure-Preserving Signatures on Equivalence Classes from Standard Assumptions". In: *ASIACRYPT 2019, Part III*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11923. LNCS. Springer, Heidelberg, Dec. 2019, pp. 63–93. DOI: 10.1007/978-3-030-34618-8_3.

[49] A. Kiayias, M. Kohlweiss, and A. Sarencheh. "PEReDi: Privacy-Enhanced, Regulated and Distributed Central Bank Digital Currencies". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by H. Yin, A. Stavrou, C. Cremers, and E. Shi. ACM, 2022, pp. 1739–1752. DOI: 10.1145/3548606.3560707. URL: https://doi.org/10.1145/3548606.3560707.

[50] T. Kim and J. Jeong. "Extended Tower Number Field Sieve with Application to Finite Fields of Arbitrary Composite Extension Degree". In: *PKC 2017, Part I*. Ed. by S. Fehr. Vol. 10174. LNCS. Springer, Heidelberg, Mar. 2017, pp. 388–408. DOI: 10.1007/978-3-662-54365-8_16.

[51] M. Kohlweiss, A. Lysyanskaya, and A. Nguyen. "Privacy-Preserving Blueprints". In: *Cryptology ePrint Archive* (2022). URL: https://eprint.iacr.org/2022/1536.

[52] M. Kohlweiss, A. Lysyanskaya, and A. Nguyen. "Privacy-Preserving Blueprints". In: *EUROCRYPT 2023, Part II*. Ed. by C. Hazay and M. Stam. Vol. 14005. LNCS. Springer, Heidelberg, Apr. 2023, pp. 594–625. DOI: 10.1007/978-3-031-30617-4_20.

[53] M. Kohlweiss, A. Lysyanskaya, and A. Nguyen. "Privacy-Preserving Blueprints". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by C. Hazay and M. Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 594–625. DOI: 10.1007/978-3-031-30617-4_20. URL: https://doi.org/10.1007/978-3-031-30617-4_20.

[54] M. Kohlweiss and M. Zając. *On Simulation-Extractability of Universal zkSNARKs*. Cryptology ePrint Archive, Report 2021/511. https://eprint.iacr.org/2021/511. 2021.

[55] J. Konvalina. "A unified interpretation of the binomial coefficients, the Stirling numbers, and the Gaussian coefficients". In: *The American Mathematical Monthly* 107.10 (2000), pp. 901–910.

[56] A. Kothapalli, S. Setty, and I. Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*. Cryptology ePrint Archive, Report 2021/370. https://eprint.iacr.org/2021/370. 2021.

[57] U. M. Maurer. "Unifying Zero-Knowledge Proofs of Knowledge". In: *AFRICACRYPT 09*. Ed. by B. Preneel. Vol. 5580. LNCS. Springer, Heidelberg, June 2009, pp. 272–286.

[58] *MIRACL Benchmarks*. https://github.com/miracl/MIRACL/blob/master/docs/miracl-explained/benchmarks.md. Accessed: 2023-07-25.

[59] N. Narula, W. Vasquez, and M. Virza. "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers". In: *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*. Ed. by S. Banerjee and S. Seshan. USENIX Association, 2018, pp. 65–80. URL: https://www.usenix.org/conference/nsdi18/presentation/narula.

[60] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO'91*. Ed. by J. Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[61] P. de Perthuis and D. Pointcheval. "Two-Client Inner-Product Functional Encryption with an Application to Money-Laundering Detection". In: *ACM CCS 2022*. Ed. by H. Yin, A. Stavrou, C. Cremers, and E. Shi. ACM Press, Nov. 2022, pp. 725–737. DOI: 10.1145/3548606.3559374.

[62] A. Pertsev, R. Semenov, and R. Storm. *Tornado Cash Privacy Solution, version 1.4*. https://web.archive.org/web/20211026053443/https://tornado.cash/audits/TornadoCash_whitepaper_v1.4.pdf. Dec. 2019.

[63] C. Rackoff and D. R. Simon. "Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack". In: *CRYPTO'91*. Ed. by J. Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 433–444. DOI: 10.1007/3-540-46766-1_35.

[64] P. Reuter and E. M. Truman. *Chasing Dirty Money: The Fight Against Money Laundering*. Peterson Institute for International Economics, 2004.
[65] T. Sander and A. Ta-Shma. "Flow Control: A New Approach for Anonymity Control in Electronic Cash Systems". In: *FC'99*. Ed. by M. Franklin. Vol. 1648. LNCS. Springer, Heidelberg, Feb. 1999, pp. 46–61.
[66] A. Scafuro. "Break-glass Encryption". In: *PKC 2019, Part II*. Ed. by D. Lin and K. Sako. Vol. 11443. LNCS. Springer, Heidelberg, Apr. 2019, pp. 34–62. DOI: 10.1007/978-3-030-17259-6_2.
[67] C.-P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO'89*. Ed. by G. Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22.
[68] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, and C. Allen. *Decentralized Identifiers (DIDs) v1.0 - Core architecture, data model, and representations*. https://w3c-ccg.github.io/did-spec/. Accessed: 2023-05-18.
[69] M. Tibouchi. *CRYPTO and CHES 2016, Santa Barbara, CA, USA*. https://ellipticnews.wordpress.com/2016/09/02/crypto-and-ches-2016-santa-barbara-ca-usa/. Accessed: 2023-05-16.
[70] A. Tomescu et al. *UTT: Decentralized Ecash with Accountable Privacy*. Cryptology ePrint Archive, Report 2022/452. https://eprint.iacr.org/2022/452. 2022.
[71] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun. "PRCash: Fast, Private and Regulated Transactions for Digital Currencies". In: *FC 2019*. Ed. by I. Goldberg and T. Moore. Vol. 11598. LNCS. Springer, Heidelberg, Feb. 2019, pp. 158–178. DOI: 10.1007/978-3-030-32101-7_11.

A Non-Interactive Zero-Knowledge Proofs

Section 3 introduces NIZK syntax and mentions some of the most important aspects. In this section, due to the lack of space, we describe common security definitions that we use to prove our construction secure.

We defined update completeness in Definition 4. The following definition is a standard variant that does not consider Update.

Definition 14 (Completeness). *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* $\mathcal{L}$ *is* perfectly complete*, if for* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, and all* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}), \mathsf{x}) = 1\right] = 1$$

where the randomness is over the random coins of Prove*.*

Definition 15 (Soundness). *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* $\mathcal{L}$ *is* computationally sound*, if for all* $\mathsf{x} \notin \mathcal{L}$ *and all PPT* $\mathcal{A}$ *playing a role of a malicious prover,*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \pi, \mathsf{x}) = 1 \;:\; \begin{array}{l}\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathcal{A}(\mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where the randomness is over the random coins of Setup *and* $\mathcal{A}$*.*

Definition 16 (Zero-Knowledge). *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* $\mathcal{L}$ *is* perfectly zero-knowledge *if there exists a PPT simulator* Sim *such that given for all* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_\mathcal{L}$

$$\left\{(\mathsf{crs}, \pi) \;\middle|\; \begin{array}{c}(\mathsf{crs}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, \mathsf{x})\end{array}\right\} \stackrel{p}{=} \left\{(\mathsf{crs}, \pi) \;\middle|\; \begin{array}{c}(\mathsf{crs}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w})\end{array}\right\}$$

Definition 17 (Strong Simulation-Extractability). *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* $\mathcal{L}$ *is* (strongly straight-line) simulation-extractable*, if it is zero-knowledge and there exists* Ext *such that for all PPT* $\mathcal{A}$

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{crs}, \pi, \mathsf{x}) = 1 \;\wedge \\ (\mathsf{x}, \pi) \notin Q \;\wedge (\mathsf{x}, \mathsf{w}) \notin \mathcal{R}_\mathcal{L}\end{array} \;:\; \begin{array}{l}(\mathsf{crs}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ Q \leftarrow \emptyset \\ (\mathsf{x}, \pi) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sim}}(\cdot,\cdot)}(\mathsf{crs}) \\ \mathsf{w} \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, \pi)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}_{\mathsf{Sim}}(\mathsf{x},\mathsf{w})$ *behaves as follows: it first checks whether* $(\mathsf{x},\mathsf{w}) \in \mathcal{R}_{\mathcal{L}}$*, and if not, aborts; otherwise it generates* $\pi \xleftarrow{\$} \mathsf{Sim}(\mathsf{td},\mathsf{x})$*, sets* $Q \leftarrow Q \cup \{(\mathsf{x},\pi)\}$*, and returns* π*. The existence of* Sim *is guaranteed by ZK.*

The *strong* aspect here highlights non-malleability: an adversary which can randomize proofs can easily win a strong SE game by obtaining a simulated proof π for x, randomizing it into π', and passing it to the game: $(\mathsf{x},\pi') \notin Q$, but Ext will definitely fail because simulated proofs do not contain the witness. Therefore, strong SE prohibits proof randomization.

Note that the definition commonly called *knowledge-soundness* (KS) is a weaker form of simulation-extractability that does not give $\mathcal{A}$ the oracle, does not maintain Q, and does not perform the $(\mathsf{x},\pi) \notin Q$ check.

B Updatable NIZK Arguments

For our main construction, we use CH20 [26] NIZK, which we refer to as Π_{u} which has controlled malleability. In this section, we recall the proof system and argue why is it updatable.

The Π_{u} proof is using a uniform CRS consisting of a single group element $[z]_2$, where $z \leftarrow \mathbb{Z}_q$ is a uniformly sampled trapdoor. For a language matrix $M(\vec{X}) \in \mathcal{P}^{l\times t}$ where l is the size of the instance x and t is the size of the witness w, $\mathsf{Prove}([z]_2,\mathsf{x},\mathsf{w})$ returns $\pi = ([\boldsymbol{a}]_1 \in \mathbb{G}_1^l, [\boldsymbol{d}]_2 \in \mathbb{G}_2^t)$ where elements are constructed as follows:

$$[\boldsymbol{a}]_1 \leftarrow [M(\mathsf{x})]_1 \cdot \boldsymbol{s}$$
$$[\boldsymbol{d}]_2 \leftarrow [z]_2 \cdot \mathsf{w} + [1]_2 \cdot \boldsymbol{s}$$

where $\boldsymbol{s} \in \mathbb{Z}_q^t$ is sampled randomly. That is, the vector $\boldsymbol{s}$ is used as randomization to avoid leakage. Hence proof construction requires $2t+|M(\mathsf{x})|$ curve multiplications (when abusing notation to let $|M(\mathsf{x})|$ denote non-zero entries), of which only $2t$ are in curve 2. Then the proof is verified by running $\mathsf{Verify}([z]_2,\pi) := ([\boldsymbol{a}]_1,[\boldsymbol{d}]_2))$ which checks a single equation

$$\hat{e}([M(\mathsf{x})]_1,[\boldsymbol{d}]_2) \stackrel{?}{=} \hat{e}(\mathsf{x},[z]_2) \cdot \hat{e}([\boldsymbol{a}]_1,[1]_2) \ .$$

That is, the left-hand side is a matrix-vector multiplication of an $l \times t$ matrix and a vector of t elements, where the multiplication operation happens through the pairing. Hence this requires the amount of pairing operations as there are non-zero elements in $M(\mathsf{x})$. On the right-hand side, we compute $2l$ pairings. Hence proof validation requires $2l + |M(\mathsf{x})|$ pairings and no direct curve multiplications. The proof itself contains $|\mathsf{x}| + |\mathsf{w}| = l + t$ elements, of which l are on curve 1 and t on curve 2.

Updates in $\mathcal{L}_M$*.* Rephrasing the definition of updatability for algebraic languages covered in Section 3, $\mathcal{L}$ is updatable w.r.t. transformation $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ if for all $(\mathsf{x},\mathsf{w}) \in \mathcal{R}$, the following relation holds:

$$T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \left(T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}\right)$$

Intuitively, it means that there is a linear way to update both the instance and the witness simultaneously. One simple example is the language of Diffie-Hellman tuples $(C_1,C_2,C_3) := (g^x, g^y, g^{xy})$. The matrix M defines the following relations: $C_1 = G^x, C_2 = G^y, C_3 = C_1^y$. Then, for any α,β it holds that $(C_1^\alpha, C_2^\beta, C_3^{\alpha\beta})$ is also in the relation (this defines T_{xm} implicitly, $T_{\mathsf{xa}} = 0$), since $x' = \alpha x, y' = \beta y$ (this defines T_{wm} with α,β as diagonal, and $T_{\mathsf{wa}} = 0$).

Updates in the CH20 NIZK. The proof updatability in CH20 requires an extra condition: a valid language transformation must be compatible in the following way:

Definition 18 (Blinding-Compatible Transformations). *Let $\mathcal{L}_M$ be an algebraic language defined w.r.t. a matrix $M(\mathsf{x})$. A valid transformation $T := (T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ on $\mathcal{L}$ is said to be* blinding-compatible*, if there exists another pair of matrices $(T_{\mathsf{am}}, T_{\mathsf{aa}}) \in \mathbb{Z}_p^{l\times 2l} \times \mathbb{Z}_p^l$ such that the following equation holds for all $\mathsf{x} \in \mathcal{L}$ and* all $\boldsymbol{s} \in \mathbb{Z}_p^t$*:*

$$T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \left(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}\right)$$

Some valid T are *trivially blinding-compatible*: a simple $(T_{\mathsf{am}}, T_{\mathsf{aa}}) := (T_{\mathsf{xm}} \mid 0^{l\times l}, T_{\mathsf{xa}})$ will satisfy the equation; but this is not true generally.

Note that this equation is quite different from the one for language updatability. First, it applies to all $\boldsymbol{s}$ completely independently from the instance x. Second, $\boldsymbol{s} \in \mathbb{Z}_p^t$ is uniform and might not be a valid witness (e.g. $\mathcal{L}$ may require $w_2 = w_1^2$, but the equation must work for completely independent s_1, s_2). Finally, T_{am} applies not only to $M(\mathsf{x}) \cdot \boldsymbol{s}$, but also to the instance x itself; this actually relaxes the requirement.

Overall, this much stronger condition is necessary due to the linearity that Schnorr-like NIZKs use for blinding w with a uniform $\boldsymbol{s}$.

Define $\mathsf{Update}(([\boldsymbol{a}]_1, [\boldsymbol{d}]_2), T := (T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}}))$ as a function returning $\pi' = ([\boldsymbol{a}']_1, [\boldsymbol{d}']_2)$ constructed as follows:

$$[\boldsymbol{a}']_1 = T_{\mathsf{am}} \cdot \begin{pmatrix} [\boldsymbol{a}]_1 \\ \mathsf{x} \end{pmatrix} + [1]_1 \cdot T_{\mathsf{aa}} + [M(\mathsf{x})]_1 \cdot \hat{\boldsymbol{s}}$$
$$[\boldsymbol{d}']_2 = T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2 + [z]_2 \cdot T_{\mathsf{wa}} + [1]_2 \cdot T_{\mathsf{wa}} + [1]_2 \cdot \hat{\boldsymbol{s}}$$

where $\hat{\boldsymbol{s}}$ is sampled uniformly at random.

Theorem 3 (Update Completeness of CH20). *The CH20 proof system Π_{u} with* Update *satisfies update completeness with respect to all blinding-compatible transformations $T = (T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ on an algebraic language $\mathcal{L}_M$.*

Proof. Consider first the case with $\hat{\boldsymbol{s}} = 0$ — this element rerandomizes the proof, which we will cover in the second step. Observe how the new proof elements are expressed in terms of the old instance and witness:

$$[\boldsymbol{a}']_1 = T_{\mathsf{am}} \cdot \begin{pmatrix} [M(\mathsf{x})]_1 \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + [T_{\mathsf{aa}}]_1$$
$$[\boldsymbol{d}']_2 = [z]_2(T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}) + [T_{\mathsf{wm}} \cdot \boldsymbol{s}]_2 + [T_{\mathsf{wa}}]_2$$

For convenience, define $\mathsf{x}' = T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}$, $\mathsf{w}' = T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}$, $\boldsymbol{s}' = T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}$. Let us check the verification equation directly; by looking at the left and right-hand sides separately:

$$\hat{e}([M(\mathsf{x}')]_1, [\boldsymbol{d}']_2) \qquad = [z \cdot M(\mathsf{x}') \cdot (T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}) + M(\mathsf{x}')(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})]_3$$
$$\hat{e}(\mathsf{x}', [z]_2) \cdot \hat{e}([\boldsymbol{a}']_1, [1]_2) = [z \cdot \mathsf{x}' \qquad\qquad + T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}}]_3$$

Since T is a valid transformation, we first have that $M(\mathsf{x}') \cdot \mathsf{w}' = \mathsf{x}'$. Since, furthermore, T is blinding-compatible, we have $T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}} = M(\mathsf{x}')(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})$. This means that the verification equations will be satisfied by $(\boldsymbol{a}', \boldsymbol{d}')$.

Now, considering $\hat{\boldsymbol{s}} \neq 0$, observe the form of the updated proof elements:

$$[\boldsymbol{a}'']_1 = [\boldsymbol{a}']_1 + [M(\mathsf{x})\hat{\boldsymbol{s}}]_1$$
$$[\boldsymbol{d}'']_2 = [\boldsymbol{d}']_2 + [\hat{\boldsymbol{s}}]_2$$

This merely sets the new challenge randomness to $\boldsymbol{s}' + \hat{\boldsymbol{s}}$, essentially rerandomizing the proof. □

Note that setting $(T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}}) \leftarrow (I_l|0^{l\times l}, 0^l, I_l, 0^l, I_t, 0^t)$ (where I_l is the identity matrix of size l) turns Update into a rerandomization function without transforming the instance.

Theorem 4 (Derivation Privacy of CH20). *The CH20 proof system Π_{u} with Update satisfies derivation privacy for any valid transformation T on algebraic languages.*

Proof. This follows from the form of the update function, and the remark on randomization. The updated proofs which are additionally randomized are distributed in the same way as the honest proofs because randomization has exactly the same form as the honest Prove. □

B.1 Example of Updatability for Algebraic Languages

We will illustrate updatability of algebraic languages and Π_{u} on the example of a concrete small language encoding a Diffie-Hellman tuple. Given a group $\mathbb{G}_1$ of finite prime order p with a generator G, define the language as follows:

$$\mathcal{L}_{\mathsf{dh}} := \{(C_1, C_2, C_3) \mid \exists (a, b) \text{ s.t. } C_1 = G^a \wedge C_2 = G^b \wedge C_3 = G^{ab}\}$$

It is clear that the relation $\mathcal{R}_{\mathsf{dh}}$ is hard under DDH in $\mathbb{G}_1$, therefore proving $\mathsf{x} \in \mathcal{L}_{\mathsf{dh}}$ is non-trivial.

We start by formally defining how $\mathcal{L}_{\mathsf{dh}}$ is expressed algebraically:

$$\begin{matrix}\mathsf{x} = (C_1, C_2, C_3) \in \mathbb{G}_1^3 \\ \mathsf{w} = (a, b) \in \mathbb{Z}_p^2\end{matrix} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 \\ 0 & G \\ 0 & C_1 \end{bmatrix}$$

Recall that our formalization defines $M(\vec{X}) \in \mathcal{P}^{l\times t}$ — in our example, the only instance variable in M is C_1, and all other elements are constants. It is easy to see that for a fixed (x, w) the fact that $M(\mathsf{x}) \cdot \mathsf{w} = \mathsf{x}$ directly implies $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{dh}}$.

First, examining transformations on $\mathcal{L}_{\mathsf{dh}}$ we observe that the following matrices can be used:

$$T_{\mathsf{wm}} = \begin{bmatrix} \gamma & 0 \\ 0 & \delta \end{bmatrix} \quad T_{\mathsf{xm}} = \begin{bmatrix} \gamma & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{wa}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad T_{\mathsf{xa}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

With them we can derive the new instance $\mathsf{x}' := T_{\mathsf{xm}} \cdot \mathsf{x} = (G^{\gamma a}, G^{\delta b}, G^{(\gamma a)(\delta b)})$ that corresponds to the transformed witness $\mathsf{w}' := T_{\mathsf{wm}} \cdot \mathsf{w} = (\gamma a, \delta b)$. Indeed, as required by Definition 2, with these two matrices the following equation holds:

$$M(T_{\mathsf{xm}} \cdot \mathsf{x}) \cdot \left(T_{\mathsf{wm}} \cdot \mathsf{w}\right) = T_{\mathsf{xm}} \cdot \left(M(\mathsf{x}) \cdot \mathsf{w}\right)$$

Finally, let there be a CH20 proof of $\mathsf{x} \in \mathcal{L}$ created as

$$([\boldsymbol{a}]_1, [\boldsymbol{d}]_2) = ([M(\mathsf{x})]_1 \cdot \boldsymbol{s}, [z]_2 \cdot \mathsf{w} + [1]_2 \cdot \boldsymbol{s}) = \left(\begin{pmatrix} [s_1]_1 \\ [s_2]_1 \\ [a \cdot s_2]_1 \end{pmatrix}, \begin{pmatrix} [za + s_1]_2 \\ [zb + s_2]_2 \end{pmatrix}\right)$$

for some random $\boldsymbol{s}$. When verifying this proof, we will compute, on each side:

$$\hat{e}([M(\mathsf{x})]_1, [\boldsymbol{d}]_2) = \begin{pmatrix} [za + s_1]_3 \\ [zb + s_2]_3 \\ [a(zb + s_2)]_3 \end{pmatrix} \text{ and } \hat{e}(\mathsf{x}, [z]_2) \cdot \hat{e}([\boldsymbol{a}]_1, [1]_2) = \begin{pmatrix} [a \cdot z]_3 \\ [b \cdot z]_3 \\ [ab \cdot z]_3 \end{pmatrix} + \begin{pmatrix} [s_1]_3 \\ [s_2]_3 \\ [a \cdot s_2]_3 \end{pmatrix}$$

which are exactly equal. The transformation is trivially blinding-compatible, which means that $(T_{\mathsf{am}}, T_{\mathsf{aa}}) := (T_{\mathsf{xm}} \mid 0^{l\times l}, T_{\mathsf{xa}})$ will do. Then:

$$([\boldsymbol{a}']_1, [\boldsymbol{d}']_2) = (T_{\mathsf{xm}} \cdot [\boldsymbol{a}]_1, T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2) = \left(\begin{pmatrix} [\gamma s_1]_1 \\ [\delta s_2]_1 \\ [\gamma a \cdot \delta s_2]_1 \end{pmatrix}, \begin{pmatrix} [z\gamma a + \gamma s_1]_2 \\ [z\delta b + \delta s_2]_2 \end{pmatrix}\right)$$

will be a (non-rerandomized) transformed proof for x'. It is easy to visually verify that the transformed proof is structured exactly the same as the original proof, but w.r.t. a new instance and witness.

Adding Additive Matrices. Our example can be further extended to commitments on DH tuples; this would use additive matrices as well. Assume $H \in \mathbb{G}_1$ is uniformly sampled.

$$\mathcal{L}_{\mathsf{dh}+} := \{(C_1, C_2, C_3) \mid \exists (a, b) \text{ s.t. } C_1 = G^a H^{r_1} \wedge C_2 = G^b H^{r_2} \wedge C_3 = G^{ab} H^{r_1 b + r_3}\}$$

Because Pedersen commitments are perfectly hiding, every triple of group elements is in the language. But proving membership in it with an argument of *knowledge* is meaningful, because it shows a way to extract a witness computationally, and by the computational binding of Pedersen it will be a unique one[8]. In this case, we have:

$$\begin{array}{c} \mathsf{x} = (C_1, C_2, C_3) \in \mathbb{G}_1^3 \\ \mathsf{w} = (a, b, r_1, r_2, r_3) \in \mathbb{Z}_p^5 \end{array} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 & H & 0 & 0 \\ 0 & G & 0 & H & 0 \\ 0 & C_1 & 0 & 0 & H \end{bmatrix}$$

The matrices we will use are:

$$T_{\mathsf{wm}} = \begin{bmatrix} \gamma & 0 & 0 & 0 & 0 \\ 0 & \delta & 0 & 0 & 0 \\ 0 & 0 & \gamma & 0 & 0 \\ 0 & 0 & 0 & \delta & 0 \\ 0 & 0 & 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{xm}} = \begin{bmatrix} \gamma & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{wa}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ r'_2 \\ r'_3 \end{bmatrix} \quad T_{\mathsf{xa}} = \begin{bmatrix} 0 \\ H^{r'_2} \\ H^{r'_3} \end{bmatrix}$$

To see that this matrix constitutes a valid transformation consider how the new instance and witness look:

$$\begin{aligned} \mathsf{x}' &= \begin{pmatrix} C_1^{\gamma} \\ C_2^{\delta} H^{r'_2} \\ C_3^{\gamma\delta} H^{r'_3} \end{pmatrix} = \begin{pmatrix} G^{\gamma a} H^{\gamma r_1} \\ G^{\delta b} H^{\delta r_2 + r'_2} \\ G^{(\gamma a)(\delta b)} H^{(\gamma r_1)(b\delta) + (\gamma\delta r_3 + r'_3)} \end{pmatrix} \\ \mathsf{w}' &= (\gamma a, \delta b, \gamma r_1, \delta r_2 + r'_2, \gamma\delta r_3 + r'_3) \end{aligned}$$

which clearly belong to the relation $\mathcal{R}_{\mathsf{dh}+}$. Proving the proof transformation is similar to how it was done in the previous example, except now we need to consider the non-zero additive matrices. The proof is structured as follows (where h is a logarithm of H):

$$([\boldsymbol{a}]_1, [\boldsymbol{d}]_2) = ([M(\mathsf{x})]_1 \cdot \boldsymbol{s}, [z]_2 \cdot \mathsf{w} + [1]_2 \cdot \boldsymbol{s}) = \left(\begin{pmatrix} [s_1 + hs_3]_1 \\ [s_2 + hs_4]_1 \\ [as_2 + h(r_1 s_2 + s_5)]_1 \end{pmatrix}, \begin{pmatrix} [za + s_1]_2 \\ [zb + s_2]_2 \\ [zr_1 + s_3]_2 \\ [zr_2 + s_4]_2 \\ [zr_3 + s_5]_2 \end{pmatrix} \right)$$

Now, after updating the proof (again without rerandomization for simplicity) it will look as follows:

$$[\boldsymbol{a}']_1 = T_{\mathsf{xm}} \cdot [\boldsymbol{a}]_1 + [T_{\mathsf{xa}}]_1 = \begin{pmatrix} [\gamma s_1 + h(\gamma s_3)]_1 \\ [\delta s_2 + h(\delta s_4 + r'_2)]_1 \\ [(\gamma a)(\delta s_2) + h((\gamma r_1)(\delta s_2) + (\gamma\delta s_5 + r'_3))]_1 \end{pmatrix}$$

$$[\boldsymbol{d}']_2 = T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2 + [z]_2 T_{\mathsf{wa}} + [T_{\mathsf{wa}}]_2 = \begin{pmatrix} [z \cdot \gamma a + \gamma s_1]_2 \\ [z \cdot \delta b + \delta s_2]_2 \\ [z \cdot \gamma r_1 + \gamma s_3]_2 \\ [z \cdot (\delta r_2 + r'_2) + \delta s_4 + r'_2]_2 \\ [z \cdot (\gamma\delta r_3 + r'_3) + \gamma\delta s_5 + r'_3]_2 \end{pmatrix}$$

It is easy to verify that this is structurally equal to the fresh proof, but w.r.t. $(\mathsf{x}', \mathsf{w}')$; formally, $([\boldsymbol{a}']_1, [\boldsymbol{d}']_2) = ([M(\mathsf{x}')]_1 \cdot \boldsymbol{s}', [z]_2 \cdot \mathsf{w}' + \boldsymbol{s}')$ with $\boldsymbol{s}' = T_{\mathsf{wm}} \boldsymbol{s} + T_{\mathsf{wa}}$.

[8] Even though CH20 proofs are only sound, in practice this issue can be overcome with pairing CH20 with a regular Schnorr, which is ignored in this example.

B.2 Updatability for the Consistency Language

In Section 5.3 we presented the consistency language $\mathcal{L}_{\mathsf{c}}$ together with a general language transformation, that affects virtually all the witness elements simultaneously. In this section, we show that there exist two distinct blinding-compatible transformations for $\mathcal{L}_{\mathsf{c}}$.

The first one, T_{upd}, will assume that the current witness has $\hat{\alpha} = \hat{r}_\alpha = 0$, and therefore $\mathfrak{A} = 1$ in the instance, and will update all the internal randomness values, including the commitment value $\hat{x}_\iota$. This transformation may be parameterized with either $\alpha = r_\alpha = 0$, in which case it can be applied to the transformed proof further; or with $\alpha, r_\alpha \neq 0$, in which case the new hints will become properly blinded, but the transformation will no longer apply. The second one, T_{blind} does not assume $\hat{\alpha} = \hat{r_\alpha} = 0$, can introduce non-zero (α, r_α) and other randomizers, except it does *not* update the value x_ι.

The intuition here is that although the consistency language is fully updatable, we can only update the proof (1) with x_ι, while (A_i, B_i) are not blinded, (2) with the blinder α, but then we cannot update x_ι further. In both cases, all other randomizers are included and non-exclusive. The matrices $T_{\mathsf{am}}, T_{\mathsf{aa}}$ for both of these transformations look exactly the same structurally, except they will need to be parameterized with zero or non-zero update values depending on the case.

We will not further consider T_{blind} in this section because for our application it is only necessary to use T_{upd} in two modes: either with $\alpha = r_\alpha = 0$ when running Update, or with non-zero α, r_α when blinding within Escrow.

We will describe the matrices $T_{\mathsf{am}}, T_{\mathsf{aa}}$ in the form of a list for conciseness — we will present the linear transformations, per row, that characterize the mapping $(\mathsf{x}, S) \mapsto T_{\mathsf{am}} \cdot (S, \mathsf{x})^T + T_{\mathsf{aa}}$, where $S = M(\mathsf{x}) \cdot \boldsymbol{s}$. For clarity, unlike in Section 5, we will use $U_{\cdot}$ notation to denote update values explicitly, e.g. U_x is an update value corresponding to the variable x. In this syntax S_{A_i} corresponds to the line of $M(\mathsf{x}) \cdot \boldsymbol{s}$ that would produce A_i if there was w instead of $\boldsymbol{s}$. Note that S_{3+2d+i} for $i \in [d+1]$ are generally not $1_{\mathbb{G}}$, unlike the real counterparts in x. Given the update values $(U_x, U_{r_{\mathsf{x}},i}, \{U_{r_{\iota},i}\}_{i=1}^{d}, U_\alpha, U_{r_\alpha})$, the update transformation is as follows:

1. $S'_{\mathfrak{T}} = S_{\mathfrak{T}}$.
2. $S'_{\mathfrak{X}} = S_{\mathfrak{X}} G^{U_x} \mathfrak{H}^{U_{r_{\mathsf{x}}}}$.
3. $S'_{\mathfrak{A}} = G^{U_\alpha} \mathfrak{H}^{U_{r_\alpha}}$.
4. $S'_{A_1} = S_{A_1} G^{U_{r_1}}$
5. $S'_{D_1} = S_{D_1} G^{U_x} H^{U_{r_1}} W_1^{U_\alpha}$
6. $S'_{A_i} = \Big(\prod_{j=1}^{i} (S_{A_j})^{V_{i,j}(U_x)} \Big) G^{U_{r_{\iota},i}}$ for $i \in [1, d]$
7. $S'_{D_i} = G^{U_x{}^i_\iota} \Big(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \Big) \Big(\prod_{j=1}^{i-1} D_j^{\binom{i-1}{j} U_x^{i-j}} \Big) H^{U_{r_i}} W_i^{U_\alpha}$ for $i \in [1, d]$
8. $S'_{3+2d+1} = S_{3+2d+1}$
9. $S'_{3+2d+2} = 1$
10. $S'_{5+2d+i} = \Big(\prod_{j=1}^{i} (S_{5+2d+j})^{V_{i,j}(U_x)} \Big) \Big(\prod_{j=1}^{i} (A_j)^{\binom{i}{j} U_x^{i-j+1}} \Big) \Big(\prod_{j=1}^{i} (S_{A_j})^{-\binom{i}{j} U_x^{i-j+1}} \Big)$ for $i \in [1, d-1]$

Now, we need to show why $T_{\mathsf{am}}, T_{\mathsf{aa}}$ defined implicitly so satisfies the blinding-compatibility equation:

$$T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \Big(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}} \Big)$$

More precisely, we need to prove that the equation is satisfied for all $\mathsf{x} \in \mathcal{L}, \boldsymbol{s}$ such that $\mathsf{x}_{\mathfrak{X}} = 1$, and $\boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$.

Lemma 1. *The transformation T_{upd} described above is blinding-compatible w.r.t. $(x, w) \in \mathcal{L}_{\mathsf{c}}$ for which $\alpha = r_\alpha = \boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$.*

Proof. Assume $\boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$. Since the $(T_{\mathsf{am}}, T_{\mathsf{aa}})$ (defined through the set of 10 clauses we just mentioned) determines the left-hand side (LHS) of the blinding-compatibility equation, we will focus on showing that the RHS is equal to it.

Let us look at $\boldsymbol{s}' := (T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})$:

1. $s'_t = s_t$
2. $s'_{r_\mathfrak{t}} = s_{r_\mathfrak{t}}$
3. $s'_x = U_x + s_x$
4. $s'_{r_\mathfrak{x}} = U_{r_\mathfrak{x}} + s_{r_\mathfrak{x}}$
5. $s'_\alpha = U_\alpha$
6. $s'_{r_\alpha} = U_{r_\alpha}$
7. $s'_{x-t} = U_x + s_{x-t}$
8. $s'_{\alpha(x-t)} = U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}$
9. $s'_{r_\alpha(x-t)} = U_{r_\alpha} \cdot U_x + U_{r_\alpha} \cdot s_{x-t}$
10. $s'_{r_i} = \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}\right) + U_{r_i}$, for $i \in [d]$.
11. $s'_{r_i(x-t)} = \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j+1} s_{r_j}\right) + \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j(x-t)}\right) + U_{r_i} U_x + U_{r_i} \cdot s_{x-t}$, for $i \in [d-1]$.

Now let us look at $M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \left(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}\right)$, which we expect to be equal to LHS defining $T_{\mathsf{am}}, T_{\mathsf{aa}}$.

1. "$\mathfrak{T}$": $G^{s_t} \mathfrak{H}^{s_{r_\mathfrak{t}}}$
2. "$\mathfrak{X}$": $G^{U_x + s_x} \mathfrak{H}^{U_{r_\mathfrak{x}} + s_{r_\mathfrak{x}}}$
3. "$\mathfrak{A}$": $G^{U_\alpha} \mathfrak{H}^{U_{r_\alpha}}$
4. "A_1": $G^{U_{r_1} + s_{r_1}}$
5. "D_1": $G^{U_x + s_{x-t}} H^{U_{r_1} + s_{r_1}} W_1^{U_\alpha}$
6. "A_i",$i \in [2,d]$: $G^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}\right) + U_{r_i}}$
7. "D_i",$i \in [2,d]$:
$$\begin{aligned}(D'_{i-1})^{U_x + s_{x-t}} \times (H^{-1})^{\left(\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j} s_{r_j}\right) + \left(\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j-1} s_{r_j(x-t)}\right) + U_{r_{i-1}} U_x + U_{r_{i-1}} \cdot s_{x-t}} \\ \times H^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}\right) + U_{r_i}} \\ \times (W_{i-1}^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} W_i^{U_\alpha}\end{aligned}$$
8. "$3 + 2d + 1$": $G^{s_x + U_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t} + U_x}$
9. "$3 + 2d + 2$": $(G^{U_\alpha} \mathfrak{H}^{U_{r_\alpha}})^{U_x + s_{x-t}} (G^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} (\mathfrak{H}^{-1})^{U_{r_\alpha} \cdot U_x + U_{r_\alpha} \cdot s_{x-t}}$
10. "$5 + 2d + i$", $i \in [d-1]$:
$$(A'_i)^{U_x + s_{x-t}} (G^{-1})^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j+1} s_{r_j}\right) + \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j(x-t)}\right) + U_{r_i} U_x + U_{r_i} \cdot s_{x-t}}$$

Clause 1 is completely unchanged. In clauses 2,3,4,5,6,8 M does not change ($M(\mathsf{x})[i] = M(\mathsf{x}')[i]$), but the witness $\boldsymbol{s}'$ does. In the rest (clauses 7,9,10) both M and the witness change.

It is easy to verify that RHS clauses 1-6 are equal to the LHS presented earlier. Let us examine the other clauses one by one.

Clause 8. The LHS is $S'_{3+2d+1} = S_{3+2d+1}$, while the RHS is $G^{s_x + U_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t} + U_x}$, equal to $G^{s_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t}}$, which is in turn exactly equal to S_{3+2d+1}, as expected.

Clause 9. The LHS is $S_{3+2d+2} = 1$, while the RHS is

$$(G^{U_\alpha} \mathfrak{H}^{U_{r_\alpha}})^{U_x + s_{x-t}} (G^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} (\mathfrak{H}^{-1})^{U_{r_\alpha} \cdot U_x + U_{r_\alpha} \cdot s_{x-t}}$$

which is also equal to 1 when all the exponent terms cancel.

Clause 10. The RHS is equal to:

$$\begin{aligned}
&(A_i')^{U_x+s_{x-t}}(G^{-1})^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(U_x s_{r_j}+s_{r_j(x-t)})\right)+U_{r_i}U_x+U_{r_i}\cdot s_{x-t}}\\
&\quad=(G^{\sum_{j=1}^{i}\binom{i}{j}r_j U_x^{i-j}+U_{r_i}})^{U_x+s_{x-t}}(G^{-1})^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(U_x s_{r_j}+s_{r_j(x-t)})\right)+U_{r_i}U_x+U_{r_i}\cdot s_{x-t}}\\
&\quad=G^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(r_j(U_x+s_{x-t})-U_x s_{r_j}-s_{r_j(x-t)})\right)}
\end{aligned}$$

Now, consider the LHS:

$$\begin{aligned}
&\Big(\prod_{j=1}^{i}(S_{5+2d+j})^{V_{i,j}(U_x)}\Big)\times\Big(\prod_{j=1}^{i}(A_j)^{\binom{i}{j}x_\iota^{i-j+1}}\Big)\times\Big(\prod_{j=1}^{i}(S_{A_j})^{-\binom{i}{j}x_\iota^{i-j+1}}\Big)\\
&\quad=\Big(\prod_{j=1}^{i}(S_{5+2d+j})^{V_{i,j}(U_x)}\Big)\times\Big(\prod_{j=1}^{i}G^{\binom{i}{j}x_\iota^{i-j+1}(r_j-s_{r_j})}\Big)
\end{aligned}$$

Recall that $S_{5+2d+j}=A_j^{s_{x-t}}G^{-s_{r_j(x-t)}}=G^{r_j s_{x-t}-s_{r_j(x-t)}}$, then

$$\begin{aligned}
&=\Big(\prod_{j=1}^{i}(G^{r_j s_{x-t}-s_{r_j(x-t)}})^{\binom{i}{j}U_x^{i-j}}\Big)\times\Big(\prod_{j=1}^{i}G^{\binom{i}{j}x_\iota^{i-j+1}(r_j-s_{r_j})}\Big)\\
&=\Big(\prod_{j=1}^{i}G^{\binom{i}{j}U_x^{i-j}(r_j s_{x-t}-s_{r_j(x-t)}+U_x(r_j-s_{r_j})}\Big)
\end{aligned}$$

which is exactly equal to the RHS.

Clause 7. We start with the RHS:

$$\begin{aligned}
(D_{i-1}')^{U_x+s_{x-t}}\ &\times(H^{-1})^{\left(\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(U_x s_{r_j}+s_{r_j(x-t)})\right)+U_{r_{i-1}}U_x+U_{r_{i-1}}\cdot s_{x-t}}\\
&\times H^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}\right)+U_{r_i}}\\
&\times(W_{i-1}^{-1})^{U_x\cdot U_\alpha+U_\alpha\cdot s_{x-t}}W_i^{U_\alpha}
\end{aligned}$$

Expanding the first term with D_{i-1}' into

$$=(G^{(x+U_x-t)^{i-1}}H^{\sum_{j=1}^{i-1}\binom{i-1}{j}r_j U_x^{i-j-1}+U_{r_{i-1}}}W_{i-1}^{U_\alpha})^{U_x+s_{x-t}}\times\dots$$

we immediately see that the W_{i-1} terms and $H^{U_{r_{i-1}}(U_x+s_{x-t})}$ terms cancel:

$$\begin{aligned}
=(G^{(x+U_x-t)^{i-1}}H^{\sum_{j=1}^{i-1}\binom{i-1}{j}r_j U_x^{i-j-1}})^{U_x+s_{x-t}}\ &\times(H^{-1})^{\left(\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(U_x s_{r_j}+s_{r_j(x-t)})\right)}\\
&\times H^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}\right)+U_{r_i}}\times W_i^{U_\alpha}
\end{aligned}$$

Rearrange the terms, grouping them by base:

$$\begin{aligned}
&=H^{\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(r_j(U_x+s_{x-t})-U_x s_{r_j}-s_{r_j(x-t)})}\times H^{\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}}\\
&\quad\times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})}H^{U_{r_i}}W_i^{U_\alpha}
\end{aligned}$$

Given that $\binom{i}{j}=\binom{i-1}{j-1}+\binom{i-1}{j}$, we have $H^{\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}}=H^{s_{r_i}}H^{\sum_{j=1}^{i-1}(\binom{i-1}{j-1}+\binom{i-1}{j})U_x^{i-j}s_{r_j}}$. Plugging it in into the previous equation we arrive at

$$\begin{aligned}
&=H^{\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(r_j(U_x+s_{x-t})-s_{r_j(x-t)})}\times H^{\sum_{j=1}^{i}\binom{i-1}{j-1}U_x^{i-j}s_{r_j}}\\
&\quad\times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})}H^{U_{r_i}}W_i^{U_\alpha}\\
&=H^{\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(r_j(U_x+s_{x-t})-s_{r_j(x-t)}+s_{r_{j+1}})}\times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})}H^{U_{r_i}+U_x^{i-1}s_{r_1}}W_i^{U_\alpha}
\end{aligned}$$

Now, we will switch to the LHS, and show that it is equal to this last reduced version of the RHS.

$$G^{U_x^i}\left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1}U_x^{i-j}}\right)\left(\prod_{j=1}^{i-1} D_j^{\binom{i-1}{j}U_x^{i-j}}\right) H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{U_x^i} \cdot \prod_{j=1}^{i-1} D_j^{\binom{i-1}{j}U_x^{i-j-1}U_x}\right)\left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1}U_x^{i-j}}\right) \times H^{U_{r_i}} W_i^{U_\alpha}$$

Expand product over D_j:

$$= \left(G^{U_x^i} \cdot \prod_{j=1}^{i-1} D_j^{\binom{i-1}{j}U_x^{i-j-1}U_x}\right)\left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1}U_x^{i-j}}\right) \times H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{U_x^{i-1}} \prod_{j=1}^{i-1} (G^{(x-t)^j} H^{r_j})^{\binom{i-1}{j}U_x^{i-j-1}}\right)^{U_x}\left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1}U_x^{i-j}}\right) \times H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{(x+U_x-t)^{i-1}} H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j-1}}\right)^{U_x}\left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1}U_x^{i-j}}\right) \times H^{U_{r_i}} W_i^{U_\alpha}$$

Recalling that $S_{D_1} = G^{s_{x-t}} H^{s_{r_1}}$ and $S_{D_j} = D_{j-1}^{s_{x-t}} (H^{-1})^{s_{r_{j-1}(x-t)}} H^{s_{r_j}}$, substitute:

$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}}\left(\prod_{j=2}^{i} (D_{j-1}^{s_{x-t}} H^{s_{r_j} - s_{r_{j-1}(x-t)}})^{\binom{i-1}{j-1}U_x^{i-j}}\right)$$
$$\times G^{U_x(x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}}\left(\prod_{j=1}^{i-1} (D_j^{s_{x-t}} H^{s_{r_{j+1}} - s_{r_j(x-t)}})^{\binom{i-1}{j}U_x^{i-j-1}}\right)$$
$$\times G^{U_x(x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}}\left(\prod_{j=1}^{i-1} ((G^{(x-t)^j s_{x-t}} H^{r_j s_{x-t} + s_{r_{j+1}} - s_{r_j(x-t)}})^{\binom{i-1}{j}U_x^{i-j-1}}\right)$$
$$\times G^{U_x(x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(\prod_{j=1}^{i-1} H^{\binom{i-1}{j}U_x^{i-j-1}(r_j U_x + r_j s_{x-t} + s_{r_{j+1}} - s_{r_j(x-t)})}\right) \times G^{(s_{x-t}+U_x)(x+U_x-t)^{i-1}} H^{U_{r_i} + s_{r_1} U_x^{i-1}} W_i^{U_\alpha}$$

It is easy to see now that LHS is equal to the RHS, so clause 7 holds.

□

C Deferred Proofs

Theorem 5 (History Binding). *The* uBlu *protocol is a* history binding *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T, X))$ *by strong simulation-extractability of* Π.

Proof. The first winning condition for the history binding game, that VfHistory must verify for any prefix, is trivially satisfied by the construction: our VfHistory checks every proof linearly, so if all proofs are valid, any prefix is also valid.

The second condition says that two histories are "merged": the last tags are the same, $\mathsf{tag}_\iota^{(0)} = \mathsf{tag}_\iota^{(1)}$, but the histories are different: $\exists i.\, (\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}) \neq (\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)})$. Locate the greatest index j satisfying the second condition. This also implies that $(\mathsf{tag}_{j+1}^{(0)}, \mathfrak{C}_{j+1}^{(0)}) = (\mathsf{tag}_{j+1}^{(1)}, \mathfrak{C}_{j+1}^{(1)})$. We will refer to $\mathsf{tag}_{j+1}^{(0)} = \mathsf{tag}_{j+1}^{(1)}$ as just tag_{j+1}.

Because $\mathsf{tag}_j \leftarrow (\pi_{\mathsf{x},j}, \mathfrak{X}_j)$, our condition is equivalent to either $\pi_{\mathsf{x},j}$, $\mathfrak{X}_j$, or $\mathfrak{C}_j$ being different in the two cases. Note that both $\mathfrak{C}_j$ and $\mathfrak{X}_j$ are inputs to $\pi_{\mathsf{x},j}$; and that $\mathfrak{X}_j$ is an input to $\pi_{\mathsf{x},j+1}$ which is equal in both cases.

By strong simulation-extractability, proofs with different instances are distinct, so by SE of $\pi_{\mathsf{x},j+1}$ we get $\mathfrak{X}_j^{(0)} = \mathfrak{X}_j^{(1)}$ and $\pi_{\mathsf{x},j}^{(0)} = \pi_{\mathsf{x},j}^{(1)}$. Applying the same SE logic to $\pi_{\mathsf{x},j}$ now, we obtain $\mathfrak{C}_j^{(0)} = \mathfrak{C}_j^{(1)}$. This contradicts the second winning condition. □

Theorem 6 (Soundness). *The* uBlu *protocol is a* sound *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T, X))$*, if* Π *is straightline-extractable knowledge sound,* Π_{u} *is sound, and binding of* $\mathsf{PedersenC}$.

Proof. To prove soundness we must show the existence of an extractor Ext that satisfies two probabilistic statements for any PPT $\mathcal{A}$. Our extractor Ext, when given $\mathsf{tag}_\iota = (\pi_{\mathsf{x},\iota}, \mathfrak{X}_\iota)$, will use td_Π produced by Setup to extract witness $(x_\iota, r_{\mathsf{x},\iota}, \mathfrak{r}_\iota)$ from $\pi_{\mathsf{x},\iota}$ such that $\mathfrak{C}_\iota = \mathsf{Commit}(x_\iota, \mathfrak{r}_\iota)$ and $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} G^{x_\iota} \mathfrak{H}^{r_{\mathsf{x},\iota}}$.

The first part of the soundness proof is then a trivial application of knowledge-soundness of Π since it guarantees the well-formedness of $\mathfrak{C}_\iota$ w.r.t. the extracted values. In the rest of the proof, we will focus on the second statement.

The second part of the soundness proof uses the same extractor multiple times, using which we now obtain $\{x_i, r_{\mathsf{x},i}, \mathfrak{r}_i\}_{i=1}^{\iota}$ from the whole trace. In addition, we rely on KS of Π for the escrow proof, and on Π_{u} soundness for the consistency proof.

Right after the extraction from $\mathfrak{X}_i = \mathfrak{X}_{i-1} G^{x_i} \mathfrak{H}^{r_{\mathsf{x},i}}$ and $\mathfrak{X}_0 = 1$ we conclude that $\mathfrak{X}_\iota = G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$ where $\hat{x}_\iota := \sum_{i=1}^{\iota} x_i$ and similarly $\hat{r}_{\mathsf{x},\iota} := \sum_{i=1}^{\iota} r_{\mathsf{x},\iota}$. Because $\mathfrak{T}$ is honestly constructed, we know $\mathfrak{T} = G^t \mathfrak{H}^{r_{\mathfrak{t}}}$.

By soundness of the escrow proof π_{e} we know that $\exists \alpha, r_\alpha, \beta, r_\beta, \beta\alpha, r_\beta\alpha$ such that

$$
\begin{aligned}
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
\mathfrak{B} &= G^\beta \mathfrak{H}^{r_\beta} \\
1 &= \mathfrak{B}^{\hat{\alpha}} (G^{-1})^{\beta\alpha} (\mathfrak{H}^{-1})^{r_\beta \alpha} \\
(E_1, E_2) &= \left(\prod_{i=1}^{d} (A_i^{U_i})^\beta, \prod_{i=1}^{d} (D_i^{U_i})^\beta \cdot \prod_{i=1}^{d} (W_i^{-U_i})^{\beta\alpha} \right)
\end{aligned}
$$

The purpose of lines (2) and (3) is to merely introduce the product variables correctly (without revealing β): by binding of $\mathfrak{B}$ (which is binding of $\mathsf{PedersenC}$) we know that the witness standing for $\beta\alpha$ is actually equal to $\beta \cdot \alpha$, and same stands for $r_\beta\alpha$.

From the lines (1) and (4), simplifying, we now deduce that $\exists \alpha, r_\alpha, \beta$:

$$
\begin{aligned}
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
(E_1, E_2) &= \left(\prod_i A_i^{U_i \beta}, \prod_i D_i^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right)
\end{aligned}
$$

Let us now analyze the consistency proof π_{c}. It can be shown similarly that by binding of the commitments, soundness of π_{c}, and in particular the set of equations in line 7 in the description of $\mathcal{L}_c$ in Section 5.3, that the product witness elements $\alpha(\hat{x}_\iota - t)$ and $\hat{r}_{\iota,i}(\hat{x}_\iota - t)$ are product of, correspondingly, witness elements α with $(\hat{x}_\iota - t)$ and $\hat{r}_{\iota,i}$ with $(\hat{x}_\iota - t)$. In addition, we thus deduce that witness variable $(\hat{x}_\iota - t)$ is equal to the difference of the other two witness variables $\hat{x}_\iota$ and t. With these simplifications in mind, and collapsing

recursive relations in line 6.b of $\mathcal{L}_c$ description, we obtain that $\exists t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}, \alpha, r_\alpha, \{\hat{r}_i\}_{i=1}^d$ such that

$$\begin{aligned}
\mathfrak{T} &= G^t \mathfrak{H}^{r_{\mathsf{t}}} \\
\mathfrak{X}_\iota &= G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}} \\
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
(A_i, D_i) &= (G^{\hat{r}_i}, G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha) \qquad \text{for } i \in [d]
\end{aligned}$$

By binding of $\mathfrak{T}$ and all $\mathfrak{X}_\iota$, we know that the existentially introduced $t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}$ are the same that we constructed from the output of an earlier extractor. Therefore they can be removed from the existential statement together with commitment-validity lines.

By binding of $\mathfrak{A}$ we know that the existentially introduced variables (α, r_α) are equal in both statements, and the statement $\mathfrak{A} = G^\alpha \mathfrak{H}^{r_\alpha}$ itself can be removed from both.

Combining two sets of equations for π_{c} and π_{e} we now have: $\exists \alpha, \beta, \{\hat{r}_i\}_{i=1}^d$:

$$\begin{aligned}
(A_i, D_i) &= (G^{\hat{r}_i}, G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha) \qquad \text{for } i \in [d] \\
(E_1, E_2) &= \left(\prod_i A_i^{U_i \beta}, \prod_i D_i^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right)
\end{aligned}$$

Substituting the first into the second we arrive at: $\exists \alpha, \beta, \{\hat{r}_i\}_{i=1}^d$:

$$\begin{aligned}
(E_1, E_2) &= \left(\prod_i G^{\hat{r}_i U_i \beta}, \prod_i (G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha)^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right) \\
&= \left(\prod_i G^{\hat{r}_i U_i \beta}, \prod_i G^{(\hat{x}_\iota - t)^i U_i \beta} H^{\hat{r}_i U_i \beta} \right) \\
&= \left(G^{\beta \sum_i \hat{r}_i U_i}, G^{\beta \sum_i (\hat{x}_\iota - t)^i U_i} H^{\beta \sum_i \hat{r}_i U_i} \right) \\
&= \left(G^{\beta \sum_i \hat{r}_i U_i}, G^{\beta P(t, \hat{x}_\iota)} H^{\beta \sum_i \hat{r}_i U_i} \right)
\end{aligned}$$

By combining everything, we deduce that (E_1, E_2) is an encryption of $\beta P(t, \hat{x}_\iota := \sum_{i=1}^{\iota} x_i)$; by ElGamal completeness, this will be the predicate value. This last transition used soundness of $\pi_{\mathsf{c}}, \pi_{\mathsf{e}}$ and binding of the commitment scheme. □

Theorem 7 (Threshold Hiding). *The* uBlu *protocol is a* threshold hiding *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T, X))$*, under DDH in* $\mathbb{G}_1$*, ZK of both* Π *and* Π_{u}*, and hiding of* PedersenC.

Proof. First, recall that the honest $\mathsf{KeyGen}(t)$ outputs

$$\begin{aligned}
\mathsf{pk} &= (H, \mathfrak{T}, \pi_{\mathsf{pk}}) \\
\mathsf{hint}_0 &= \left(\{A_{0,i}, D_{0,i}\}_{i \in [d]}, \mathfrak{X}_0 = 1_{\mathbb{G}}, \pi_{\mathsf{c}} \right)
\end{aligned}$$

We proceed in the sequence of hybrid games.

In the first game transition, we will replace $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ with $(\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, and using td will simulate both π_{pk} and π_{c} by zero-knowledge of Π and Π_{u}. The transition is perfect.

In the second game, we will replace $\mathfrak{T}$ with a randomly sampled element. Since the honest $\mathfrak{T} = G^t \mathfrak{H}^{r_{\mathsf{t}}}$ for uniformly random r sampled in KeyGen, $\mathfrak{T}$ is uniform in $\mathfrak{T}$, and thus can be replaced (hiding of PedersenC). This transition is also perfect.

At this stage, the game directly reduces to the IND-CPA of ElGamal (m-message variant) w.r.t. the public key of the regulator, which is known to be implied by DDH in the underlying group. Assuming $\mathcal{A}$ breaks the threshold hiding of our construction, we describe an attacker $\mathcal{B}$ interacting with the d-IND-CPA challenger

$\mathcal{C}$. First, $\mathcal{B}$ obtains $H = G^{\mathsf{sk}}$ from $\mathcal{C}$, which is a public key of the regulator. Then, $\mathcal{B}$ gets two threshold values t_1, t_2 from $\mathcal{A}$, gives $\mathcal{C}$ threshold powers $\{t_1^i\}_{i=1}^d$ and $\{t_2^i\}_{i=1}^d$, obtains ciphertexts $\{A_{0,i}, B_{0,i}\}_{i=1}^d$ which are encryptions of those powers. Remembering that no hint blinding is involved at this step, $\mathcal{B}$ can assemble $(\mathsf{pk}, \mathsf{hint}_0)$ (with simulated proofs and randomly chosen $\mathfrak{T}$ as before) and give it to $\mathcal{A}$. Now, if $\mathcal{A}$ can return $b^\star$ that decides threshold secrecy, then $\mathcal{B}$ can pass this $b^\star$ to $\mathcal{C}$ and decide d-IND-CPA. □

Theorem 8 (Tag Hiding). *The* uBlu *protocol is a* tag hiding *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T, X))$*, assuming ZK of* Π*, and hiding of* PedersenC*.*

Proof. To prove tag hiding, we must construct $\mathcal{S}(\mathsf{td}, \mathsf{pk}, \mathsf{tag}_{\iota-1}, \mathfrak{C}_\iota := \mathfrak{Commit}(x_\iota, \mathfrak{r}_\iota))$ that outputs tag distributed equally to the honest $\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, x_\iota, \mathfrak{r}_\iota)$.

Our simulator $\mathcal{S}$ will return $\mathsf{tag}_\iota = (\pi_{\mathsf{x},\iota}, \mathfrak{X}_\iota)$, where $\mathfrak{X}_\iota$ is randomly sampled, and $\pi_{\mathsf{x},\iota}$ is fully simulated using td. Since the honest $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} \cdot \mathfrak{Commit}(x_\iota; r_{\mathsf{x},\iota}) = \mathfrak{X}_{\iota-1} G^{x_\iota} \mathfrak{H}^{r_{\mathsf{x},\iota}}$ is perfectly hiding due to $r_{\mathsf{x},\iota}$ being sampled randomly, a randomly sampled $\mathfrak{X}_\iota \in \mathbb{G}$ is in distributed exactly the same. Because Π is perfect zero-knowledge, $\mathcal{S}$ with td can simulate $\pi_{\mathsf{x},\iota}$ for the instance $\mathsf{x} = (H, \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \mathfrak{C}_\iota, \pi_{\mathsf{x},\iota-1})$, where H (part of pk), $\mathsf{tag}_{\iota-1} = (\pi_{\mathsf{x},\iota-1}, \mathfrak{X}_{\iota-1})$, and $\mathfrak{C}_\iota$ are all provided as an input to $\mathcal{S}$. □

Theorem 9 (Hint Hiding). *The* uBlu *protocol is a* hint hiding *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T, X))$*, under DDH in* $\mathbb{G}_1$*, ZK of both* Π *and* Π_{u}*, derivation privacy of* Π_{u}*, and hiding of* PedersenC*.*

Proof. Recall the hint structure; as an output of Update we obtain:

$$\mathsf{hint}_\iota = \left(\{A_{\iota,i}, D_{\iota,i}\}_{i\in[d]}, \mathfrak{X}_\iota, \pi_{\mathsf{c},\iota}\right)$$

This proof is very similar to how threshold hiding is proven since KeyGen also produces a hint.

We start with the hint hiding game. As a first step, we switch to simulated setup Setup, and simulate $\pi_{\mathsf{c},\iota}, \pi_{\mathsf{x},\iota}$. One difference with the threshold hiding proof is that while $\pi_{\mathsf{x},\iota}$ is produced by Prove, $\pi_{\mathsf{c},\iota}$ is produce as a result of update of $\pi_{\mathsf{c},\iota-1}$. However, we can still instead simulate $\pi_{\mathsf{c},\iota-1}$ for the new instance, because Π_{u} is derivation private (updated proof is distributed as a completely fresh one for the new instance) and standard zero-knowledge. This transition is perfect.

In the next game, we sample $\mathfrak{X}_\iota$ uniformly at random, as in the threshold hiding proof. Since $\mathsf{PedersenC}$ is perfectly hiding, this transition is also perfect.

Now, we can directly embed d-IND-CPA of ElGamal into our game. This is very similar to the threshold hiding case, except we will request encryptions of $\{(x^{(0)})^i\}_{i=1}^d$ and $\{(x^{(1)})^i\}_{i=1}^d$ from d-IND-CPA challenger, and then homomorphically append them to $\{A_{\iota-1,i}, B_{\iota-1,i}\}_{i=1}^d$. It follows that if $\mathcal{A}$ wins this last variant of the hint hiding game, then the reduction we just described will break d-IND-CPA of ElGamal, which is known to hold under DDH in $\mathbb{G}_1$. □

Before we formulate and prove blueprint hiding, we state simple lemmas explaining why our blinding technique with $D_i = B_i W_i^\alpha$ is hiding.

Lemma 2. *Let* $\mathbb{G}$ *be a finite group with prime order* q *and generator* G*. Then for any polynomial* d*, DDH in* $\mathbb{G}$ *is equivalent to following problem we call* d*-VDDH:*

$$(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^d) \stackrel{c}{\approx} (G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^d)$$

Proof. The basic idea is that under DDH we can always create the $d+1$ challenge tuple $(G^{y_{d+1}}, G^{xy_{d+1} \text{ or } z_{d+1}})$ as $((G^{y_d})^\beta, (G^{xy_d \text{ or } z_d})^\beta)$ for a uniformly sampled β. This proves $(d+1)$-VDDH $\implies$ d-VDDH; everything else is straightforward.

It is clear that 1-VDDH is exactly DDH. It is also straightforward that d-VDDH implies standard $(d-1)$-VDDH and thus DDH, since VDDH tuple contains DDH tuple as a subset (take (G^x, G^{y_1}, G^{xy_1})); so whenever $\mathcal{A}$ can break $(d-1)$-VDDH, $\mathcal{A}$ can break d-VDDH – we just pass $\mathcal{A}$ a subset of the bigger challenge. We will now focus on the DDH $\implies$ d-VDDH direction.

Consider 2-VDDH, a problem to distinguish

$$(G^x, G^{y_1}, G^{xy_1}, G^{y_2}, G^{xy_2}) \text{ and } (G^x, G^{y_1}, G^{z_1}, G^{y_2}, G^{z_2})$$

where z_1, z_2 are uniform in $\mathbb{Z}_p$. When we are given DDH challenge $(C_1, C_2, C_3) := (G^x, G^{y_1}, G^{xy_1}$ or $G^{z_1})$, we can always pick β, and pass $\mathcal{A}$ the following VDDH challenge: $(C_1, C_2, C_3, C_2^\beta, C_3^\beta)$. The fourth element of the tuple is $G^{y_1\beta}$, so call $y_2 := y_1\beta$; the last element of the tuple is either $G^{x(y_1\beta)} = G^{xy_2}$, or it is $G^{z_1\beta}$. This last $z_1\beta$ term is not quite a perfectly uniform G^{z_2}, but it is hard to tell because $G^{z_1\beta}$ can be viewed as a DH challenge itself.

Formally, let $\mathcal{G}_0$ be the original 2-VDDH game, where we generate all the challenges by ourselves. In the $\mathcal{G}_1$ the challenger $\mathcal{C}$ will generate the fourth and fifth element of the tuple as in our reduction, instead of picking fresh (y_2, z_2). We claim that if adversary $\mathcal{B}$ can distinguish $\mathcal{G}_0$ from $\mathcal{G}_1$, we can break DDH. Let us build reduction $\mathcal{R}_\mathcal{B}$. Our DDH challenge be $(C_1, C_2, C_3) := (G^\beta, G^{y_1}, G^{\beta y_1}$ or $G^{y_2})$, where y_2 is assumed to be uniformly random. $\mathcal{R}$ embeds the challenge as follows: it samples x, z_1, and returns either $(G^x, C_2, C_2^x, C_3, C_3^x)$ (real) or $(G^x, C_2, G^{z_1}, C_3, C_1^{z_1})$ (random). If the DDH challenge is real, then $\mathcal{R}$ behaves identical to $\mathcal{G}_1$, returning either $(G^x, G^{y_1}, G^{y_1 x}, G^{\beta y_1}, G^{\beta y_1 x})$ (real) or $(G^x, G^{y_1}, G^{z_1}, G^{\beta y_1}, G^{\beta z_1})$ (random). If the DDH challenge is random, $\mathcal{R}$ is identical to $\mathcal{G}_0$, returning either $(G^x, G^{y_1}, G^{y_1 x}, G^{y_2}, G^{y_2 x})$ (real) or $(G^x, G^{y_1}, G^{z_1}, G^{y_2}, G^{\beta z_1})$ (random). In the latter case, $\beta z_1 \stackrel{\text{p}}{=} z_2$, because β is uniformly random and is used in the tuple only there, once. This proves that $\mathcal{R}_\mathcal{B}$ solves DDH with the same probability of success that $\mathcal{B}$ has in distinguishing $\mathcal{G}_0$ from $\mathcal{G}_1$.

Now, $\mathcal{G}_0 \stackrel{\text{c}}{\approx} \mathcal{G}_1$ under DDH, and $\mathcal{G}_0$ is essentially VDDH. This concludes the proof, because the probability of $\mathcal{A}$ to win $\mathcal{G}_1$ is itself negligible under DDH since it allows for a direct reduction that constructs 2-VDDH tuple from the DDH tuple.

Now it is easy to prove $(d-1)$-VDDH $\implies$ d-VDDH. Consider the d case:

$$(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^d) \stackrel{\text{c}}{\approx} (G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^d)$$

The idea is the same: $\mathcal{G}_0$ is d-VDDH, but in $\mathcal{G}_1$ we return $((G^{y_{d-1}})^\beta, (G^{xy_{d-1}}$ or $^{z_{d-1}})^\beta)$ as the last two tuple elements instead of honest version. By DDH with challenge $(C_1, C_2, C_3) := (G^\beta, G^{y_{d-1}}, G^{\beta y_{d-1}}$ or $G^{y_d})$ we argue that $\mathcal{G}_1 \stackrel{\text{c}}{\approx} \mathcal{G}_2$. This is done as before: the reduction needs to sample $x, z_{d-1}, \{(y_i, z_i)\}_{i=1}^{d-2}$, and simulate the rest of the challenge: it will return either $(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^{d-2}, C_2, C_2^x, C_3, C_3^x)$ or $(G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^{d-2}, C_2, G^{z_{d-1}}, C_3, C_1^{z_{d-1}})$. The argument proceeds as before.

Now, $\mathcal{G}_2$ depends on $y_1 \dots y_{d-1}$ and $z_1 \dots z_{d-1}$, without depending on y_d, z_d, so if $\mathcal{A}$ can solve $\mathcal{G}_2$, then $\mathcal{R}_\mathcal{A}$ can solve $(d-1)$-VDDH. This concludes the proof. □

Lemma 3. *Let $\mathbb{G}$ be a finite group with prime order q and generator G, and d poly-sized. If DDH holds in $\mathbb{G}$, then for all PPT $\mathcal{A}$ the probability of the following game returning 1 is $\leq 1/2 + \mathsf{negl}(\lambda)$:*

1: $\{W_i\}_{i=1}^d \stackrel{\$}{\leftarrow} \mathbb{G}$
2: $(H, \{x_i\}_{i=1}^d) \stackrel{\$}{\leftarrow} \mathcal{A}(1^\lambda, \{W_i\}_{i=1}^d)$
3: $\{z_i, r_i\}_{i=1}^d, \alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_q$
4: $b \stackrel{\$}{\leftarrow} \{0,1\}$
5: $b^\star \leftarrow \mathcal{A}(\{A_i := G^{r_i}, D_i :=$ **if** $b = 0$ **then** $G^{x_i} H^{r_i} W_i^\alpha$ **else** $G^{z_i}\}_{i=1}^d)$
6: **return** $b^\star = b$

Proof. What $\mathcal{A}$ gets is essentially encryptions of $G^{x_i} W_i^\alpha$, where x_i is either random or chosen, for a random α. The structure seems similar to Pedersen commitments, except we use the same randomness α for every commitment, but with different bases W_i. We show that these exponents $x_i + w_i\alpha$ are computationally hiding x_i, assuming d-VDDH, proven secure in Lemma 2 under DDH.

Our d-VDDH challenge is $(C_0, C_1, \dots C_{2d}) := (G^\alpha, \{(G^{w_i}, G^{\alpha w_i}$ or $^{z_i})\}_{i=1}^d)$. It is straightforwardly embedded into our game. First, set all $W_i := C_{2*i-1}$ — the distribution of these is still uniform, so $\mathcal{A}$ does not see the difference when it is first for the first time. After $\mathcal{A}$ responds, as before, generate $\{r_i\}_{i=1}^d \stackrel{\$}{\leftarrow} \mathbb{Z}_q$, and now set all $(A_i := G^{r_i}, D_i := G^{x_i} H^{r_i} C_{2*i})$. The distribution of D_i is as in the honest game too: if VDDH instance is real, $D_i = G^{x_i} H^{r_i} W_i^\alpha$; if VDDH instance is random, then $D_i = G^{x_i} H^{r_i} G^{z_i}$ which is uniform in $\mathbb{G}$, which is the same as G^{z_i} for z_i uniform in $\mathbb{Z}_q$. □

Theorem 10 (Blueprint Hiding). *The* uBlu *protocol is a* blueprint hiding *UPPB scheme for* $(\mathsf{PedersenC}, P_d(T,X))$*, assuming DDH in* $\mathbb{G}_1$*, zero-knowledge of* Π, Π_{u}*, knowledge-soundness of* Π*, soundness of* Π_{u}*, and hiding and binding of* $\mathsf{PedersenC}$*.*

Proof. An escrow has the following form:

$$\mathsf{esc} = \left(E_1, E_2, \pi_{\mathsf{e}}, \pi_{\mathsf{c}}, \mathfrak{X}, \{A_i, D_i\}_{i\in[d]}, \mathfrak{B}, \mathfrak{A}\right)$$

In the blueprint hiding game, we have serveral conditions before we claim equality of distributions. Let us first present the simulator, and then argue, based on these conditions, why the distributions are indeed equal.

The simulator $\mathcal{S}(\mathsf{td}, \mathsf{pk}, v_P := P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota)$ will:

1. Sample $\beta, r_{\mathsf{E}} \xleftarrow{\$} \mathbb{Z}_q$, create fresh encryption $(E_1, E_2) = (G^{r_{\mathsf{E}}}, G^{\beta v_P} H^{r_{\mathsf{E}}})$ based on the predicate value v_P;
2. Sample $\{A_i, D_i\}_{i\in[d]}, \mathfrak{B}, \mathfrak{A} \xleftarrow{\$} \mathbb{G}$ all independently uniformly at random.
3. Set $\mathfrak{X} := \mathsf{tag}_\iota.\mathfrak{X}$.
4. Simulate $\pi_{\mathsf{e}}, \pi_{\mathsf{c}}$ for these elements using td.

To formally prove that $\left\{\mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_\iota)\right\} \overset{c}{\approx} \left\{\mathcal{S}(\mathsf{td}, \mathsf{pk}, P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota)\right\}$ we will proceed in the series of games, starting from $\mathcal{G}_0$, where $\mathcal{A}$ observes $\mathsf{Escrow}_{\mathsf{pk}}(\mathsf{hint}_\iota)$, towards $\mathcal{S}$, implementing parts of the $\mathcal{S}$ step by step.

$\mathcal{G}_1$: Simulate both $\pi_{\mathsf{e}}, \pi_{\mathsf{c}}$ in the output of Escrow. Because both Π and Π_{u} are zero-knowledge, the simulation is perfect, so $\mathcal{G}_1 \overset{\mathrm{p}}{=} \mathcal{G}_2$.

$\mathcal{G}_2$: Sample $\mathfrak{B}, \mathfrak{A} \xleftarrow{\$} \mathbb{G}$ uniformly at random. These two are honest commitments with $\mathfrak{H}$ as a base, and fresh randomness r_α, r_β generated in Escrow. By perfect hiding of Pedersen commitments, $\mathcal{G}_2 \overset{\mathrm{p}}{=} \mathcal{G}_1$.

$\mathcal{G}_3$: In this game, we simulate setup for Π and extract from all $\{\pi_{\mathsf{x},i}\}_{i=1}^{\iota}, \pi_{\mathsf{pk}}$, similarly to how it is done in the soundness proof. The extractor will return $(t', r_{\mathsf{t}}, \{x'_i, \mathfrak{r}'_i, r_{\mathsf{x},i}\}_{i=1}^{\iota})$. Additionally, we assert that $(t', \{x'_i, \mathfrak{r}'_i\}_{i=1}^{\iota})$ are the same as provided by $\mathcal{A}$; we will refer to these values $(t, \{x_i, \mathfrak{r}_i\}_{i=1}^{\iota})$ (without "primes") as before.
This assertion does not fail unless with negligible probability because commitments $\mathfrak{T}, \{\mathfrak{X}_i\}_{i=1}^{\iota}$ are binding. This transition is thus computational by knowledge-soundness of Π and DH in $\mathbb{G}_1$ (which guarantees Pedersen binding).

$\mathcal{G}_4$: In this game we replace (E_1, E_2) with a *fresh* encryption of βv_P instead; except for that, all other parts of esc are as before. This is in contrast with the previous game, where we still honestly call $(E_1, E_2) \leftarrow \mathsf{Evaluate}(\{A_i, B_i\}_{i=1}^{d}, \beta)$.
We start by establishing the form that hint_ι takes, using soundness of $\pi_{\mathsf{c},\iota}$ similarly to the proof of Theorem 6 (soundness). From the previous game we know $x_i, \{r_{\mathsf{x},i}\}_{i\in[d]}$ that each $\mathfrak{X}_i = \mathfrak{X}_{i-1} G^{x_i} \mathfrak{H}^{r_{\mathsf{x},i}}$, and $\mathfrak{X}_0 = 1_{\mathbb{G}}$. Inductively, this implies that $\mathfrak{X}_\iota = G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$, where $\hat{x}_\iota := \sum_{i=1}^{\iota} x_i$, $\hat{r}_{\mathsf{x},\iota} := \sum_{i=1}^{\iota} r_{\mathsf{x},i}$. By soundness of the input $\pi_{\mathsf{c},\iota}$ and binding of $\mathsf{PedersenC}$ (both of which are needed to assert that witness-products are formed correctly; similar to the soundness proof), we deduce that $\exists \{\hat{r}_{\iota,i}\}_{i=1}^{d}, \hat{x}'_\iota, \hat{r}'_{\mathsf{x},\iota}, t', r'_{\mathsf{t}}, \alpha', r'_\alpha$ such that for $i \in [d]$, $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$, $B_{\iota,i} = G^{(\hat{x}'_\iota - t')^i} H^{\hat{r}'_{\iota,i}} W_i^{\alpha'}$, $\mathfrak{X}_\iota = G^{\hat{x}'_\iota} \mathfrak{H}^{\hat{r}'_{\mathsf{x},\iota}}$, $\mathfrak{T} = G^{t'} \mathfrak{H}^{r'_{\mathsf{t}}}$, $1 = \mathfrak{A} = G^{\alpha'} \mathfrak{H}^{r'_\alpha}$. First, since $\pi_{\mathsf{c},\iota}$ verifies w.r.t. $\mathfrak{A} = 1$, we must conclude by binding of $\mathsf{PedersenC}$ that $\alpha' = r'_\alpha = 0$, and thus $B_{\iota,i} = G^{(\hat{x}'_\iota - t')^i} H^{\hat{r}'_{\iota,i}}$ is not blinded. Also by binding of $(\mathfrak{T}, \{\mathfrak{X}_i\}_{i=1}^{\iota})$, we know that $\hat{x}'_\iota$ guaranteed existentially by π_{c} is the same as the extracted $\hat{x}_\iota$; same applies to $t', r'_{\mathsf{t}}, \hat{r}'_{\mathsf{x},\iota}$, so we can remove the "primes". Now we simply know that $\exists \{\hat{r}_{\iota,i}\}_{i=1}^{d}$ such that for $i \in [d]$, $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$, $B_{\iota,i} = G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_{\iota,i}}$.
Honest $\mathsf{Evaluate}$ will return $(E_1, E_2) = \left(\prod_{i\in[d]} (A_i G^{r''_{\iota,i}})^{U_i \beta}, \prod_{i\in[d]} (B_i H^{r''_{\iota,i}})^{U_i \beta}\right)$, where $r''_{\iota,i}$ are freshly sampled on the line 1 of Escrow. Denote $\hat{r}''_{\iota,i} := \hat{r}_{\iota,i} + r''_{\iota,i}$. Then together with the previous existential statement

in mind, we conclude that $\exists\{\hat{r}_{\iota,i}\}_{i=1}^{d}$ such that

$$(E_1, E_2) = \left(\prod_{i\in[d]} G^{(\hat{r}_{\iota,i}+r''_{\iota,i})\cdot U_i\beta}, \prod_{i\in[d]} G^{(\hat{x}_\iota - t)^i \cdot U_i\beta} H^{(\hat{r}_{\iota,i}+r''_{\iota,i})\cdot U_i\beta} \right)$$
$$= \left(G^{\sum_{i=1}^{d} \hat{r}''_{\iota,i}\cdot U_i\beta}, G^{\beta P(t,\hat{x}_\iota)} H^{\sum_{i=1}^{d} \hat{r}''_{\iota,i}\cdot U_i\beta} \right)$$

which is exactly the encryption of $\beta P(t, \hat{x}_\iota)$ with the uniformly random $\sum_{i=1}^{d} \hat{r}''_{\iota,i} \cdot U_i\beta$.
This means that our honestly-produced encryption of $v_P = \beta P(t, \hat{x}_\iota)$ will be distributed equally to freshly encrypted (E_1, E_2). The computational transition is by soundness of π_{c} and binding of PedersenC that implies binding of various $\mathfrak{H}$-based Pedersen commitments.

$\mathcal{G}_5$: Replace $\{A_i, D_i\}_{i\in[d]}$ with randomly sampled elements, instead of returning a blinded re-randomized version of $\{A_i, B_i\}_{i\in[d]}$ with locally sampled α. If $\mathcal{G}_5 \overset{c}{\not\approx}_{\mathcal{B}} \mathcal{G}_4$, then we can build a d-VDDH attacker from $\mathcal{B}$ by embedding the $\{A_i, D_i\}_{i\in[d]}$ challenge directly into our game (and changing W_i in the setup on the first line of the game). Note that important aspect is that Escrow fully rerandomises honest $\{A_i, B_i\}_{i\in[d]}$ at line 1 with $\{r''_{\iota,i}\}_{i\in[d]}$. Therefore, by Lemma 3, $\mathcal{G}_5 \overset{c}{\approx} \mathcal{G}_4$ under DDH in $\mathbb{G}_1$.

The last game is exactly equivalent to the output of $\mathcal{S}$ — note that the honest and simulated $\mathfrak{X}$ is exactly the same, so it does not appear in any game transitions. This concludes the proof. □

D Analysis of Protocol Performance

Section 8 gives a good summary of the performance analysis of our updatable blueprint scheme. In practice, all our computations, except for the quadratic UpdatePowers, are easy to analyze in terms of performance. Zero-knowledge proofs are definitely an exception, and we cover them in detail in this section.

Instantiating a Curve. We mentioned that our construction is not curve-specific, but for the sake of estimates, we mention the performance of BN256 and BLS12-381. While the computational security of the Barreto-Naehrig 256-bit curve (BN256) is generally considered to have less than 128 bits of computational security [69] due to algorithmic advancements [50], this curve is supported by native operations in Solidity on Ethereum[9] and can thus be evaluated with a cheap gas price, in contrast to any other pairing scheme whose operations would have to be manually implemented at a huge gas penalty. To give a rough estimate of curve performance, on AWS z1d.3xlarge a BN256 pairing can be computed in about 0.8ms. Sticking with multiplicative group notation as in our protocol descriptions, exponentiation with a full field scalar is 0.05/0.15ms (in $\mathbb{G}_1$ and $\mathbb{G}_2$ correspondingly); in BLS12-381 implementations timings are only slightly higher: a pairing costs about 1.1 ms and an exponentiation 0.1/0.25ms (for $\mathbb{G}_1$ and $\mathbb{G}_2$ correspondingly) [42, 58].

Consistency Proof. The consistency proof is constructed in KeyGen, updated in Update and Escrow, and verified in VfKeyGen, VfHint and VfEscrow. The consistency language $\mathcal{L}_{\mathsf{c}}$ uses witness of size $|\mathsf{w}| = 2d+8$, and instance of size $|\mathsf{x}| = 2d+4$, therefore being overall $4d+12 \in O(d)$ in size. During updates, four of the witness elements corresponding to zero values of $\alpha = r_\alpha = 0$ related elements, do not need to be communicated, but we will ignore this minor performance improvement and focus on the $O(d)$ terms instead. Our matrices — both $M(\mathsf{x})$ and update matrices $(T_{\mathsf{am}}, T_{\mathsf{wm}}, \ldots)$ — are quite sparse. Estimating the biggest asymptotic term and its constant, as we will do in the rest of the section as well, $|M(\mathsf{x})| \approx 8d$ ($|M(\mathsf{x})|$ stands for the number of non-zero items in the matrix) elements excluding constants, $|M(\mathsf{x})| \approx 6d$ if blinding is disabled, $|T_{\mathsf{am}}| \approx 3 \cdot d^2$, and $|T_{\mathsf{wm}}| \approx (3/2) \cdot d^2$. Performance of Π_{u} mostly depends on the number of elements in these matrices.

[9] Specified in EIP-1108 (https://eips.ethereum.org/EIPS/eip-1108) and introduced in Istanbul fork (https://eips.ethereum.org/EIPS/eip-1679)

The setup phase is $O(1)$ since we only need to generate a single $\mathbb{G}_2$ challenge element. In terms of space, the proof takes $|\mathsf{x}|\cdot\mathbb{G}_1+|\mathsf{w}|\cdot\mathbb{G}_2 = 4d+12$ elements. Proving time for Π_u is equal to $|M(\mathsf{x})|\cdot E_1+2|\mathsf{w}|\cdot E_2$, which is in case of $\mathcal{L}_\mathsf{c}$ is equal to $\approx 6d\cdot E_1+4d\cdot E_2$, where E_i stands for exponentiation in group $\mathbb{G}_i$. Update time is equal to $(|T_{\mathsf{am}}|+|T_{\mathsf{aa}}|+|M(\mathsf{x})|)\cdot E_1+(|T_{\mathsf{wm}}|+|T_{\mathsf{wa}}|+|\mathsf{w}|)\cdot E_2$ (for definition of $T_{\mathsf{am}}, T_{\mathsf{aa}}$ see Appendix B). In the case of $\mathcal{L}_c$ it is quadratic and dominated by the update matrices sizes: $\approx (3d^2+O(d))\cdot E_1+((3/2)d^2+O(d))\cdot E_2$. We note however, that update time in Escrow will be linear because the update matrices will only introduce blinders α, r_α and randomize B_i without updating the commitment value $\hat{x}$ — thus the matrices will only contain a linear number of non-zero elements.

As for verification time, Π_u supports two approaches: naive and batched. With the naive approach, directly as described in [26], the cost will be dominated by about $|M(\mathsf{x})|+2|\mathsf{x}|$ pairings. For the consistency proof language, $\mathcal{L}_\mathsf{c}$, this is equal to $12d+O(1)$ pairings. However, using a well-known pairing batching optimization, we can bring down the verification cost to $(|\mathsf{w}|+2)\cdot P+(|M(\mathsf{x})|+2|\mathsf{x}|)\cdot E_1+3|\mathsf{x}|\cdot E_2$, in our case $\approx (2d+10)\cdot P+10d\cdot E_1+6d\cdot E_2$ (when letting P denote pairings). This is achieved by sampling random values ϕ_i for $i\in[l]$ on the verifier's side and combining all the rows of the verification equation into a single one, where every row is taken with scalar multiple ϕ_i. Security of such a transformation is guaranteed by the Schwartz-Zippel lemma as usual: if the final ϕ_i-linearly-combined equation holds, equations corresponding to the individual rows must hold unless with negligible probability due to ϕ_i being sampled randomly and independently. In practice, this is more efficient, since not only pairings are more expensive than multiplications, but also because we can use multi-scalar-multiplication optimizations to batch the multiplications further.

Non-Updatable Proofs. The key, trace, and escrow proofs are all created using standard Σ-protocol techniques, instantiated in $\mathbb{G}_1$. For all of them, we assume Π to be Fiat-Shamir transformed, but also straightline-extractable, which we will assume is achieved by adding ElGamal encryptions of corresponding witnesses (to a CRS-embedded public key). Without the extra encryptions, Π is quite similar in Π_u, except $\mathbb{G}_2$ exponentiations are now $\mathbb{F}_q$ multiplications, and $\mathbb{G}_1$ exponentiations are used instead of pairings; we still achieve $(|\mathsf{x}|+|\mathsf{w}|)\cdot\mathbb{G}_1$ proof size, $\approx |M(\mathsf{x})|\cdot E_1+|\mathsf{w}|\cdot E_\mathbb{F}$ proving time (where $E_\mathbb{F}$ are multiplications in $\mathbb{F}_q$), and $\approx |M(\mathsf{x})|$ multiplications in verification. For more details see [57, Sec. 6.5]. Straightline extractability will add extra $|\mathsf{w}|$ elements into the witness (Paillier randomness), $2|\mathsf{w}|$ ciphertexts into the instance (Paillier ciphertexts), and extra $3|\mathsf{w}|$ exponentiations into $M(\mathsf{x})$ (each ciphertext is three exponentiations). The setup will only see the extra one Paillier modulus generated, which we will treat as a constant cost. In practice, such transformation will not change asymptotics, only increasing the constant factors by $\approx 2-3$, which is affordable because Π is generally efficient. The approach of [27] can also be used for better asymptotic efficiency.

The key proof is very small and $O(1)$ for our calculations: it only contains 3 instance elements, 4 witness elements, and 5 exponentiations in $M(\mathsf{x})$ in total. It is created in KeyGen and verified in VfKeyGen.

Similarly, the trace proof is $O(1)$ with its 5 instance elements, 3 witness elements, and 5 exponentiations in total. The proof is created as part of Update and is verified in VfHistory.

The escrow proof is also compact, since even though it uses $\{A_i, D_i\}_{i=1}^d$, only their *products* $\prod A_i^{U_i}$ (similarly for D_i) enter $M(\mathsf{x})$, so these products can be treated like constants from the perspective of $\mathcal{L}_\mathsf{e}$. Therefore we have $|\mathsf{x}|=6$, $|\mathsf{w}|=6$, the proof is constant sized, and we will only treat its proving and verification time as $O(d)$ due to the need to construct the aforementioned $\prod A_i^{U_i}$. The proof is constructed as part of Escrow and verified of VfEscrow.