Privacy-Preserving Blueprints

Markulf Kohlweiss¹, Anna Lysyanskaya², and An Nguyen²

University of Edinburgh and Input Output, markulf.kohlweiss(at)ed.ac.uk
Brown University, {anna_lysyanskaya,an_q_nguyen}(at)brown.edu

Abstract. If everyone were to use anonymous credentials for all access control needs, it would be impossible to trace wrongdoers, by design. This would make legitimate controls, such as tracing illicit trade and terror suspects, impossible to carry out. Here, we propose a privacy-preserving blueprint capability that allows an auditor to publish an encoding pk_A of the function $f(x,\cdot)$ for a publicly known function f and a secret input x. For example, x may be a secret watchlist, and f(x,y) may return y if $y \in x$. On input her data y and the auditor's pk_A , a user can compute an escrow Z such that anyone can verify that Z was computed correctly from the user's credential attributes, and moreover, the auditor can recover f(x,y) from Z. Our contributions are:

- We define secure f-blueprint systems; our definition is designed to provide a modular extension to anonymous credential systems.
- We show that secure f-blueprint systems can be constructed for all functions f from fully homomorphic encryption and NIZK proof systems, or from non-interactive secure computation and NIZK. These results are of theoretical interest but is not efficient enough for practical use.
- We realize an optimal blueprint system under the DDH assumption in the random-oracle model for the watchlist function.

1 Introduction

It is a reasonable concern that information technology might enable totalitarian control and favor dystopian rather than utopian societies. Cryptography offers powerful answers on how to strike a balance between privacy and accountability. The study of anonymous credentials [28,55,18,54,19,20,3] has given us general practical tools that make it possible to obtain and prove possession of cryptographic credentials without revealing any additional information. In other words, users can obtain credentials without revealing who they are, and then prove possession of credentials in a way that is unlinkable to the session where these credentials were obtained and to other sessions in which they were shown. Anonymous credentials can be shown a limited number of times (compact ecash) [16], or at a limited rate total or per verifier [15,17]. Anonymous credentials are compatible with identity escrow [52,4], where an appropriate trusted authority can establish the identity of the user when needed.

In this paper, we extend the state-of-the-art on anonymous credentials by adding a new desirable feature: that of a privacy-preserving blueprint capability:

even a malicious authority cannot learn anything about a user other than what's revealed by comparing the blueprinted data with the user's data.

Consider anonymous e-cash with a secret watchlist as a motivating application. In anonymous e-cash [26,27,29,16], we have a Bank that issues e-coins (credentials), Users who withdraw and spend them, and Vendors (or Verifiers) that verify e-coins and accept them as payment in exchange for goods and services. Some small number of users are suspected of financial crimes, and, unbeknownst to them, a judge has placed them on a watchlist. We need a mechanism that allows an auditor to trace the transactions of these watchlisted users without violating the privacy of any other users, and also while keeping the contents of the watchlist confidential from everyone.

A high-level definition of a privacy-preserving blueprint. We have three types of participants: the users, the verifiers, and the auditor who ultimately learns the desired output. On input x (for example, a watchlist), the auditor prepares a blueprint pk_A that the users and verifiers will need.

Next, the user and the verifier engage in an anonymous transaction; we don't actually care what else happens in this transaction; the user might be proving to the verifier that they are authorized, or it may be an e-cash transaction. What we do care about is that, as a by-product of this transaction, the user and the verifier have agreed on a cryptographic commitment C such that (1) the user is in possession of the opening of C; and (2) the transaction that just occurred guarantees that the opening of C contains user data y that is relevant for the auditor's needs. For example, imagine that x is a watchlist consisting of names of individuals of interest, and y contains a user's name; then this user is of interest to the auditor if $y \in x$.

To enhance this anonymous transaction with privacy-preserving blueprint capability, the user runs the algorithm Escrow to compute a value Z that is an escrow of the opening of the commitment C; from Z, the auditor will be able to recover the information relevant to him, and no other information about the user. Specifically, in the watchlist scenario, the auditor will recover y if $y \in x$, but will learn nothing about the user if $y \notin x$. More generally, in an f-blueprint scheme, the auditor will recover f(x,y) and no additional information. The verifier's job is to verify the escrow Z against C using VerEscrow and only let the transaction go through if, indeed, it verifies.

It is important that even a malicious auditor cannot create a blueprint that corresponds to an unauthorized input x. To capture this, we also require that there is a publicly available cryptographic commitment C_A . Outside of our protocol, we expect a mechanism for arriving at an acceptable (but secret) input x and the commitment C_A to x. For example, a judge may publish a commitment to a secret watchlist, and privately reveal the opening to the auditor; or several authorities may be responsible for different components of a watchlist and the auditor aggregates them together in a publicly verifiable fashion; or another distributed protocol can be agreed upon for arriving at the commitment C_A such that its opening (i.e., x) is known to the auditor. To ensure that only such an authorized secret input x is blueprinted, a secure blueprint scheme must include

an algorithm VerPK that verifies that pk_A indeed corresponds to the value to which C_A is a commitment.

Our security definition mandates that the following properties hold: (1) correctness, so that honestly created blueprints and escrows pass VerPK and VerEscrow, respectively, and the escrow Z correctly decrypts; (2) soundness of VerEscrow that ensures that if, for a commitment C, escrow Z is accepted, then it correctly decrypts to f(x,y) where x is the opening of C_A and y is the opening of C; (3) blueprint hiding, i.e., the blueprint pk_A does not reveal anything about x other than what the adversary can learn by forming valid escrows and submitting them for decryption; (4) privacy against a dishonest auditor that ensures that even if the auditor is malicious, an honest user's escrow contains no information beyond f(x,y), where x is the opening of C_A and y is the opening of C; and finally (5) privacy with an honest auditor that ensures that an adversary who does not control the auditor learns nothing from the escrows. We give a precise formal definition of an f-blueprint scheme in Sect. 3.

Our results. Our first result is a blueprint scheme specifically for watchlists; more precisely, it is an f-blueprint scheme for

$$f(x,y) = \begin{cases} y & \text{if } y = y_1 || y_2 \text{ and } y_2 \in x \\ \bot & \text{otherwise} \end{cases}$$

where y_1 denotes $O(\log \lambda)$ most significant bits of y. This first scheme is secure in the random-oracle model under the decisional Diffie-Hellman assumption. The size of pk_A is optimal at $O(\lambda n)$ where λ is the security parameter, which is linear in the number of bits needed to represent a group element; and the watchlist x consists of n elements of \mathbb{Z}_q , where q ($\log q = \Theta(\lambda)$) is the order of the group. The size of the escrow Z is also $O(\lambda n)$.

Our second result consists of two constructions of f-blueprint schemes for any f. The first construction is from fully homomorphic encryption (FHE) and non-interactive zero-knowledge proofs of knowledge (NIZKPoK). The second is from non-interactive secure computation (NISC) [51] and NIZKPoK.

Technical roadmap. We obtain the results above via the same general method: by first defining (Sect. 4) and then realizing (Sections 6, 7 and 8) a homomorphic-enough cryptosystem (HEC) for the function f. We can think of a homomorphic-enough cryptosystem as a protocol between Alice and Bob that works as follows: first, Alice uses the HECENC algorithm to encode her input x into a value X, and she also obtains a decryption key d for future use; next, Bob uses HECEVAL to compute an encryption Z of z = f(x,y) from Alice's encoding X and his input y. Finally, Alice runs HECDEC to recover z from Z. To be useful for our application, an HEC scheme must be correct even when the inputs to the algorithms are chosen maliciously, and it also must ensure that X hides x, and that X and Z together hide the inputs x and y. Additionally, it must allow for an algorithm HECDIRECT that computes an encryption Z of z directly from

X and z = f(x, y), such that its output is indistinguishable from the output of HECEVAL, even if Alice is malicious.

A HEC combined with an appropriate non-interactive zero-knowledge (NIZK) proof system gives a generic construction of an f-blueprint:. The auditor obtains $(X, d) \leftarrow \text{HECenc}(x)$, and an NIZK proof π_A that X was computed correctly in a way that corresponds to the opening of C_A ; he sets the blueprint as $\mathsf{pk}_A = (X, \pi_A)$. Verifying this blueprint amounts to verifying π_A . To compute the escrow Z, the user obtains $Z' \leftarrow \text{HECeval}(X, y)$ and then a proof π_Z that Z' was computed correctly from X and the opening of C; then set $Z = (Z', \pi_Z)$. Verifying the escrow amounts to verifying π_Z . Finally, in order to recover f(x, y) from the escrow Z, the auditor uses the decryption key d to run HECDec(d, Z').

Given this roadmap, our theoretical construction that works for any f is relatively straightforward: in Section 7 we show that HEC can be realized from circuit-private fully homomorphic encryption [47,60,36] which, in turn, can be realized from regular fully homomorphic encryption [47,11,10,48]. The circuit-privacy guarantee ensures that Z hides Bob's input y from a malicious Alice. In Section 8, we explore how it can also be realized for any f from a related primitive of non-interactive secure computation (NISC) [51]. Since here we don't aim for efficiency, general (inefficient) simulation-extractable NIZK PoK can be used for the proofs.

Our practical construction for watchlists under the decisional Diffie-Hellman assumption is not as straightforward: first, it requires that we construct a practical homomorphic enough cryptosystem based on DDH, and next we need efficient non-interactive zero-knowledge proof systems for computing and verifying π_{A} and π_{Z} . Let us give a brief overview.

Our HEC construction uses the ElGamal cryptosystem [37,65] as a building block. Suppose as part of setup we are given a group $\mathbb G$ of order q in which the decisional Diffie-Hellman assumption holds. Let g be a generator of $\mathbb G$. In order to encode her input $x=(a_1,\ldots,a_n)$, Alice's HECENC algorithm first generates an ElGamal key pair (pk, sk). She then picks a random $s\leftarrow\mathbb Z_q^*$ and computes the coefficients c_0,\ldots,c_n of the n-degree polynomial $p(\mathbf x)=s\prod(\mathbf x-a_i)$ for which a_1,\ldots,a_n are the n zeroes. The encoding X are ElGamal encryptions C_0,\ldots,C_n of the values g^{c_0},\ldots,g^{c_n} under the ElGamal public key pk, so the output of HECENC is $X=(C_0,\ldots,C_n,\mathsf{pk})$, and $d=\mathsf{sk}$.

Bob's algorithm HECEVAL computes Z as follows: first, it parses $y = y_1 || y_2$ (recall that y_1 denotes the first $O(\log \lambda)$ bits of y). Then Bob obtains an ElGamal encryption E of $g^{p(y_2)}$ from the encrypted coefficients C_0, \ldots, C_n : since the ElGamal cryptosystem is multiplicatively homomorphic, $E = C_0 C_1^{y_2} C_2^{y_2^2} \ldots C_n^{y_2^n}$ is the desired ciphertext (for an appropriate multiplication operation on ElGamal ciphertexts). Next, let F be an encryption of g^y ; finally, Bob obtains the ciphertext $Z = FE^r$, i.e., Bob uses E to mask the encryption of g^y ; if E is an encryption of 0, the mask won't work and Z will decrypt to g^y .

This is reminiscent of the private set intersection construction of Freedman, Nissim and Pinkas [43], but with a subtle difference: the polynomial encoded as part of X has an additional random coefficient, s. Thus, even if Bob knows

Alice's entire input x, he still does not know p(a) for $a \notin x$. This ensures that in the event that $f(x,y) = \bot$, Bob cannot set r in such a way that Z will decrypt to a value of his choice; instead, it will decrypt to a random value.

Finally, HECDEC(d, Z) decrypts the ElGamal ciphertext Z to some group element $u \in \mathbb{G}$, and for each $a_i \in x$, and for all possible values for y_1 , checks whether $g^{y_1||a_i|} = u$. If it finds such a pair, it outputs it; else, it outputs \perp .

Plugging in this HEC scheme in our generic construction gives us an efficient blueprint scheme for watchlists as long as we can also find efficient instantiations of the NIZK proof systems for computing the proofs π_{A} and π_{Z} . As was already well-known [45,24,58], we can represent the statement that a given ElGamal ciphertext encrypts g^a such that a given Pedersen [62] commitment C is a commitment to a as a statement about equality of discrete logarithm representations; moreover, we can also represent statements about polynomial relationships between committed values (i.e., that C_p is a commitment to the value $p(a_1,\ldots,a_\ell)$ where p is a polynomial, and commitments C_1,\ldots,C_ℓ are to values a_1,\ldots,a_ℓ) as statements about equality of representations. Using this fact, as well as the fact that efficient NIZK proofs of knowledge for equality of discrete logarithm representations in the random-oracle model are known [45,44,32], we can also efficiently instantiate the NIZK proof system in the random-oracle model.

A subtlety in using these random-oracle-based proof systems, however, is that generally such proof systems' knowledge extractors require black-box access to the adversary and involve rewinding it. In situations where the adversary expects to also issue queries to its challenger, and a security experiment or reduction must extract the adversary's witness in order to answer them, using such proof systems runs into the nested rewinding problem. One could opt to use straight-line extractable proofs instead (in such proofs, the knowledge extractor does not need to rewind the adversary); however, known techniques to achieve straight-line extraction come either at a $\omega(\log \lambda)$ multiplicative cost [14,40] or require cumbersome setup assumptions [22].

A more efficient technique is to have a common random string (CRS) and interpret it as an ElGamal public key. There is an efficient Σ -protocol [32] for proving that the contents of two ElGamal ciphertexts under two different keys are equal; it can be converted into a non-interactive zero-knowledge proof using the Fiat-Shamir heuristic [39] in the random-oracle model. If one of these public keys comes from the CRS, then the soundness of the proof system allows for straight-line extraction that uses the corresponding secret key as the extraction trapdoor. Here we give a formalization of this previously used (e.g. [21]) approach.

Specifically, we formulate a new flavor of NIZK proof of knowledge systems: black-box extractability with partial straight-line (BB-PSL) extraction, and give an efficient NIZK BB-PLS PoK proof system for equality of discrete-logarithm representations. This proof system allows straight-line extraction (i.e. extraction from the proof itself, without rewinding the adversary) of a function of the witness (for example, instead of extracting w the extractor computes g^w); this gives the security experiment enough information to proceed. Although this approach

is somewhat folklore, we believe our rigorous formulation and instantiation in the random-oracle model (Sect. 2.2) may be of independent interest.

How our scheme builds on the anonymous credentials literature. Note that, as stated so far, neither the definitions nor the schemes concern themselves with credentials. Instead, the user and the verifier agree on a commitment C to the user's relevant attribute y. Out of band, the user may have already convinced the verifier that she has a credential from some third-party organization attesting that y is meaningful. For example, if y is the user's name, then the third-party organization might be the passport bureau. Indeed, this is how anonymous credentials work in general [54,19,20,3], and therefore this modeling of the problem allows us to add this feature to anonymous credentials in a modular way. Moreover, our ElGamal-based scheme is compatible with literature on anonymous credentials [54,19,20,3] and compact e-cash and variants [16,17,15] because Pedersen commitments are used everywhere.

Related work. Group signatures and identity escrow schemes [30,23,52,2,6] allow users to issue signatures anonymously on behalf of a group such that an anonymity-revoking trustee can discover the identity of the signer. The difference between this scenario and what we are doing here is that in group signatures the signer's identity is always recoverable by the trustee, while here it is only recoverable if it matches the watchlist.

Group signatures with message-dependent opening [63] and bifurcated [53], multimodal and related signatures [59,35,42] allow a tracing authority to recover a function of the user's private information that's known to the user at the time of group-signing or credential showing. In contrast, in a secure blueprinting scheme, the user knows only one of two inputs to this function.

Another related line of work specifically for watchlists is private set intersection (PSI) [43,25]. Although techniques from PSI are helpful here, in general PSI is an interactive two-party protocol, while here, the Auditor who knows the watchlist x is offline at the time when the user is forming the escrow. Private searching on streaming data [61] allows an untrusted proxy to process streaming data using encrypted keywords. The resulting encrypted data does not come with any assurance that it was correct. In contrast, in our scenario, the verifier and the auditor can both verify that Z was computed correctly.

A series of recent papers explored accountable law enforcement access system [49,41,64,50]. None of them, however, consider integration with anonymous credential systems for privacy-preserving authentication. Break-glass encryption by Scafuro [64] realizes a mechanism in which the auditor can decrypt Alice's ciphertexts simply be reliably revealing that he did so, i.e., that he broke the glass. The choice of which messages are to be leaked can happen even after Alice's public key is generated. Scafuro achieves this in certain strong models, such as those of hardware tokens and existence of a blockchain. In abuse-resistant law enforcement access systems (ARLEAS) [50], a law enforcement agency with a valid warrant can secretly place a user Alice under surveillance. They will be able

to decrypt messages that are encrypted to Alice's public key, but not those encrypted to other users for whom surveillance has not been authorized. Moreover, ARLEAS make it possible for an email server to enforce compliance by verifying that an encrypted message indeed allows lawful access by law enforcement; and (in a nutshell) all participants can verify the validity of all warrants even though they are unable to tell who is under surveillance. In our view, ARLEAS follows principles that are similar to ours: finding a way to reconcile the need to monitor illegal activity with privacy needs of the law-abiding public. However, since ARLEAS concerns itself with encryption, while we worry about privacy-preserving authentication, our technical contributions are somewhat orthogonal.

2 Preliminaries

The ElGamal Cryptosystem and Its Security. Let KGen be an algorithm that, on input a description of a group $\mathbb G$ with generator g of prime order q in which the discrete-logarithm problem is hard for PPT (in the security parameter 1^{λ}) adversaries, outputs (1) a public key pk consisting of an element $y \leftarrow \mathbb G$; and (2) a secret key $\mathsf{sk} = s$ such that $g^s = y$. The encryption algorithm Enc encrypts a message $m \in \mathbb G$ by sampling $r \leftarrow \mathbb Z_q$ and outputting the ciphertext $c = (g^r, my^r)$. The decryption algorithm Dec decrypts $c = (a, b) \in \mathbb G^2$ by computing ba^{-s} . We use $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ and $\mathsf{Enc}(\mathsf{pk}, m; r)$ to specify the randomness used in the encryption.

ElGamal is semantically secure under the decisional Diffie-Hellman assumption [66]. In this paper, we use the equivalent notion of security against chosen plaintext attack (IND-CPA) formulated by Boneh and Shoup [8]. In their security game, the adversary continuously interacts with either the 0-encryption oracle that always encrypts the first of the two messages the adversary sends it, or with the 1-encryption oracle that always encrypts the second message. Their security definition is more convenient for us because it allows us to avoid an additional hybrid argument.

Let $\oplus : \mathbb{G}^2 \times \mathbb{G}^2 \to \mathbb{G}^2$ be the operator for the homomorphic composition of two ElGamal ciphertexts $c_1 = (a_1, b_1) \in \mathbb{G}^2$, $c_2 = (a_2, b_2) \in \mathbb{G}^2$ such that: $c_1 \oplus c_2 := (a_1 \cdot a_2, b_1 \cdot b_2)$ where \cdot is the group operator of \mathbb{G} . We also write c^a as shorthand for repeated operation of c with itself a times.

Definition 1 (Statistically hiding non-interactive commitment). A pair of algorithms (CSetup, Commit) constitute a statistically hiding non-interactive commitment scheme for message space M_{cpar} and randomness space R_{cpar} if they satisfy (1) statistical hiding, i.e., for any cpar output by $CSetup(1^{\lambda})$, for any $m_0, m_1 \in M_{cpar}$, the distributions $D(cpar, m_0)$ and $D(cpar, m_1)$ are statistically close, where $D(cpar, m) = \{r \leftarrow R_{cpar} : Commit_{cpar}(m;r)\}$; and (2) computational binding, i.e. for any PPT adversary A, there exists a negligible ν such that $Pr[cpar \leftarrow CSetup(1^{\lambda}); (m_0, r_0, m_1, r_1) \leftarrow A(cpar) : Commit_{cpar}(m_0; r_0) = Commit_{cpar}(m_1; r_1) \land m_0 \neq m_1] = \nu(\lambda)$

We will use the Pedersen commitment scheme which employs a cyclic group \mathbb{G} of prime order q. Let g,h_1,h_2,\ldots,h_n be generators of \mathbb{G} and $m_1,m_2,\ldots,m_n \in \mathbb{Z}_q^n$, then $\mathsf{Commit}_{h_1,h_2,\ldots,h_n,g}(m_1,m_2,\ldots,m_n)$ samples $r \leftarrow \mathbb{Z}_q$ and computes $g^r \prod_{i=1}^n h_i^{m_i}$. This scheme is binding under the discrete logarithm assumption in \mathbb{G} . We write $\mathsf{Commit}_{h_1,\ldots,h_n,g}(m_1,\ldots,m_n;r)$ to specify the randomness for the commitment.

2.1 Non-interactive Zero Knowledge

Let \mathcal{R} be a polynomial-time verifiable binary relation. For a pair $(x, w) \in \mathcal{R}$, we refer to x as the statement and w as the witness. Let $\mathcal{L} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}$.

A non-interactive proof system for $\mathcal R$ consists of a prover algorithm P and verifier algorithm V both given access to a setup S. The setup can either be a random oracle or a reference string—we show later how we abstract over the differences in their interfaces. P takes as input a statement $\mathbb x$ and witness $\mathbb w$, and outputs a proof π if $(\mathbb x,\mathbb w)\in\mathcal R$ and \perp otherwise. V takes as input $(\mathbb x,\pi)$ and either outputs 1 or 0.

Definition 2 (NIZK). Let S be the setup, and (P,V) be a pair of algorithms with access to setup S. $\Phi = (S,P,V)$ is a simulation-sound (optionally extractable) non-interactive zero-knowledge proof system for relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ if it has the following properties:

Completeness: For all
$$(x, w) \in \mathcal{R}$$
, $\Pr[\pi \leftarrow \mathsf{P}^{\mathsf{S}}(x, w) : \mathsf{V}^{\mathsf{S}}(x, \pi) = 0] = 0$.

S is a stateful oracle that captures both the common-random-string setting and the random-oracle setting. In the random-oracle setting, S responds to a query m by sampling a random string h of appropriate length ℓ (clear from context). In the common-reference-string (CRS) setup, it samples a reference string on the first invocation, and from then onward returns the same reference string to all callers.

Zero-knowledge: The zero-knowledge property requires that no adversary can distinguish the real game in which the setup is generated honestly and an honest prover computes proofs using the correct algorithm P, from the simulated game in which the proofs are computed by a simulator that does not take witnesses as inputs, and in which the setup is also generated by the simulator. More formally, there exist probabilistic polynomial time (PPT) simulator algorithms (SimS, Sim) such that, for any PPT adversary $\mathcal A$ interacting in the experiment in Fig. 2.1, the advantage function $\nu(\lambda)$ defined below is negligible:

$$\mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{A}}(\lambda) = \left| \Pr \Big[\mathsf{NIZK}^{\mathcal{A},0}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{NIZK}^{\mathcal{A},1}(1^{\lambda}) = 0 \Big] \right| = \nu(\lambda)$$

$NIZK^{\mathcal{A},0}(1^\lambda)$	$O_S(m)$	$O_P(x, w)$
$\boxed{\mathbf{return}\ \mathcal{A}^{S(\cdot),P^{S}(\cdot,\cdot)}(1^{\lambda})}$	$state, h, \tau_{Ext} \leftarrow SimS(state, m)$	$\mathbf{if}\ (\mathtt{x},\mathtt{w}) \notin \mathcal{R}: \mathbf{return}\ \bot$
$NIZK^{\mathcal{A},1}(1^\lambda)$	$\mathbf{return}\ h$	$state, \pi \leftarrow Sim(state, \mathbf{x})$
$\boxed{\mathbf{return}\ \mathcal{A}^{O_{S}(\cdot),O_{P}(\cdot,\cdot)}(1^{\lambda})}$	-	return π

Fig. 2.1: NIZK game

SimS shares state with Sim modeling both RO programming and CRS trapdoors. Additionally there is an extraction trapdoor τ_{Ext} that will be used below to define simulation extractability.

Soundness: A proof system is sound if no adversary can fool a verifier into accepting a proof of a false statement. It is simulation sound if the adversary cannot do so even given oracle access to the simulator — of course in that case the adversary is prohibited from outputting statement-proof pairs for which the proof was obtained from the simulator. It is a proof of knowledge if a knowledge extractor algorithm can compute the witness given appropriate access to the adversarial prover's algorithm. We explore various flavors of simulation soundness in Appendix A.2, but here we focus on just one of them: the flavor of a proof of knowledge that allows for (partial) straight-line extraction.

2.2 NIZK Proof of Knowledge

Simulation Extractability: A proof system is extractable (also often called a proof of knowledge, or PoK for short) if there exists a polynomial-time extractor algorithm that, on input a proof π for a statement \mathbbm{x} that passes verification, outputs the witness \mathbbm{w} for \mathbbm{x} . In order to reconcile extractability with the zero-knowledge property, it is important that the extractor algorithm Ext have some additional information that is not available to any regular participants in the system. This information depends on the setup \mathbbm{S} : in the CRS setting, it is a trapdoor that corresponds to the CRS; in the random-oracle setting it comes from the ability to observe the adversary's queries to the random oracle. Note that, in addition, trapdoors can be embedded by programming the random oracle. Further, a proof system is simulation-extractable if the extractor algorithm works even when the adversary has oracle access to the simulator and can thus obtain simulated proofs.

Let \mathcal{Q} denote the simulator's query tape that records all the queries the adversary \mathcal{A} made to the simulator. \mathcal{Q}_S denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by O_S and \tilde{O}_S . As we will discuss below, \tilde{O}_S additionally reveals to \mathcal{A} the extraction trapdoor τ_{Ext} ; this captures adaptive extraction from many proofs.

An attractive definition of simulation extractability is the one of straight-line extractability [40]: the extractor obtains the witness just from Q_S and the pair (x, π) . A weaker definition allows for black-box extractability, where the extractor additionally obtains black-box access to A, i.e. it can reset it to a previous state. By BB(A) we denote this mode of access to A, and by $\operatorname{Ext}^{\operatorname{BB}(A)}(Q_S, x, \pi)$ we denote an extractor algorithm that, in addition to its inputs, also has this access to A. See Appendix A.2 for additional discussions and the definition of the black-box simulation extractability game NISimBBExtract. We now propose a notion that falls between straight-line and black-box simulation extractability.

Black-Box with Partial Straight Line (BB-PSL) Simulation Extractability: Sometimes, it is good enough that a straight-line extractor be able to learn something about the witness, say some function f(w), not necessarily the entire witness. For such a scenario, it is convenient to have two extractors: Ext that is a black-box extractor that extracts the entire witness given black-box access to the adversary, and ExtSL that extracts some function of that witness in a straight-line fashion. The reason this is good enough for some proofs of security is that, in a reduction, f(w) may be enough information for the reduction to know how to proceed, without the need to reset the entire security experiment. This is similar to f-extractability [5].

Let us now formalize BB-PSL simulation extractability; let $\Phi = (S, P, V)$ be an NIZK proof system satisfying the zero-knowledge property above; let (SimS, Sim) be the simulator. Let f be any polynomial-time computable function. Φ is f-BB-PSL simulation-extractable if there exists a pair of polynomial-time extractor algorithms (Ext, ExtSL) such that for any PPT adversary \mathcal{A} participating in the game defined in Fig. 2.2, the advantage function $\nu(\lambda)$ defined below is negligible. As mentioned before, \mathcal{Q} denotes the query tape. \mathcal{Q}_{Ext} denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by O_S .

$$\mathsf{Adv}^{\mathsf{NISimBBPSLExtract}}_{\mathcal{A}}(\lambda) = \Pr\Big[f\text{-NISimBBPSLExtract}^{\mathcal{A}}(1^{\lambda}) = 1\Big] = \nu(\lambda)$$

for some negligible function ν .

More on the simulator and extractor. In the games NIZK, NISimSound, and NISimBBPSLExtract the simulator initializes and updates the setup using SimS and then responds to queries from \mathcal{A} for simulated proofs using Sim. Note that the two halves of the simulator, SimS and Sim, share state information, and update it when queried. This captures both the CRS and the random-oracle settings. In the CRS setting, SimS computes the reference string S so that it can pass the corresponding simulation trapdoor to Sim via the shared state. In the random-oracle (RO) setting, SimS programs the random oracle (computes the value h that the random oracle will return when queried on m) and uses the shared state in order to memorize the information that Sim will need to use h in the future. Similarly, in the random-oracle mode, Sim has the ability to program the random oracle as well and memorize what it did using the state variable.

In the simulation extractability experiments, the extractor Ext takes Q_S as input. In the CRS model, Q_S will contain the extraction trapdoor corresponding

Fig. 2.2: f-NISimBBPSLExtract game

to the CRS. In the RO model, Q_S also contains information that the simulator algorithms SimS generated, such as how the RO was programmed and where the adversary queried it. It does not, however, contain the simulation trapdoor or give the extractor the ability to program the RO.

Our definition requires successful extraction even when all information in Q_S , in particular τ_{Ext} , is available to the adversary. This allows the adversary to run Ext itself, and thus allows for extraction from multiple proofs.

Instantiating Simulation Extractable proofs. While simulation-extractable proof systems exist for all NP relations [34], there are multiple ways to realize non-interactive zero-knowledge (NIZK) proof systems more efficiently. One of them is to start with Σ -protocols and convert them into a NIZK proof in the random oracle model, e.g. using the techniques of [39,38,56]. As we will elaborate below, Σ protocols are particularly suitable for proving knowledge of group isomorphisms such as discrete logarithm representations; see, e.g. [57]. They can also efficiently prove disjunctive statements [31]. This has been used for range proofs.

Bulletproofs [13] is a practically efficient NIZK proof system for arithmetic circuits, specifically optimized for range-proofs. Recent work shows that Bulletproofs are simulation extractable [46] and can be integrated with Σ -protocols [12].

Bernhard et al. [7, Theorem 1] state that Fiat-Shamir Σ -protocols are blackbox simulation extractable with respect to expected polynomial-time adversaries. To show partial straightline extractability we use a theorem of [38, Theorem 2] that shows that Σ -protocols compiled using Fiat-Shamir are simulation-sound and adapt the theorem of [33, Theorem F.1] which shows how to transform simulation-sound into simulation-extractable NIZK, by encrypting the witness to the sky. Our approach differs from their approach in that we only encrypt a partial witness and can thus use groups for which computing discrete logarithms is hard.

In Sect. 2.5 we give a construction from Σ -protocols of a proof system Ψ for equality of discrete logarithm representation relations and prove that it is an f-BB-PSL simulation-extractable NIZK proof system in the random-oracle model for an appropriate f.

Notation. When using NIZK proofs of knowledge in a protocol, it is convenient to be able to compactly specify what exactly the prover is proving its knowledge of. We shall use the notation:

$$\pi \leftarrow \mathsf{PoK}_{\varPsi} \Big\{ \mathbb{w} : R(\mathbb{x}, \mathbb{w}) \Big\}$$

to indicate that the proof π was computed as follows: the proof system $\Psi = (\mathsf{S},\mathsf{P},\mathsf{V})$ for the relation R was used; the prover ran $\mathsf{P}^\mathsf{S}(\mathtt{x},\mathtt{w})$; to verify π , the algorithm $\mathsf{V}^\mathsf{S}(\mathtt{x},\pi)$ needs to be run. In other words, the value \mathtt{w} in this notation is the witness the knowledge of which the prover is proving to the verifier, while \mathtt{x} is known to the verifier. A helpful feature of this notation is that it describes what we need Ψ to be: it needs to be a NIZK PoK for the relation R.

2.3 Σ -protocol for proof of equality of discrete logarithm representations

Let R_{eqrep} be the following relation: $R_{\mathsf{eqrep}}(\mathbb{x}, \mathbb{w})$ accepts if $\mathbb{x} = (\mathbb{G}, \{x_i, \{g_{i,1}, \ldots, g_{i,m}\}\}_{i=1}^n)$ where \mathbb{G} is the description of a group of order q, and all the x_i s and $g_{i,j}$ s are elements of \mathbb{G} , and witness $\mathbb{w} = \{w_j\}_{j=1}^m$ such that $x_i = \prod_{j=1}^m g_{i,j}^{w_j}$.

- $\mathbf{P} \rightarrow \mathbf{V}$ On input the $(\mathbf{x}, \mathbf{w}) \in R_{\mathsf{eqrep}}$, the Prover chooses $e_j \leftarrow \mathbb{Z}_q$ for $1 \leq j \leq m$ and computes $d_i = \prod_{j=1}^m g_{i,j}^{e_j}$ for $1 \leq i \leq n$. Finally, the Prover sends to the Verifier the values $\mathsf{com} = (d_1, \dots, d_n)$.
- $\mathbf{P}\leftarrow\mathbf{V}$ On input \mathbb{X} and \mathbf{com} , the Verifier responds with a challenge $\mathbf{chal}=c$ for $c\leftarrow\mathbb{Z}_q$.
- **P** \rightarrow **V** The Prover receives chal = c and computes $s_i = e_i + cw_i \mod q$ for $1 \le i \le m$, and sends res = (s_1, \ldots, s_m) to the Verifier.
- **Verification** The Verifier accepts if for all $1 \le i \le n$, $d_i x_i^c = \prod_{j=1}^m g_{i,j}^{s_j}$; rejects otherwise.
- **Simulation** On input x and chal = c, the simulator chooses $s_j \leftarrow \mathbb{Z}_q$ for $1 \le j \le m$, and sets $d_i = (\prod_{j=1}^m g_{i,j}^{s_j})/x_i^c$ for $1 \le i \le n$. He then sets $com = (d_1, \ldots, d_n)$ and $res = (s_1, \ldots, s_m)$.
- **Extraction** On input two accepting transcripts for the same $\mathsf{com} = (d_1, \ldots, d_n)$, namely $\mathsf{chal} = c$, $\mathsf{res} = (s_1, \ldots, s_m)$, and $\mathsf{chal}' = c'$, $\mathsf{res}' = (s'_1, \ldots, s'_m)$, output $w_j = (s_j s'_j)/(c c') \bmod q$ for $1 \le j \le m$.

2.4 From Σ -protocols to BB simulation extractable NIZK PoK via Fiat-Shamir

Let $\Psi_{eqrep} = (S_{eqrep}, P_{eqrep}, V_{eqrep})$ be the proof system we get from the Σ -protocol described in Sect. 2.3 via the Fiat-Shamir heuristic. Specifically, S_{eqrep} is a random oracle.

We use a theorem of [38, Theorem 2] that shows that Σ -protocols compiled using Fiat-Shamir are simulation-sound; moreover, it follows from a theorem of [7, Theorem 1] and the proof of [38, Theorem 3] that it is in fact black-box simulation extractable.

Recall that the notation $\pi \leftarrow \mathsf{PoK}_{\Psi\mathsf{eqrep}} \big\{ \mathbb{w} : R_{\mathsf{eqrep}}(\mathbb{x}, \mathbb{w}) \big\}$ denotes that the proof π is the output of P_{eqrep} .

2.5 g^x -BB-PSL simulation extractable NIZK from $\Psi_{\sf eqrep}$

Now we want a BB-PSL simulation extractable proof system for R_{eqrep} such that, in a straight-line fashion, a function of w can be extracted. Specifically, recall that $\mathbf{x} = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^n)$ and $\mathbf{w} = \{w_j\}_{j=1}^m$ such that $x_i = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^n)$ $\prod_{j=1}^m g_{i,j}^{w_j}.$

Consider the following proof system $\Psi = (S, P, V)$ for the relation R_{eqrep} and for the function $f(J,\cdot)$, defined as follows. Let g be the generator of \mathbb{G} included in the description of \mathbb{G} . Let J be a subset of the set of indices [m]. Let $f(J, \mathbf{w}) = \{g^{w_j} : j \in J\}.$

S is a random oracle, but we interpret its output as follows: On input the description of a group \mathbb{G} with generator g of order q, outputs a random element h of \mathbb{G} ; we can think of this h as the public key of the ElGamal cryptosystem.

P works as follows: on input $x = (\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^n)$ and w = $\{w_j\}_{j=1}^m$, it first obtains $h=\mathsf{S}(\mathbb{G})$ and then forms the ElGamal ciphertexts of $g^{w_{j_k}}$ for each $j_k \in J$: $(c_{k,1}, c_{k,2}) = (g^{r_k}, g^{w_{j_k}} h^{r_k})$, for $1 \le k \le |J|$.

It then forms x' and w' that allow us to express the following relation R as a special case of R_{egrep} :

$$R = \{ \mathbf{x}', \mathbf{w}' \mid \mathbf{x}' = (\mathbf{x}, \{(c_{j_k,1}, c_{j_k,2})\}) \text{ and}$$

$$\mathbf{w}' = (\mathbf{w}, \mathbf{w}'') \text{ where } \mathbf{w}'' = (r_1, \dots, r_{|J|}) \text{ such that}$$
for $1 \le k \le |J|, (c_{k,1}, c_{k,2}) = (g^{r_k}, g^{w_{j_k}} h^{r_k})$

In order to express x' and w' as a statement and witness for R_{egrep} , form them as follows: $\mathbf{x}' = (\mathbb{G}, \{x'_i, \{g'_{i,1}, \dots, g'_{i,m'}\}\}_{i=1}^{n'}), \text{ where }$

$$n' = n + 2|J|, m' = m + |J|$$

 $n'=n+2|J|,\ m'=m+|J|$ For $1\leq i\leq n,\ x_i'=x_i,$ and for $1\leq j\leq m,\ g_{i,j}'=g_{i,j},$ and for $m< j\leq m+|J|,$

 $g'_{i,j} = 1.$ For $1 \le k \le |J|$, $x'_{n+2(k-1)+1} = c_{k,1}$, $g'_{n+2(k-1)+1,m+k} = g$, and for $\ell \ne m+k$, $1 \le \ell \le m + |J|, g'_{n+2(k-1)+1,\ell} = 1.$

For $1 \le k \le |J|$, $x'_{n+2k} = c_{k,2}$, $g'_{n+2k,j_k} = g$, $g'_{n+2k,m+k} = h$, and for $\ell \notin \{j_k, m+k\}$, $1 \le \ell \le m+|J|$, $g'_{n+2(k-1)+1,\ell} = 1$.

Set $w' = (w_1, \ldots, w_m, r_1, \ldots, r_k)$. Using the algorithm $\mathsf{P}_{\mathsf{eqrep}}^{\mathsf{S}}$, compute $\pi_{\mathsf{eqrep}} \leftarrow$ $\mathsf{PoK}_{\Psi\mathsf{eqrep}}\big\{\, \mathbb{W}' : R_{\mathsf{eqrep}}(\mathbb{X}',\mathbb{W}') \big\}, \, \text{and output } \pi = (\{(c_{k,1},c_{k,2})\},\pi_{\mathsf{eqrep}}).$

V works as follows: on input the statement x, and the proof $\pi = \{\{(c_{k,1}, c_{k,2})\},$ π_{egrep}), first compute x' exactly the same way as the prover's algorithm P did. Then output $V_{\text{eqrep}}^{S}(\mathbf{x}', \pi_{\text{eqrep}})$.

Theorem 1. Let the relation $R_{\mathsf{eqrep}} = \{(\mathbb{X}, \mathbb{W})\}$ be an equality of discrete logarithm representations relation. For any $J \subseteq [m]$, let $f(J, \mathbb{W}) = \{g^{w_j} : j \in J\}$. The proof system $\Psi = (\mathsf{S}, \mathsf{P}, \mathsf{V})$ is an $f(J, \cdot)$ -BB-PSL simulation-extractable NIZK proof system in the random-oracle model.

Proof (Sketch). We need to describe the setup simulator, the proof simulator, the extractor trapdoor and the two extractors.

- SimS(state, m) \rightarrow (state, h', τ_{Ext}): On input the description of a group $\mathbb G$ with generator g of order q, sample $\tau_{\mathsf{Ext}} \leftarrow \mathbb Z_q$ and output the hash value that will be interpreted as the element $g^{\tau_{\mathsf{Ext}}}$ of $\mathbb G$; we can think of this as the public key of the ElGamal cryptosystem for secret key τ_{Ext} . On other inputs simulate the random oracle faithfully.
- Sim(state, \mathbb{x}) \to (state, π): On input \mathbb{x} , the simulator extends \mathbb{x} with random ElGamal ciphertexts to \mathbb{x}' , chooses $c \leftarrow \mathbb{Z}_q$, $s_j \leftarrow \mathbb{Z}_q$ for $1 \leq j \leq m + |J|$, and sets $d_i = (\prod_{j=1}^{m+k} g_{i,j}^{s_j})/x_i^c$ for $1 \leq i \leq n+2|J|$. He then sets com = $(d_1, \ldots, d_{n+2|J|})$, stores $H[\mathbb{x}, \text{com}] = c$ in state, sets chal = c, and res = (s_1, \ldots, s_m) and return (chal, res).
- Ext^{BB(A)}(Q_S, x, π) $\to w$: Parse π as (chal, res) and compute com as Sim. Rewind BB(A) to the point where it queried the random oracle on (x, com) and provide it fresh random results. Repeat until it obtains two accepting transcripts for the same com = $(d_1, \ldots, d_{n+2|J|})$ and then run the extractor of the Σ -protocol to obtain w'. Remove the last k elements to obtain w.
- ExtSL(Q_S, x, π) $\to f(J, w)$: Parse x as $(\mathbb{G}, \{x_i, \{g_{i,1}, \dots, g_{i,m}\}\}_{i=1}^{n+2|J|})$, obtain τ_{Ext} from the entry $(\mathbb{G}, h, \tau_{\mathsf{Ext}})$ of Q_S . Interpret the last 2|J| elements x_i as ElGamal ciphertext and decrypt them to obtain f(J, w).

3 Definition of Security of f-Blueprint Scheme

Our scheme features three parties: an auditor, a set of users, and a set of recipients. It is tied to a non-interactive commitment scheme (CSetup, Commit); let *cpar* be the parameters of the commitment scheme output by CSetup.

The auditor A has private input x and publishes a commitment $C_A = \mathsf{Commit}_{cpar}(x)$. The user has private data y and publishes a commitment $C = \mathsf{Commit}_{cpar}(y)$. For example, x could be a list and y could be the user's attributes in a credential system.

The auditor creates a key pair $(\mathsf{pk_A}, \mathsf{sk_A})$ corresponding to its input x, and the user can escrow its private data y under $\mathsf{pk_A}$ to obtain an escrow Z. We require that Z decrypts (with the help of $\mathsf{sk_A}$) to f(x,y) for a function f that all parties have agreed upon in advance. In the definition, we do not restrict f: it can be any efficiently computable function. Moreover, an escrow recipient R can verify that indeed Z was computed correctly for the given $\mathsf{pk_A}$ and C. Similarly, a privacy-conscious user can verify that indeed $\mathsf{pk_A}$ was computed correctly for the given warrants data commitment C_A .

Definition 3. An f-blueprint scheme tied to a non-interactive commitment scheme (CSetup, Commit) consists of the following probabilistic polynomial time algorithms:

- Setup(1^{λ} , cpar) $\to \Lambda$: is the algorithm that sets up the public parameters Λ . It takes as input the security parameter 1^{λ} and the commitment parameters cpar output by $\mathsf{CSetup}(1^{\lambda})$; to reduce the number of inputs to the rest of the algorithms, Λ includes 1^{λ} and cpar; we will also write Commit instead of Commit_{cpar} to reduce notational overhead.
- $\mathsf{KeyGen}(\Lambda, x, r_\mathsf{A}) \to (\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{A})$: is the key generation algorithm for auditor A. It takes in input 1^λ , parameters Λ , and values (x, r_A) , and outputs the key pair $(\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{A})$. The values (x, r_A) define a commitment C_A . This allows to integrate KeyGen into larger systems.³
- $\mathsf{VerPK}(\Lambda, \mathsf{pk}_\mathsf{A}, C_\mathsf{A}) \to 1$ or 0: is the algorithm that, on input the auditor's public key pk_A and a commitment C_A , verifies that the warrant public key was computed correctly for the commitment C_A .
- Escrow $(\Lambda, \mathsf{pk}_\mathsf{A}, y, r) \to Z$: is the algorithm that, on input the values (y, r) outputs an escrow Z for commitment $C = \mathsf{Commit}(y; r)$.
- VerEscrow(Λ , pk_A, C, Z) \rightarrow 1 or 0: is the algorithm that, on input the auditor's public key pk_A, a commitment C, and an escrow Z, verifies that the escrow was computed correctly for the commitment C.
- Decrypt(Λ , sk_A , C, Z) $\to f(x,y)$ or \bot : is the algorithm that, on input the auditor's secret key sk_A , a commitment C and an escrow Z such that $\operatorname{VerEscrow}(\Lambda,\operatorname{pk}_A,C,Z)=1$, decrypts the escrow. Our security properties will ensure that it will output f(x,y) if C is a commitment to y.

Definition 4 (Secure blueprint). An f-blueprint scheme Blu=(Setup,KeyGen, VerPK, Escrow, VerEscrow, Decrypt) tied to commitment scheme (CSetup, Commit) constitutes a secure f-blueprint scheme if it satisfies the following properties:

Correctness of VerPK and VerEscrow: Values $(cpar, \mathsf{pk}_\mathsf{A}, C_\mathsf{A}, C, Z)$ are generated honestly if: (1) cpar is generated by $\mathsf{CSetup}(1^\lambda)$; (2) Λ is generated by $\mathsf{Setup}(1^\lambda, cpar)$; (3) pk_A is the output of $\mathsf{KeyGen}(\Lambda, x, r_\mathsf{A})$; (4) $C_\mathsf{A} = \mathsf{Commit}_{cpar}(x; r_\mathsf{A})$; (5) $C = \mathsf{Commit}_{cpar}(y; r)$; (6) Z is generated by $\mathsf{Escrow}(\Lambda, \mathsf{pk}_\mathsf{A}, y, r)$. For honestly generated values $(cpar, \mathsf{pk}_\mathsf{A}, C_\mathsf{A}, C, Z)$, we require that algorithms $\mathsf{VerEscrow}$ and VerPK accept with probability 1.

Correctness of Decrypt: Similarly, we require for honestly generated (cpar, pk_A , sk_A , C, Z) that with overwhelming probability $Decrypt(\Lambda, sk_A, C, Z) = f(x, y)$.

Soundness: Let C_A and C be commitments whose openings (x, r_A) and (y, r) are known to the adversary. Let $(\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{KeyGen}(\Lambda, x, r_A)$ be honestly derived keys. Soundness guarantees that any pk_A, Z pair that passes $\mathsf{VerEscrow}(\Lambda, \mathsf{pk}_A, C, Z)$ will decrypt to f(x, y) with overwhelming probability. More formally, for all PPT adversaries $\mathcal A$ involved in the experiment in Fig. 3.1, there exists a negligible function ν such that: $\mathsf{Adv}_{\mathcal A,\mathsf{Blu}}^{\mathsf{Sound}}(\lambda) = \mathsf{Pr} \big[\mathsf{Sound}_{\mathsf{Blu}}^{\mathcal A}(\lambda) = 1 \big] = \nu(\lambda)$

 $^{^3}$ E.g., A can prove that x does not contain journalists, but does contain all Russian oligarchs on the OFAC's sanctions list. https://home.treasury.gov/policy-issues/financial-sanctions

Fig. 3.1: Experiments Sound $_{Blu}^{\mathcal{A}}(\lambda)$

Blueprint Hiding: We want to make sure that pk_A just reveals that x is a valid first argument to f (i.e. this may possibly reveal the size of x or an upper bound on its size). Otherwise, x is hidden even from an adversary who (1) may already know a lot of information about x a-priori; and (2) has oracle access to $\mathsf{Decrypt}(\Lambda, \mathsf{sk}_\mathsf{A}, \cdot, \cdot)$.

We formalize this security property by requiring that there exist a simulator $\operatorname{Sim} = (\operatorname{SimSetup}, \operatorname{SimKeygen}, \operatorname{SimDecrypt})$ such that a PPT adversary cannot distinguish between the following two games: the "real" game in which Λ is chosen honestly, the public key pk_A is computed correctly for adversarially chosen x, r_A , and the adversary's decryption queries (C, Z) are answered with $\operatorname{Decrypt}(\Lambda, \operatorname{sk}_A, C, Z)$; and the "ideal" game in which Λ is computed using $\operatorname{SimSetup}$, the public key pk_A is computed using $\operatorname{SimKeygen}$ independently of x (although with access to the commitment C_A), and the adversary's decryption query Z_i is answered by first running $\operatorname{SimDecrypt}$ to obtain enough information about the user's data y_i to be able to compute $f(x,y_i)$. When we say "enough information," we mean that $\operatorname{SimDecrypt}$ obtains $y_i^* = g(y_i)$ for some function g such that $f(x,y) = f^*(x,g(y))$ for an efficiently computable f^* , for all possible inputs $(x,y)^4$.

More formally, for all probabilistic poly-time adversaries A involved in the game described in Fig. 3.2, the advantage function satisfies:

$$\mathsf{Adv}^{\mathsf{BH}}_{\mathcal{A},\mathsf{Sim}}(\lambda) = \Big|\Pr\Big[\mathsf{BHreal}^{\mathcal{A}}_{\mathsf{Blu}}(\lambda) = 0\Big] - \Pr\Big[\mathsf{BHideal}^{\mathcal{A}}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = 0\Big]\Big| = \nu(\lambda)$$

for some negligible ν .

Privacy against Dishonest Auditor: There exists a simulator such that the adversary's views in the following two games are indistinguishable:

For example, if x is a list (x_1, \ldots, x_n) and f(x, y) checks if $y = x_i$ for some i, g(y) can be a one-way permutation: in order to determine whether y is on the list, it is sufficient to compute $g(x_i)$ and compare it to $y^* = g(y)$.

```
\mathsf{BHreal}^{\mathcal{A}}_\mathsf{Blu}(\lambda)
                                                                                          \mathsf{BHideal}^{\mathcal{A}}_{\mathsf{Blu},\mathsf{Sim}}(\lambda)
        cpar \leftarrow \mathsf{CSetup}(1^{\lambda})
                                                                                            1: cpar \leftarrow \mathsf{CSetup}(1^{\lambda})
         \Lambda \leftarrow \mathsf{Setup}(1^{\lambda}, cpar)
                                                                                            2: (\Lambda, \mathsf{state}) \leftarrow \mathsf{SimSetup}(1^{\lambda}, \mathit{cpar})
         (x, r_{\mathsf{A}}, \mathsf{state}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^{\lambda}, \Lambda)
                                                                                            3: (x, r_A, \mathsf{state}_A) \leftarrow \mathcal{A}(1^\lambda, \Lambda)
                                                                                            4: dsim \leftarrow (|x|, Commit(x; r_A))
 4:
           (\mathsf{pk}_\mathsf{A}, \mathsf{sk}_\mathsf{A}) \leftarrow \mathsf{KeyGen}(\Lambda, x, r_\mathsf{A})
                                                                                            5: (\mathsf{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}}) \leftarrow \mathsf{SimKeygen}(1^{\lambda}, \mathsf{state}, dsim)
          return \mathcal{A}^{O_0(pk_A,sk_A,\cdot,\cdot)}(pk_\Delta,state_\Delta)
                                                                                            6: return \mathcal{A}^{O_1(pk_A, state, x, \cdot, \cdot)}(pk_A, state_A)
O_0(\mathsf{pk}_\mathsf{A},\mathsf{sk}_\mathsf{A},C,Z)
                                                                              O_1(pk_A, simtrap, x, C, Z)
          if \neg VerEscrow(\Lambda, pk_A, C, Z) 1: if \neg VerEscrow(\Lambda, pk_A, C, Z)
                \operatorname{return} \perp
                                                                                                return \perp
 \mathbf{3}: \quad \mathbf{return} \ \mathsf{Decrypt}(\varLambda, \mathsf{sk}_{\mathsf{A}}, C, Z) \quad \mathbf{3}: \quad y^* \leftarrow \mathsf{SimDecrypt}(\mathsf{state}, C, Z)
                                                                                4: return f(x,y) = f^*(x,y^*)
```

Fig. 3.2: Experiments $\mathsf{BHreal}^{\mathcal{A}}_{\mathsf{Blu}}(\lambda)$ and $\mathsf{BHideal}^{\mathcal{A}}_{\mathsf{Blu},\mathsf{Sim}}(\lambda)$

- 1. **Real Game**: The adversary generates the public key and the data x corresponding to this public key, honest users follow the Escrow protocol using adversarial inputs and openings.
- 2. **Privacy-Preserving Game**: The adversary generates the public key and the data x corresponding to this public key. Next, for adversarially chosen inputs and openings, the users run a simulator algorithm that depends only on the commitment and f(x,y) but is independent of the commitment openings.

More formally, there exists algorithms $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimEscrow})$ such that, for any PPT adversary \mathcal{A} involved in the game described in Fig. 3.3, the following equation holds for some negligible function ν :

$$\mathsf{Adv}^{\mathsf{PADA}}_{\mathcal{A},\mathsf{Blu},\mathsf{Sim}}(\lambda) = \bigg| \Pr \bigg[\mathsf{PADA}^{\mathcal{A},0}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = 1 \bigg] - \Pr \bigg[\mathsf{PADA}^{\mathcal{A},1}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = 1 \bigg] \bigg| = \nu(\lambda)$$

Privacy with Honest Auditor: There exists a simulator Sim such that the adversary's views in the following two games are indistinguishable:

- 1. **Real Game**: The honest auditor generates the public key on input x provided by the adversary, and honest users follow the Escrow protocol on input adversarially chosen openings.
- 2. **Privacy-Preserving Game**: The honest auditor generates the public key on input x provided by the adversary. On input adversary-generated commitments and openings, the users run a simulator that is independent of y (although with access to the commitment C) to form their escrows.

```
 \begin{array}{|c|c|} \hline \mathsf{PADA}_{\mathsf{Blu},\mathsf{Sim}}^{\mathcal{A},b}(\lambda) \\ \hline 1: & \mathit{cpar} \leftarrow \mathsf{CSetup}(1^{\lambda}) \\ 2: & \mathcal{A}_0 \leftarrow \mathsf{Setup}(1^{\lambda}, \mathit{cpar}); (\mathcal{A}_1, \mathsf{state}) \leftarrow \mathsf{SimSetup}(1^{\lambda}, \mathit{cpar}) \\ 3: & (x, r_\mathsf{A}, \mathsf{pk}_\mathsf{A}, \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(1^{\lambda}, \mathcal{A}_b) \\ 4: & \text{if } \mathsf{VerPK}(\mathcal{A}_b, \mathsf{pk}_\mathsf{A}, \mathsf{Commit}(x; r_\mathsf{A})) = 0: \mathbf{return} \perp \\ 5: & \mathbf{return} \ \mathcal{A}^{\mathsf{O}_b(y, r)}(\mathsf{state}_\mathcal{A}) \\ \hline \\ \frac{\mathsf{O}_0(y, r)}{1: & \mathbf{return} \ \mathsf{Escrow}(\mathcal{A}_0, \mathit{pk}_\mathsf{A}, y, r)} & \frac{\mathsf{O}_1(y, r)}{1: & \mathbf{return} \ \mathsf{SimEscrow}(\mathsf{state}, \mathcal{A}_1, \mathsf{pk}_\mathsf{A}, \mathsf{Commit}(y; r), \\ 2: & f(x, y)) \\ \hline \end{array}
```

Fig. 3.3: Game $\mathsf{PADA}^{\mathcal{A},b}_\mathsf{Blu}(\lambda)$

In both of these games, the adversary has oracle access to the decryption algorithm.

We formalize these two games in Fig. 3.4. We require that there exists a simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimEscrow})$ such that, for any PPT adversary $\mathcal A$ involved in the game described in the figure, the following equation holds:

$$\mathsf{Adv}^{\mathsf{PWHA}}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = \bigg| \Pr \bigg[\mathsf{PWHA}^{\mathcal{A},0}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = 0 \bigg] - \Pr \bigg[\mathsf{PWHA}^{\mathcal{A},1}_{\mathsf{Blu},\mathsf{Sim}}(\lambda) = 0 \bigg] \bigg| = \nu(\lambda)$$

for some negligible function ν .

4 Homomorphic Enough Encryption

Definition 5 (Homomorphic-enough cryptosystem (HEC) for a function family). Let $F = \{f \mid f : domain_{f,x} \times domain_{f,y} \mapsto range_f\}$ be a set of polynomial-time computable functions. We say that the set HEC of algorithms (HECSETUP, HECENC, HECEVAL, HECDEC, HECDIRECT) constitute a homomorphic-enough cryptosystem (HEC) for F if they satisfy the following input-output, correctness, and security requirements:

- HECSETUP(1^{λ}) \rightarrow hecpar is a PPT algorithm that, on input the security parameter, outputs the parameters hecpar; in case there is no HECSETUP algorithm, hecpar = 1^{λ} .
- HECENC(hecpar, f, x) \to (X, d) is a PPT algorithm that, on input the parameters hecpar, a function $f \in F$, and a value $x \in domain_{f,x}$, outputs an encrypted representation X of the function $f(x, \cdot)$, and a decryption key d.
- HECEVAL(hecpar, f, X, y) $\to Z$ is a PPT algorithm that, on input the parameters hecpar, a function $f \in F$, an encrypted representation of $f(x, \cdot)$, and a value $y \in domain_{f,y}$, outputs a ciphertext Z, an encryption of f(x, y).

```
 \begin{array}{|c|c|} \hline \mathsf{PWHA}_{\mathsf{Blu},\mathsf{Sim}}^{A,b}(\lambda) \\ \hline 1: & cpar \leftarrow \mathsf{CSetup}(1^{\lambda}) \\ 2: & \varLambda_0 \leftarrow \mathsf{Setup}(1^{\lambda}, cpar); \varLambda_1 \leftarrow \mathsf{SimSetup}(1^{\lambda}, cpar) \\ 3: & M \leftarrow [\ ] \\ \hline 4: & x, r_{\mathsf{A}} \leftarrow \mathcal{A}(1^{\lambda}, \varLambda_b) \\ 5: & (\mathsf{pk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{A}}) \leftarrow \mathsf{KeyGen}(\varLambda_b, x, r_{\mathsf{A}}) \\ \hline 6: & \mathbf{return} \ \mathcal{A}^{\mathsf{O}_b^{\mathsf{Escrow}}(\cdot, \cdot), \mathsf{O}^{\mathsf{Decrypt}}(\varLambda_b, \mathsf{sk}_{\mathsf{A}}, \cdot, \cdot)}(\mathsf{pk}_{\mathsf{A}}) \\ \hline \hline 1: & \mathbf{return} \ \mathsf{Escrow}(\varLambda_0, \mathsf{pk}_{\mathsf{A}}, y, r) & \underbrace{\begin{matrix} \mathsf{O}_1^{\mathsf{Escrow}}(y, r) \\ 2: Z \leftarrow \mathsf{SimEscrow}(\mathsf{state}, \varLambda_1, \mathsf{pk}_{\mathsf{A}}, C) \\ 3: M[C, Z] \leftarrow f(x, y) \\ 4: & \mathbf{return} \ Z \\ \hline \\ \underbrace{\begin{matrix} \mathsf{O}^{\mathsf{Decrypt}}(\varLambda_1, \mathsf{sk}_{\mathsf{A}}, C, Z) \\ 1: & \mathbf{if} \ M[C, Z] \ \text{is defined } \mathbf{return} \ M[C, Z] \\ 2: & \mathbf{return} \ \mathsf{Decrypt}(\varLambda_1, \mathsf{sk}_{\mathsf{A}}, C, Z) \\ \hline \\ 2: & \mathbf{return} \ \mathsf{Decrypt}(\varLambda_1, \mathsf{sk}_{\mathsf{A}}, C, Z) \\ \hline \\ 2: & \mathbf{return} \ \mathsf{Decrypt}(\varLambda_1, \mathsf{sk}_{\mathsf{A}}, C, Z) \\ \hline \\ \end{array}
```

Fig. 3.4: Game $\mathsf{PWHA}^{\mathcal{A},b}_{\mathsf{Blu},\mathsf{Sim}}(\lambda)$

HECDEC(hecpar, d, Z) $\rightarrow z$ is a polynomial-time algorithm that, on input the parameters hecpar, the decryption key d, and a ciphertext Z, decrypts Z to obtain a value z.

HECDIRECT $(hecpar, X, z) \to Z$ is a PPT algorithm that, on input hecpar, an encrypted representation X of some function, and a value z, outputs a ciphertext Z.

HEC correctness. For a given adversary \mathcal{A} and HEC, let $\mathsf{Adv}_{\mathsf{HEC},\mathcal{A}}(\lambda)$ be the probability that the experiment HECCORRECT in Fig. 4.1 accepts. HEC is correct if $\mathsf{Adv}_{\mathsf{HEC},\mathcal{A}}(\lambda)$ is negligible for all PPT algorithms \mathcal{A} .

Security of x, security of x and y from third parties, and security of DirectZ. Consider Fig. 4.1. For a given HEC and an adversary \mathcal{A} , and for $b \in \{0,1\}$, let $p_{\mathcal{A},b}^{\text{SecX}}(\lambda)$ be the probability that \mathcal{A} outputs 0 in experiment $\text{SecXY}_b^{\mathcal{A}}$, let $p_{\mathcal{A},b}^{\text{SecXY}}(\lambda)$ be the probability that \mathcal{A} outputs 0 in experiment $\text{SecXY}_b^{\mathcal{A}}$, and let $p_{\mathcal{A},b}^{\text{DIRECTZ}}(\lambda)$ be the probability that \mathcal{A} outputs 0 in experiment $\text{DIRECTZ}_b^{\mathcal{A}}$.

HEC provides security for x if or any PPT \mathcal{A} , $|p_{\mathcal{A},0}^{\text{SecX}}(\lambda) - p_{\mathcal{A},1}^{\text{SecX}}(\lambda)|$ is negligible. HEC provides security for x and y from third parties if or any PPT \mathcal{A} , $|p_{\mathcal{A},0}^{\text{SecXY}}(\lambda) - p_{\mathcal{A},1}^{\text{SecXY}}(\lambda)|$ is negligible. HEC provides security of DIRECTZ if or any PPT \mathcal{A} , $|p_{\mathcal{A},0}^{\text{DIRECTZ}}(\lambda) - p_{\mathcal{A},1}^{\text{DIRECTZ}}(\lambda)|$ is negligible.

```
HECCORRECT^{\mathcal{A}}(\lambda)
                                                                       SecXY_b^{\mathcal{A}}(\lambda)
       hecpar \leftarrow HECsetup(\lambda)
                                                                               hecpar \leftarrow HECsetup(1^{\lambda})
      (f, x, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)
                                                                                (f, x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)
       if f \in F, x \in domain_{f,x}
                                                                               if f \in F, x_0, x_1 \in domain_{f,x}
           (X,d) \leftarrow \text{HECenc}(hecpar, f, x)
                                                                                   X, \_ \leftarrow \text{HECenc}(hecpar, f, x_b)
           (y, r_Z) \leftarrow \mathcal{A}(\mathsf{state}, X)
                                                                                   (y_0, y_1, \mathsf{state}) \leftarrow \mathcal{A}(X, \mathsf{state})
           if y \in domain_{f,y}
                                                                                   if y_0, y_1 \in domain_{f,y}
 6:
                                                                         6:
               Z \leftarrow \text{HECeval}(hecpar, f, X, y; r_Z)
                                                                                       Z \leftarrow \text{HECeval}(hecpar, f, X, y_b)
7:
                                                                         7:
              if HECDEC(hecpar, d, Z) \neq f(x, y)
                                                                                       return A(Z, state)
 8:
                                                                         8:
                                                                                   return A(\perp, state)
9:
                  return 1
               return 0
                                                                               return A(\perp, state)
10:
                                                                        10:
           return 0
11:
                                                                        DirectZ_b^A(\lambda)
       return 0
                                                                               hecpar \leftarrow \text{HECsetup}(1^{\lambda})
SecX_b^A(\lambda)
                                                                               (f, x, y, r_X, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)
       hecpar \leftarrow HECsetup(1^{\lambda})
                                                                               if f \in F, x \in domain_{f,x}, y \in domain_{f,y}
       (f, x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)
                                                                                   X, = \text{HECenc}(hecpar, f, x; r_X)
     if f \in F, x_0, x_1 \in domain_{f,x}
                                                                                   Z_0 \leftarrow \text{HECeval}(hecpar, f, X, y)
                                                                                   Z_1 \leftarrow \text{HECDIRECT}(hecpar, X, f(x, y))
           X, \_ \leftarrow \text{HECenc}(hecpar, f, x_b)
                                                                         6:
5:
           return \mathcal{A}(hecpar, X, state)
                                                                                   return \mathcal{A}(hecpar, Z_b, state)
 6: return A(\perp, state)
                                                                               return A(\perp, state)
```

Fig. 4.1: HEC correctness and security games

Remark. Why do we need HECDIRECT? It allows us to directly form a ciphertext Z that will decrypt to a specific value z. If the function f is not one-way and it is easy, given z, to sample x and y such that z = f(x, y), then we can derive such Z by computing (X, d) = HECenc(hecpar, f, x) and then computing Z = HECeval(hecpar, f, X, y). But in general, it is helpful (for some applications) to have a separate algorithm HECDIRECT(hecpar, X, z) such that, if X = HECenc(hecpar, f, x), then Z = HECdirect(hecpar, X, z) decrypts to z using the decryption key that corresponds to X, i.e. z = HECdec(hecpar, d, Z).

5 A Generic f-Blueprint scheme from HEC

We construct a privacy-preserving blueprint scheme using a commitment scheme, a homomorphic-enough cryptosystem, as well as two NIZK proof systems as building blocks. The scheme consists of the following six algorithms:

Setup takes λ and a commitment setup as input and generates *hecpar* and assigns the NIZK oracles S_1 and S_2 . Note that when instantiated using real hash functions or reference strings both RO and CRS setups can be represented as bit-strings in implementations. KeyGen uses the HEC scheme to compute an

encrypted representation of the function $f(x,\cdot)$ and proves that it was computed correctly. VerPK verifies that pk_A was computed correctly with respect to the auditor's commitment C_A . Escrow homomorphically evaluates $f(x,\cdot)$ on y to obtain a ciphertext and proves that it was formed correctly. VerEscrow verifies the ciphertext with respect to the user's commitment C, and Decrypt decrypts.

Our construction in Fig. 5.1 uses VerPK as a subroutine in Escrow and VerEscrow. To be consistent with the syntax we add C_A to pk_A . Similarly, we use VerEscrow in Decrypt and add pk_A to sk_A .

```
\mathsf{Setup}(\lambda, cpar)
                                                                                          \mathsf{Escrow}(\Lambda, \mathsf{pk}_{\mathtt{A}}, y, r)
hecpar \leftarrow HECsetup(1^{\lambda})
                                                                                          parse \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
return \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
                                                                                          \mathbf{parse}\ \mathsf{pk}_{\mathsf{A}} = (X, C_{\mathsf{A}}, \_)
                                                                                          if VerPK(\Lambda, pk_A, C_A) = 0
\mathsf{KeyGen}(\Lambda, x, r_\mathsf{A})
                                                                                              return 0
parse \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
                                                                                          \hat{Z} \stackrel{\hat{r}_{\hat{Z}}}{\leftarrow} \text{HECeval}(hecpar, f, X, y)
(X, d) \stackrel{r_X}{\leftarrow} \text{HECenc}(hecpar, f, x)
                                                                                          C = \mathsf{Commit}_{cpar}(y; r)
C_{\mathsf{A}} = \mathsf{Commit}_{cpar}(x; r_{\mathsf{A}})
                                                                                          \pi_{\mathsf{U}} \leftarrow \mathsf{PoK}_{\Psi_2}^{\mathsf{S}_2} \Big\{ (y, r, r_{\hat{Z}}) :
\pi_{\mathsf{A}} \leftarrow \mathsf{PoK}_{\Psi_1}^{\mathsf{S}_1} \Big\{ (x, d, r_X, r_{\mathsf{A}}) :
                                                                                              \hat{Z} = \text{HECeval}(hecpar, f, X, y; r_{\hat{z}})
    (X,d) = \text{HECenc}(hecpar, f, x; r_X)
                                                                                               \wedge C = \mathsf{Commit}_{cpar}(y; r) 
     \wedge C_{\mathsf{A}} = \mathsf{Commit}_{cpar}(x; r_{\mathsf{A}})) 
                                                                                          return (\hat{Z}, \pi_{\mathsf{U}})
\mathsf{pk_A} \leftarrow (X, C_\mathsf{A}, \pi_\mathsf{A}); \mathsf{sk_A} \leftarrow (\mathsf{pk_A}, d)
                                                                                          VerEscrow(\Lambda, pk_A, C, Z = (\hat{Z}, \pi_U))
return (pk_{\Delta}, sk_{\Delta})
                                                                                          parse \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
\mathsf{VerPK}(\Lambda, \mathsf{pk}_{\mathsf{A}}, C_{\mathsf{A}})
                                                                                          parse pk_{\Delta} = (\_, C_{A}, \_)
parse \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
                                                                                          return VerPK(\Lambda, pk_A, C_A)
\mathbf{parse}\ \mathsf{pk}_{\mathsf{A}} = (X, C'_{\mathsf{A}}, \pi_{\mathsf{A}})
                                                                                               \wedge \mathsf{V}_2^{\mathsf{S}_2}((\hat{Z}, hecpar, f, X, C, cpar), \pi_{\mathsf{U}})
return V_1^{S_1}((X, hecpar, f, C_A, cpar), \pi_A)
     \wedge (C'_{\mathsf{A}} = C_{\mathsf{A}})
                                                                                          \mathsf{Decrypt}(\Lambda, \mathsf{sk}_{\mathsf{A}}, C, Z = (\hat{Z}, \pi_{\mathsf{U}}))
                                                                                          parse \Lambda = (\lambda, cpar, hecpar, S_1, S_2)
                                                                                          \mathbf{parse}\ \mathsf{sk}_{\mathsf{A}} = (\mathsf{pk}_{\mathsf{A}}, d)
                                                                                          if VerEscrow(\Lambda, pk_A, C, Z) = 0
                                                                                              return 0
                                                                                          return HECDEC(hecpar, d, \hat{Z})
```

Fig. 5.1: Construction of generic f-blueprint scheme

Theorem 2. If HEC is a secure homomorphic-enough cryptosystem, the commitment scheme is binding, and the NIZK PoKs Ψ_1 and Ψ_2 are zero-knowledge and BB-PSL simulation extractable then our generic blueprint scheme is a secure f-blueprint scheme.

Note that, our formal security theorem does not require the commitment to be hiding. It only shows, using simulation, that no additional information besides the commitment is revealed. To benefit from the hiding and privacy properties of the blueprint scheme it is, however, crucial that the transaction system employing it uses a hiding commitment scheme.

We prove correctness of VerEscrow and VerPK, correctness of Decrypt, soundness, blueprint hiding, privacy against dishonest auditor, and privacy with honest auditor in separate lemmas.

Lemma 1. If the NIZK PoKs Ψ_1 and Ψ_2 are complete, then the generic blueprint scheme satisfies correctness of VerEscrow and VerPK.

Proof. Consider VerPK as defined in Fig. 5.1. Suppose the same VerPK returns 0 in the experiment in Fig. ??. Then either $(C_A \neq C'_A)$ or $V_1^{S_1}(pk_A) = 0$. Since $C_A = C'_A = \mathsf{Commit}_{cpar}(y;r)$, the later must be true. However, this contradicts completeness of the NIZK scheme because the proof π_A in pk_A is generated by KeyGen on a valid statement and witness pair.

Similarly, consider VerEscrow as defined in Fig. 5.1. We know that VerPK returns 1, so VerEscrow only returns 0 if $V_2^{S_2}(Z, hecpar, f, X, C, cpar) = 0$. However, similar to the case of VerPK, Z was generated using $P_2^{S_2}$ on a valid statement and witness pair. This again contradicts completeness of the NIZK PoK schemes.

Lemma 2. If the NIZK PoKs Ψ_1 and Ψ_2 are complete and the HEC is correct, then the generic blueprint scheme satisfies correctness of Decrypt.

Proof. Consider Fig. ??. By Lemma 1, we get that Escrow and in extension Decrypt will not return 0, as that requires VerEscrow and VerPK to reject on correct inputs. This tells us that any parts of DecCorrect that relates to the NIZK does not affect the output of the algorithm. Omitting these lines results in:

```
 \begin{aligned} & \frac{\mathsf{DecCorrect}'(\lambda, x, r_\mathsf{A}, y, r)}{1: & \mathit{hecpar} \leftarrow \mathsf{HECSETUP}(1^\lambda)} \\ & 2: & (X, d) \overset{r_X}{\leftarrow} \mathsf{HECENC}(\mathit{hecpar}, f, x) \\ & 3: & \hat{Z} \overset{r_{\hat{Z}}}{\leftarrow} \mathsf{HECEVAL}(\mathit{hecpar}, f, X, y) \\ & 4: & m \leftarrow \mathsf{HECDEC}(\mathit{hecpar}, d, \hat{Z}) \\ & 5: & \mathbf{return} \ [m = f(x, y)] \end{aligned}
```

Fig. 5.2: Experiment DecCorrect_{Blu} (λ, x, r_A, y, r) with all NIZK parts removed

which is the definition for correctness of our HEC scheme for adversaries that hardcode f, x, y and sample $r_{\hat{Z}}$ at random. Assuming HEC is correct, DecCorrect' returns 1 with overwhelming probability, and so does DecCorrect.

Lemma 3. Let Ψ_2 be a BB extractable NIZK scheme, let (CSetup, Commit) be a computationally binding commitment scheme, and HEC be correct with adversarial evaluation randomness, then our proposed scheme achieves Soundness.

Proof. Consider Fig. 3.1. Suppose, for the sake of contradiction, that there exists a PPT adversary \mathcal{A} such that $\mathsf{Adv}^{\mathsf{Sound}}_{\mathcal{A},\mathsf{Blu}}(\lambda) = \nu(\lambda)$ is non negligible. Let Z, one of the adversary's output in the experiment, be divided into \hat{Z} and a proof π to validate \hat{Z} .

The events where \mathcal{A} outputs 1 can be divided into three: (i) when $C = \mathsf{Commit}(y;r)$, $C = \mathsf{Commit}(y';r')$ and $\hat{Z} = \mathsf{HECEVAL}(hecpar, f, X, y'; r_{\hat{Z}})$ for $y \neq y'$, (ii) when $C = \mathsf{Commit}(y;r)$ and $\hat{Z} = \mathsf{HECEVAL}(hecpar, f, X, y; r_{\hat{Z}})$ for some $r_{\hat{Z}}$ where in both (i) and (ii) X is a part of pk_{A} , and (iii) the case where neither of these equalities holds. Let the probability of these events be expressed with functions $\nu_0(\lambda)$, $\nu_1(\lambda)$, and $\nu_2(\lambda)$ respectively. Since $\nu(\lambda)$ is non negligible and these three events covers all cases where \mathcal{A} would output 1, at least one of $\nu_0(\lambda)$, $\nu_1(\lambda)$, or $\nu_2(\lambda)$ must be non negligible.

Suppose $\nu_2(\lambda)$ is non negligible. The adversary produced a proof of a false statement and we can construct a reduction \mathcal{B} to the BB extractable NIZK system. \mathcal{B} runs \mathcal{A} the same way as Sound, see Fig. 3.1, but outputs $(\hat{Z}, hecpar, f, X, C, cpar), \pi_{\mathsf{U}})$ instead. By BB extractability of the NIZK, $\Pr[\mathcal{B} \text{ wins}]$ of extraction failure is negligible, which contradicts our assumption that $\nu_2(\lambda)$ is non negligible.

We now assume that the BB extractor extracts a witness $(y', r', r_{\hat{Z}})$, such that $\hat{Z} = \text{HECeval}(hecpar, f, X, y'; r_{\hat{Z}})$ and $C = \text{Commit}_{cpar}(y'; r')$.

Suppose $\nu_0(\lambda)$ is non negligible. In this event, we break the computational binding property using a reduction that outputs (y, r, y', r').

Suppose $\nu_1(\lambda)$ is non negligible. In this event, we get a situation where both pk_A and Z were generated correctly with adversarial randomness $r_{\widehat{Z}}$, but the output of decrypt is incorrect. We can construct a reduction \mathcal{B} using \mathcal{A} to HEC correctness with adversarial evaluation randomness. \mathcal{B} runs \mathcal{A} , in the same way as Sound, see Fig. 3.1, but instead of returning a bit at the end, it outputs the tuple $(y, r_{\widehat{Z}})$.

Lemma 4. Let Ψ_1 be a NIZK with simulation algorithms (SimS, Sim) and Ψ_2 be a g^* -BB-PSL extractable NIZK with extraction algorithms (Ext, ExtSL) along with an efficiently computable f^* such that $f^*(x, g^*(y)) = f(x, y)$. Let HEC be correct with adversarial evaluation randomness and satisfy security of x. Then our proposed scheme achieves Blueprint Hiding.

Proof. Consider the game in Fig. 3.2. We shall define SimSetup, SimKeygen, and SimDecrypt as follows:

- SimSetup runs HECSETUP, but also sets up state_i to contain the state and query list $\mathcal{Q}_{\mathsf{S}_i}$ of the simulated PoK setup $\mathsf{O}_{\mathsf{S}_i}$ available to \mathcal{A} , P_i , and V_i , for $i \in \{1,2\}$. The state_1 will additionally be programmed by Sim while $\mathcal{Q}_{\mathsf{S}_2}$ will be used for extraction.
- SimKeygen first parses dsim as (ℓ, C_A) , samples $x' \leftarrow domain_{f,x}$ with the same size ℓ as x, and computes $X \leftarrow \text{HECenc}(hecpar, f, x')$. Then it runs $\text{Sim}(\mathsf{state}_1, (X, C_A))$ to produce a simulated proof π_A that X is generated honestly and corresponds to C_A . Finally, it produces pk_A that contains Λ , X, C_A , and the simulated proof.
- SimDecrypt splits Z into \hat{Z} and the proof π_{U} , then runs $\mathsf{ExtSL}(Q_{\mathsf{S}_2},(\hat{Z},C),\pi_{\mathsf{U}})$ to obtain y^* .

Consider the following games:

- Game 0: The left blueprint hiding game BHreal as described in Fig. 3.2.
- Game 1: In this game, we replace Setup with SimSetup, and run a modified version of KeyGen that generates the proof π_A using Sim instead.
- Game 2: Functions identically to the last game, but with Decrypt swapped with SimDecrypt that outputs y^* and returns $f^*(x, y^*)$.
- Game 3: We replace our modified KeyGen with SimKeygen, that is we replace x with a randomly sampled x'. This game is identical to the game BHideal.

Let \mathcal{A} be a fixed probabilistic polynomial time adversary. We denote the interaction with the adversary \mathcal{A} in Game i as $\mathsf{BH}^{\mathcal{A}}_i$. We claim that

$$\left| \Pr \Big[\mathsf{BH}_{j}^{\mathcal{A}}(\lambda) = 0 \Big] - \Pr \Big[\mathsf{BH}_{j+1}^{\mathcal{A}}(\lambda) = 0 \Big] \right|$$

is negligible with respect to λ for $j \in [0, 2]$.

For j=0, we can show this via a reduction to the Zero-Knowledge property of the NIZK PoK system. We can construct a reduction $\mathcal B$ to the NIZK game described in Fig. 2.1. $\mathcal B$ receives two oracles from its challenger, denoted (S',P') and uses S' to answer queries to the S_1 oracle, and P' to generate the proof in the modified KeyGen. Otherwise, $\mathcal B$ interacts with $\mathcal A$ as if it is $\mathcal A$'s challenger in the blueprint security games, and output whatever $\mathcal A$ outputs. Notice that, when in the experiment $\mathsf{NIZK}^{\mathcal B,0}$, $\mathcal B$ functions identically to $\mathcal A$'s challenger in Game 0. Similarly, when in experiment $\mathsf{NIZK}^{\mathcal B,1}$, $\mathcal B$ acts identically as $\mathcal A$'s challenger in Game 1. Therefore, we get that:

$$\begin{split} & \left| \Pr \Big[\mathsf{BH}_0^{\mathcal{A}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{BH}_1^{\mathcal{A}}(1^{\lambda}) = 0 \Big] \right| \\ & = \left| \Pr \Big[\mathsf{NIZK}^{\mathcal{B},0}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{NIZK}^{\mathcal{B},1}(1^{\lambda}) = 0 \Big] \right| = \nu(\lambda) \end{split}$$

for negligible function ν , as required.

For j=1, let us propose a Game 1.5 that is identical to Game 1, but instead of running only Decrypt or SimDecrypt, it runs both. When Decrypt and SimDecrypt agrees, the decryption oracle returns their output. Otherwise, the

experiment is halted and \emptyset is returned. We claim that this event occurs with negligible probability.

Let $\nu(\lambda) = \Pr\left[\mathsf{BH}_{1.5}^{\mathcal{A}}(1^{\lambda}) = \emptyset\right]$. We can separate this into the event E_0 with probability $\nu_0(\lambda)$ that captures the event where the output is \emptyset and extraction fails (i.e. when the output of ExtSL used in SimDecrypt does returns a y^* for which there does not exists valid witness y such that $g^*(y) \neq y^*$ for the relation in the proof in π_{A}), and event E_1 with probability $\nu_1(\lambda)$ that captures all other cases where the experiment would output \emptyset .

 $\nu_0(\lambda)$ must be negligible, otherwise we can construct a reduction \mathcal{B} to the g^* -BB-PSL extractability property of the NIZK described in Fig. 2.2. \mathcal{B} receives two oracles \tilde{O}_S and O_{Sim} and uses \tilde{O}_S to answer \mathcal{A} 's queries to the O_{S_2} oracle. Otherwise, \mathcal{B} acts identically as \mathcal{A} 's challenger in Game 1.5, except that it outputs the statement \hat{Z} and proof π_A when of the first decryption query for which Decrypt and SimDecrypt disagrees.

If the event E_0 occurs, then \mathcal{B} succeeds in its reduction. However, by g^* -BB-PSL simulation extractability, we know that \mathcal{B} succeeds with negligible probability. This gives us:

$$\Pr[E_1] = \nu_0(\lambda) \le \Pr\Big[\mathsf{NISimBBPSLExtract}^{\mathcal{B}}(1^{\lambda}) = 1\Big] = \mu(\lambda)$$

for some negligible μ .

Assume the BB extractor succeeds in extracting a witness $(y, r, r_{\hat{Z}})$, such that $\hat{Z} = \text{HECeval}(hecpar, f, X, y; r_{\hat{Z}})$ and $C = \text{Commit}_{cpar}(y; r)$.

 $\nu_1(\lambda)$ must also be negligible, otherwise we get a situation where both $\mathsf{pk_A}$ and Z were generated correctly with adversarial randomness $r_{\widehat{Z}}$, but the output of decrypt is incorrect. We can construct a reduction \mathcal{B} using \mathcal{A} to HEC correctness with adversarial evaluation randomness. \mathcal{B} first outputs f, x that it receives from \mathcal{A} , and uses the X, d that it receives from its challenger to simulate KeyGen. Then it behaves in the same way as $\mathsf{BH}_{1.5}$, but instead of answering an invalid decryption oracle, it outputs the tuple $(y, r_{\widehat{Z}})$.

For j=2, we can construct a reduction \mathcal{B} to the HEC Security of x game in Fig. 4.1. \mathcal{B} functions identically to the challenger in Game 2, but instead of running KeyGen, it generates the public key pk_A using by first sampling $x' \leftarrow \$$ domain_{f,x}. It then outputs (f,x,x') to its challenger and receives X back. \mathcal{B} then obtains the simulated π_A by running $\mathsf{Sim}(X)$. At the end, it outputs what the adversary outputs.

We see that \mathcal{B} acts identically to \mathcal{A} 's challenger in Game 2 if its challenger chose b=0, and identically to \mathcal{A} 's challenger in Game 3 otherwise. Since the encryption scheme is HEC secure, we get that \mathcal{B} has negligible distinguishing advantage, and so does \mathcal{A} .

Lemma 5. Let (CSetup, Commit) be a computationally binding non-interactive commitment scheme. Let Ψ_1 be a BB extractable NIZK with extraction algorithms Ext, and Ψ_2 be a NIZK with simulation algorithms (SimS, Sim). Let HEC satisfy security of HECDIRECT. Then our proposed construction achieves privacy against dishonest auditor.

Proof. We shall construct SimSetup and SimEscrow for the PADA game as follows:

- 1. SimSetup(cpar, state) runs HECSETUP, but also sets up state_i to contain the state and query list $\mathcal{Q}_{\mathsf{S}_i}$ of the simulated PoK setup $\mathsf{O}_{\mathsf{S}_i}$ available to \mathcal{A} , P_i , and V_i , for $i \in \{1,2\}$. The state_2 will additionally be programmed by Sim while $\mathcal{Q}_{\mathsf{S}_1}$ will be used for extraction.
- 2. SimEscrow(state, Λ_1 , pk_A, Commit(y; r), f(x, y)) first runs $\hat{Z} \leftarrow \text{HECDIRECT}(hecpar, X, f(x, y))$. Afterwards, it creates π_{U} using the ZK simulator. Finally it outputs $Z = (\hat{Z}, \pi_{\mathsf{U}})$.

Consider the following games:

- 1. **Game 0**: The privacy against dishonest auditor game defined in Fig. 3.3 where b=0.
- 2. Game 1: In this game, we change from using Setup to using SimSetup. Additionally, in the Escrow function, instead of using P_2 to generate π_U , we use Sim.
- 3. **Game 2**: Instead of using the modified Escrow procedure, we use SimEscrow. This is effectively the game defined in Fig. 3.3 with b = 1.

Let \mathcal{A} be a fixed PPT adversary, and let PADA_i^{\mathcal{A}} denote \mathcal{A} 's interaction in Game i. We claim that $\left|\Pr\left[\mathsf{PADA}_0^{\mathcal{A}}(1^{\lambda})=0\right]-\Pr\left[\mathsf{PADA}_2^{\mathcal{A}}(1^{\lambda})=0\right]\right|=\nu(\lambda)$ for some negligible function ν .

First, we show that $\left| \Pr \left[\mathsf{PADA}_0^{\mathcal{A}}(1^{\lambda}) = 0 \right] - \Pr \left[\mathsf{PADA}_1^{\mathcal{A}}(1^{\lambda}) = 0 \right] \right|$ is negligible by a reduction \mathcal{B} to the NIZK zero-knowledge game in Fig. 2.1. \mathcal{B} functions as follows:

- 1. \mathcal{B} obtains two oracles (S', P') from its challenger in the NIZK game. It then functions as \mathcal{A} 's adversary in the PADA game, with minor tweaks.
- 2. In setup, \mathcal{B} runs SimSetup but assigns $S_2 \leftarrow S'$.
- 3. When its escrow oracle is queried, it generates \hat{Z} using HECEVAL similar to Escrow, but obtains π_{U} using P' with input \hat{Z} .

Now $\mathcal B$ functions identically to $\mathcal A$'s challenger in Game b for b defined in the NIZK game. We know that NIZK₂ is zero-knowledge, therefore $\mathcal B$'s advantage is negligible and we get:

$$\begin{split} & \left| \Pr \Big[\mathsf{PADA}_0^{\mathcal{A}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{PADA}_1^{\mathcal{A}}(1^{\lambda}) = 0 \Big] \right| = \\ & \left| \Pr \Big[\mathsf{NIZK}^{\mathcal{B},0}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{NIZK}^{\mathcal{B},1}(1^{\lambda}) = 0 \Big] \right| = \nu(\lambda) \end{split}$$

for some negligible function ν , as desired.

Further, we have to show that $\left| \Pr \left[\mathsf{PADA}_1^{\mathcal{A}}(1^{\lambda}) = 0 \right] - \Pr \left[\mathsf{PADA}_2^{\mathcal{A}}(1^{\lambda}) = 0 \right] \right|$ is negligible.

The event where \mathcal{A} outputs x, r_{A} , pk_{A} can be divided into three: (i) when $C_{\mathsf{A}} = \mathsf{Commit}(x; r_{\mathsf{A}})$, $C_{\mathsf{A}} = \mathsf{Commit}(x'; r'_{\mathsf{A}})$ and $X_{\mathsf{A}} = \mathsf{HECenc}(hecpar, f, x'; r_X)$ for

 $x \neq x'$, (ii) when $C_{\mathsf{A}} = \mathsf{Commit}(x; r_{\mathsf{A}})$ and X, $_{-} = \mathsf{HECenc}(hecpar, f, x; r_X)$ for some r_x where in both (i) and (ii) X is a part of pk_{A} , and (iii) the case where neither of these equalities holds.

Let the probability of these events be expressed with functions $\nu_0(\lambda)$, $\nu_1(\lambda)$, and $\nu_2(\lambda)$ respectively. Since $\nu(\lambda)$ is non negligible and these three events covers all cases of \mathcal{A} outputting these values, at least one of $\nu_0(\lambda)$, $\nu_1(\lambda)$, or $\nu_2(\lambda)$ must be non negligible.

Suppose $\nu_2(\lambda)$ is non negligible. The adversary produced a proof of a false statement and we can construct a reduction \mathcal{B} to the BB extractable NIZK system. \mathcal{B} runs \mathcal{A} , but outputs $(X, hecpar, C_A, cpar), \pi_A)$ instead of continuing the game. By BB extractability of the NIZK, $\Pr[\mathcal{B} \text{ wins}]$ is negligible, which contradicts our assumption that $\nu_2(\lambda)$ is non negligible.

Assume the BB extractor succeeds in extracting a witness (x', d, r_X, r'_A) , such that $C_A = \mathsf{Commit}(x'; r'_A)$ and $X, d = \mathsf{HECenc}(hecpar, f, x'; r_X)$.

Suppose $\nu_0(\lambda)$ is non negligible. In this event, we break the computational binding property of the commitment scheme using a reduction that outputs (x, r_A, x', r'_A) .

Suppose that it is only $\nu_1(\lambda)$ that is non negligible. Further, suppose \mathcal{A} is allowed $\ell(\lambda)$ queries to the encryption oracle where ℓ is a polynomial. We use a hybrid reduction \mathcal{B} to the security of HECDIRECT that shows that

$$\left|\Pr\Big[\mathsf{PADA}_1^{\mathcal{A}}(1^{\lambda}) = 0\right] - \Pr\Big[\mathsf{PADA}_2^{\mathcal{A}}(1^{\lambda}) = 0\Big]\right|$$

is negligible. In hybrid i, the first i queries the adversary makes to the oracle are answered as it would in PADA_1 , and the rest are answered as it would be in PADA_2 .

The reduction functions as follows:

- 1. $\mathcal B$ obtains *hecpar* from its challenger, It then uses it in the generation of Λ in SimSetup.
- 2. When given (x, r_A, pk_A) by \mathcal{A} , it runs the BB extractor to obtain (x, r_X)
- 3. It samples $j \leftarrow \$ \{0, 1, \dots, \ell(\lambda) 1\}$.
- 4. For the first j queries that A makes, B answers as it would in PADA₂.
- 5. For query j, \mathcal{B} sends (f, x, r_X, y) to its challenger and obtains \hat{Z} and runs $\mathsf{Sim}(\hat{Z})$ for π_{U} .
- 6. For the rest of the queries, \mathcal{B} answers as it would in PADA₁.

If we are in DIRECTZ₀ and extraction succeeds, then the adversary's view is identical to hybrid j. Similarly, the adversary's view is identical to hybrid j+1 if $\mathcal B$ is in DIRECTZ₁. Note that hybrid 0 is PADA₁ and hybrid $\ell(\lambda)$ is PADA₂. By the hybrid argument, we get that:

$$\nu_1(\lambda) = \ell(\lambda) \big| \Pr \big[\text{DIRECTZ}^{\mathcal{B},0}(1^{\lambda}) = 0 \big] - \Pr \big[\text{DIRECTZ}^{\mathcal{B},1}(1^{\lambda}) = 0 \big] \big|$$
$$= \ell(\lambda) \nu(\lambda)$$

for some negligible function ν , since HECDIRECT is secure. This is still negligible.

Lemma 6. Let $\Psi_1 = (S_1, P_1, V_1)$ be a NIZK proof system and $\Psi_2 = (S_2, P_2, V_2)$ be a g^* -BB-PSL simulation extractable NIZK along with an efficiently computable f^* such that $f^*(x, g^*(y)) = f(x, y)$. Let HEC be correct with adversarial evaluation randomness and satisfy secure of x and y from third parties. Then our proposed construction achieves privacy with honest auditor.

Proof. Let $(\mathsf{SimS}_1, \mathsf{Sim}_1)$ be the setup and simulator functions that satisfies soundness and zero-knowledge for Ψ_1 , and $(\mathsf{SimS}_2, \mathsf{Sim}_2)$ be the setup and simulator functions along with extractor pairs $(\mathsf{Ext}_2, \mathsf{ExtSL}_2)$ that satisfies black-box with partial straight line simulation extractability for g^* for Ψ_2 where ExtSL_2 is the straightline extractor for g^* . Recall that f^* is an efficiently computable function such that $f^*(x, g^*(y)) = f(x, y)$.

We shall construct SimSetup and Sim as follows:

- 1. $\mathsf{SimSetup}(\mathit{cpar}, \mathsf{state})$ runs $\mathsf{HECSETUP},$ but also sets up state_i to contain the state and query list $\mathcal{Q}_{\mathsf{S}_i}$ of the simulated PoK setup $\mathsf{O}_{\mathsf{S}_i}$ available to $\mathcal{A}, \mathsf{P}_i$, and V_i , for $i \in \{1,2\}$. The state_i will additionally be programmed by Sim_i while $\mathcal{Q}_{\mathsf{S}_2}$ will be used for extraction.
- 2. SimEscrow(state, Λ_1 , pk_A, Commit(y;r)) first samples $y \leftarrow s$ domain_{f,y} and obtains $\hat{Z} \leftarrow \text{HECEVAL}(hecpar, f, X, y)$. Afterwards, it runs Sim_2 to generate π_{U} . Finally it outputs $Z = (\hat{Z}, \pi_{\text{U}})$.

Consider the following games:

- **Game 0**: The privacy with honest auditor game defined in Fig. 3.4 where b=0
- Game 1: Here, we change $O_0^{\mathsf{Escrow}}(y,r)$ to do the following:
 - 1. First, it runs $Z \leftarrow \mathsf{Escrow}(\varLambda, \mathsf{pk}_\mathsf{A}, y, r)$
 - 2. Then, it sets $M[\mathbf{C}, Z] = f(x, y)$ and returns Z

Essentially, this caches every O^{Escrow} result for decryption.

- **Game 2**: In this game, we exchange Setup with SimSetup. Then, we change oracle $O_0^{\sf Escrow}$ of the game to run an algorithm identical to Escrow, but instead of generating π_U with P_2 , it generates it by running Sim_2 .
- Game 3: We change O^{Decrypt} to an alternate algorithm O'^{Decrypt} similar to
 O₁ in the blueprint hiding game described in Fig. 3.2:
 - 1. First check if M[C, Z] exists, and if so return M[C, Z]. This handles the simulated escrows.
 - 2. Then check if $VerEscrow(\Lambda, pk_A, C, Z) = 0$, and if so return \bot .
 - 3. Then it obtains α using ExtSL
 - 4. Finally, it returns $f^*(x, \alpha)$.
- Game 4: In this game, we use Sim_1 to simulate the proof π_A . Everything else stays the same.
- Game 5: In this game we change to using O^{Escrow}.
- Game 6: Now, we change to using P_1 to generate the proof π_A .
- Game 7: Now, we change back to using $O^{Decrypt}$. Notice that this is identical to the privacy with honest auditor game defined in Fig. 3.4 where b = 1.

Let \mathcal{A} be a fixed PPT adversary, and let $\mathsf{PWHA}_i^{\mathcal{A}}$ denote \mathcal{A} 's interaction in Game i. We claim that $\left|\Pr\left[\mathsf{PWHA}_0^{\mathcal{A}}(1^{\lambda})=0\right]-\Pr\left[\mathsf{PWHA}_2^{\mathcal{A}}(1^{\lambda})=0\right]\right|=\nu(\lambda)$ for some negligible function ν .

For Game 0 and Game 1, we claim that the output of O^{Decrypt} in both games agree with overwhelming probability, which can be shown via a reduction to soundness of the construction defined in Fig. 3.1. Consider a Game 0.5 where we use the following decrypt oracle:

- 1. Runs $g \leftarrow \mathsf{Decrypt}(\Lambda, \mathsf{sk}_{\mathsf{A}}, \mathbf{C}, Z)$
- 2. Check if $M[\mathbf{C}, \mathsf{Escrow}]$, and if $g \neq M[\mathbf{C}, \mathsf{Escrow}]$ then halts and return \emptyset .
- 3. Otherwise, return q.

Additionally, instead of only recording f(x,y) in M, it also records y,r but does not use this in any computation and $M[\mathbf{C},Z]$ should only return f(x,y). We see that if Game 0.5 does not end with \emptyset , then its behavior is identical to Game 0 or Game 1. We claim that the probability $\Pr\left[\mathsf{PWHA}_{0.5}^{\mathcal{A}}(1^{\lambda}) = \emptyset\right]$ is negligible by a reduction \mathcal{B} to the the soundness game in Fig. 3.1. \mathcal{B} functions identically to \mathcal{A} 's challenger in Game 0.5 with the following modifications:

- 1. Upon receiving Λ from its challenger, it passes that to A.
- 2. When receiving (x, r_A) from A, it passes it to the challenger.
- 3. When about to return \emptyset , instead give its challenger the values (\mathbf{C}, y, r, Z) that was used to call $\mathsf{O}^\mathsf{Decrypt}$.

We see that \mathcal{A} 's view of this game is identical to that of Game 0.5. Additionally, \mathcal{B} succeeds in the reduction (produce an escrow where verification passes but decrypts to an incorrect value) if it would have outputted \emptyset in Game 0.5. However, we know that \mathcal{B} 's probability of success must be negligible by Lemma 3, and so does $\Pr\left[\mathsf{PWHA}_{0.5}^{\mathcal{A}}(1^{\lambda}) = \emptyset\right]$.

For Game 1 and Game 2, we can construct a reduction $\mathcal B$ to the zero knowledge game for NIZK_2 as described in Fig. 2.1. $\mathcal B$ acts as the challenger in Game 1 except the following modifications.

- 1. \mathcal{B} receives (S', P') from its challenger and in S, it assigns $S_2 \leftarrow S'$.
- 2. When performing Escrow in O^{Escrow} , instead of computing π_U with P_2 , \mathcal{B} runs $\pi_U \leftarrow P'((\mathsf{pk}_A, \hat{Z}, \mathsf{Commit}(y; r), \Lambda), (y, r, r'))$ where r' is the value used to compute \hat{Z} .

If \mathcal{B} is interacting in $NIZK^{\mathcal{A},0}$, then \mathcal{A} is interacting with its challenger in Game 1. Similarly, when \mathcal{B} is interacting in $NIZK^{\mathcal{A},1}$, then \mathcal{A} is interacting with its challenger in Game 2. Note that, although the challenger in Game 2 simply runs Sim_2 , the challenger for \mathcal{B} will always pass the if check in O_P because we always provide it a valid witness-statement pair. We get that:

$$\begin{split} & \left| \Pr \Big[\mathsf{PWHA}_1^{\mathcal{A}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{PWHA}_2^{\mathcal{A}}(1^{\lambda}) = 0 \Big] \right| = \\ & \left| \Pr \Big[\mathsf{NIZK}^{\mathcal{B},0}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{NIZK}^{\mathcal{B},1}(1^{\lambda}) = 0 \Big] \right| = \nu(\lambda) \end{split}$$

for some negligible function ν as required, since NIZK₂ is zero-knowledge.

For Game 2 and Game 3, we claim that \mathcal{A} cannot cause the output of $\mathsf{O}^\mathsf{Decrypt}$ and $\mathsf{O}'^\mathsf{Decrypt}$ to disagree with non negligible probability. Consider a Game 2.5 where the challenger instead of running just $\mathsf{O}^\mathsf{Decrypt}$ or $\mathsf{O}'^\mathsf{Decrypt}$, it runs both and outputs the answer if both agrees, or ends the game and output \emptyset otherwise. We claim that $\mathsf{Pr}\left[\mathsf{PWHA}_{2.5}^{\mathcal{A}}(1^\lambda) = \emptyset\right]$ is negligible.

For the sake of simplicity, we assume that all queries \mathcal{A} made to the decryption oracle contains (\mathbf{C}, Z) such that $\mathsf{VerEscrow}(\Lambda, \mathsf{pk_A}, \mathbf{C}, Z) = 1$. This is something that \mathcal{A} itself can check, and if a query is made that does not fit this description, then the O will return with \bot as both $\mathsf{O}^\mathsf{Decrypt}$ and $\mathsf{O}'^\mathsf{Decrypt}$ runs this verification and return \bot when it fails. Consider two cases where \emptyset is the output: when extraction succeeds and when it fails.

We know by simulation extractability of Ψ_2 that the later occurs with negligible probability. Otherwise, we can construct a reduction \mathcal{B} to black-box partial straightline simulation extractability of Ψ_2 . \mathcal{B} runs exactly as the challenger in Game 2.5, but when $O'^{Decrypt}$ returns \bot , it halts execution and return the corresponding $(\Lambda, Z, \mathbf{C}, \mathsf{pk}_{\mathsf{A}})$ tuple that cases extraction to fail. We know Ψ_2 is simulation extractible, so \mathcal{B} 's advantage is negligible, and so is the probability of failure in Game 2.5 as long as the adversary is probabilistic polynomial time.

Consider the case where \emptyset is returned and extraction does not fail, denoted $E_{badproof}$. This happens when Decrypt does not return f(x,y) where y is the result of running Ext in O'Decrypt. We claim that this happens with negligible probability. Otherwise, we can perform a reduction \mathcal{B} to the correctness of HEC. \mathcal{B} functions the same as the challenger in Game 2.5, with the following modifications:

- 1. It receives hecpar from its challenger, and uses it in generating Λ .
- 2. It receives (x, r_A) , passes f, x to its challenger and uses the returned X, d to generate $(\mathsf{pk}_A, \mathsf{sk}_A)$.
- 3. Upon a decryption oracle query with values (C, Z) where extraction does not fail but the outputs disagrees, it pauses execution of \mathcal{A} . Further, \mathcal{B} uses Ext_2 to extract values $(y, r, r_{\hat{Z}})$ from Z via rewinding, Then, it returns $(y, r_{\hat{Z}})$ to the challenger.

Notice that, up to the point that the event in interest occurs, \mathcal{B} acts identically as \mathcal{A} 's challenger in Game 2.5. Additionally, when the event occurs, \mathcal{B} returns a valid reduction. By correctness of HEC, we get that:

$$\Pr[E_{badproof}] = \Pr[\text{HECCORRECT}^{\mathcal{B}}(1^{\lambda}) = \bot] = \nu(\lambda)$$

for some negligible function ν , as required.

Therefore, the decryption oracle in Game 2.5 agrees with that of Game 2 and Game 3 with overwhelming probability.

The logic for Game 3 and 4 is similar to that of Game 1 and Game 2 – a reduction to the zero knowledge property of Ψ_1 described in Fig. effig:NIZK. Here, \mathcal{B} receives (S', P') and assigns $S_1 \leftarrow S'$ in SimSetup. When computing the proof π_A , it uses P'. At the end, it outputs whatever \mathcal{A} outputs.

When \mathcal{B} is in $\mathsf{NIZK}^{\mathcal{B},0}$, \mathcal{B} acts as \mathcal{A} 's challenger in Game 3. Likewise, it acts as \mathcal{A} 's challenger in Game 4 if \mathcal{B} is in $\mathsf{NIZK}^{\mathcal{B},1}$. Therefore, we get:

$$\begin{split} & \left| \Pr \Big[\mathsf{PWHA}_3^{\mathcal{A}}(\lambda) = 0 \Big] - \Pr \Big[\mathsf{PWHA}_4^{\mathcal{A}}(\lambda) = 0 \Big] \right| = \\ & \left| \Pr \Big[\mathsf{NIZK}^{\mathcal{B},0}(\lambda) = 0 \Big] - \Pr \Big[\mathsf{NIZK}^{\mathcal{B},1}(\lambda) = 0 \Big] \right| = \nu(\lambda) \end{split}$$

for some negligible function ν as required, since $NIZK_1$ is zero-knowledge.

For Game 4 and Game 5, we can show the adversary performs negligibly different by a series of hybrid arguments. Suppose \mathcal{A} is allowed to submit $\ell(\lambda)$ queries to the Escrow oracle where ℓ is a polynomial. Let Hybrid i, denoted PWHA_{4,i}, defined for $i \in [0, \ell(k)]$ be the same as Game 5, but for the first i calls to $\mathsf{O}^{\mathsf{Escrow}}$, the challenger answers with $\mathsf{O}^{\mathsf{Escrow}}$, and for subsequent calls it answers with $\mathsf{O}'^{\mathsf{Escrow}}$. We see that Game 4 is the same as $\mathsf{PWHA}_{4,0}$ and Game 5 is the same as $\mathsf{PWHA}_{4,\ell(\lambda)}$.

Consider the following reduction \mathcal{B} to the security of (x,y) as described in Fig. 4.1:

- 1. \mathcal{B} receives *hecpar* and uses it in generating Λ .
- 2. \mathcal{B} outputs (f, x, x) to the challenger when it receives (x, r_{A}) and receives X. It then uses Sim_1 to generate π_{A} corresponding to X and computes $\mathsf{pk}_{\mathsf{A}} = (\Lambda, X, C_{\mathsf{A}} = \mathsf{Commit}(x; r_{\mathsf{A}}), \pi_{\mathsf{A}})$.
- 3. \mathcal{B} then samples $j \leftarrow \$ [0, \ell(\lambda))$ uniform randomly.
- 4. For the first j queries to O^{Escrow} , it computes \hat{Z} as HECEVAL(hecpar, f, X, y).
- 5. For the $(j+1)^{th}$ query, \mathcal{B} samples $y' \leftarrow s$ domain_{f,y} and outputs (y',y) to its challenger. It then receives \hat{Z} from the challenger, and computes π_{U} using Sim_2 on this \hat{Z} .
- 6. For the next queries, it computes Z with Sim the same way as in $O_1^{\sf Escrow}$.

We see that, in $SecXY_0^{\mathcal{B}}$, \mathcal{B} is identical to \mathcal{A} 's challenger in $PWHA_{4,j}$. In $SecXY_1^{\mathcal{B}}$, \mathcal{B} is identical to \mathcal{A} 's challenger in $PWHA_{4,j+1}$. By a hybrid argument, we get that:

$$\begin{split} \left| \Pr \Big[\mathsf{PWHA}_4^{\mathcal{A}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{PWHA}_5^{\mathcal{A}}(1^{\lambda}) = 0 \Big] \right| = \\ \left| \Pr \Big[\mathsf{PWHA}_{4,0}^{\mathcal{A}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathsf{PWHA}_{4,\ell(\lambda)}^{\mathcal{A}}(1^{\lambda}) = 0 \Big] \right| = \\ \ell(\lambda) \big| \Pr \big[\operatorname{SECXY}_0^{\mathcal{B}}(1^{\lambda}) = 0 \big] - \Pr \big[\operatorname{SECXY}_1^{\mathcal{B}}(1^{\lambda}) = 0 \big] \big| = \ell(\lambda) \nu(\lambda) \end{split}$$

for some negligible function ν since HEC is secure. This is still negligible, as required.

The argument for Game 5 and 6 is identical to that of Game 3 and Game 4. The argument for Game 6 and 7 is identical to that of Game 2 and Game 3, except the reduction would make use of the simulator SimS used in $\mathsf{O}_1^{\mathsf{Escrow}}$. Everything else about the reduction stays the same.

6 HEC from the ElGamal Cryptosystem

For a binary string y (or an integer which can be interpreted as a binary string) and an integer k, let $\mathsf{lobits}_k(y) = y \bmod 2^k$; i.e., $\mathsf{lobits}_k(y)$ denotes the k least significant bits of y (or, equivalently, the corresponding integer). Let $domain_{f,y} = \{0,1\}^{l_y}$. Let us use bold font to indicate that \mathbf{x} is a set of values; let $W_\ell = \{\mathbf{x} \mid \mathbf{x} \subseteq domain_{f,y}, |\mathbf{x}| = \ell\}$. Let the function family $F_\ell = \{f_k\}$, where $f_k : W_\ell \times domain_{f,y} \mapsto domain_{f,y}$ is defined as follows:

$$f_k(\mathbf{x}, y) = \begin{cases} y & \mathsf{lobits}_k(y) \in \mathbf{x} \\ \emptyset & \mathsf{otherwise} \end{cases}$$

In other words, the function reveals y if $lobits_k(y) \in \mathbf{x}$, and nothing otherwise.

In this section, we will use the ElGamal cryptosystem in order to construct an HEC for $f_{\ell} \in F_{\ell}$ for any k, ℓ such that ℓ and 2^{l_y-k} are polynomial in λ . Our cryptosystem will use a group G of prime order $q > 2^{l_y}$.

6.1 The ElGamalHEC Construction and Its Security

The idea of our construction ElGamalHEC for a HEC for the aforementioned functions $f_k \in F_\ell$, is that HECENC outputs the ElGamal ciphertexts of the coefficients of a random polynomial P of degree $\ell = |\mathbf{x}|$ whose roots are elements of \mathbf{x} . More precisely, $P = s \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i)$, that is $P = \sum_{i=0}^{|\mathbf{x}|} P_i \chi^i$. The randomness in P comes from the choice of the leading coefficient s. HECENC outputs the ciphertexts $\mathbf{C}_i \leftarrow \mathsf{Enc}(g^{P_i})$ that encrypt the coefficients P_i of P; these ciphertext are part of X.

Using these ciphertexts $\{C_i\}$ and the homomorphic properties of ElGamal, HECEVAL computes an encryption of $g^{rP(\text{lobits}_k(y))+y}$ for a random r. Note that if $\text{lobits}_k(y) \in \mathbf{x}$, this is just an encryption of g^y ; otherwise, it is an encryption of a random element of G. Thus, HECDEC can use the ElGamal decryption algorithm to obtain some group element g^z , and then use the fact that ℓ and 2^{l_y-k} to either recover g with exhaustive search, or determine that $f_k(\mathbf{x},y) = \emptyset$.

Fig. 6.1 describes our construction, ElGamalHEC. Here, (KGen, Enc, Dec) are the key generation, encryption, and decryption algorithms of ElGamal. Recall that \oplus is the homomorphic operator for ciphertexts.

Theorem 3. Under the decisional Diffie-Hellman assumption, ElGamalHEC constitutes a homomorphic-enough encryption for f_k any k, ℓ such that ℓ and 2^{l_y-k} are polynomial in λ , for any $f_k \in F_{\ell}$.

The theorem follows from Lemmas 7, 8, 9, and 10 below.

Lemma 7. Under the decisional Diffie-Hellman assumption, ElGamalHEC satisfies the correctness property of HEC for f_k .

```
HECenc(hecpar, f_k, \mathbf{x})
                                                                                  HECeval(hecpar, f_k, X, y)
 1: (\mathsf{pk}_E, \mathsf{sk}_E) \leftarrow \mathsf{KGen}(1^{\lambda})
                                                                                   1: parse X = (\mathsf{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{|\mathbf{x}|})
                                                                                   2: \quad eval \leftarrow \bigoplus_{i=0}^{|\mathbf{x}|} (\mathbf{C}_i)^{\mathsf{lobits}_k(y)^i}
         P \leftarrow s \prod^{|\mathbf{x}|} (\chi - x_i)
                                                                                   3: enc \leftarrow \mathsf{Enc}(\mathsf{pk}_E, g^y)
         for i in\{0, \ldots, |\mathbf{x}|\}
                                                                                   4: r \leftarrow \mathbb{Z}_q
                                                                                   5: \mathbf{return} \ Z = eval^r \oplus enc
                  \mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_E, q^{P_i})
         return (X = (\mathsf{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{|\mathbf{x}|}),
                                                                                  HECDIRECT(hecpar, X, z)
              d = (\mathsf{sk}_E, f_k, \mathbf{x}))
                                                                                           \mathbf{parse}\ X = (\mathsf{pk}_E, \mathbf{C}_0, \dots, C_{|\mathbf{x}|})
HECDEC(hecpar, d = (sk_E, f_k, \mathbf{x}), Z)
                                                                                   2: if z = \emptyset
                                                                                   3:
                                                                                                \beta \leftarrow \mathbb{Z}_a
 1: D \leftarrow \mathsf{Dec}(\mathsf{sk}_E, Z)
        for y in domain_{f,y} \land \mathsf{lobits}_k(y) \in \mathbf{x}
                                                                                                return Enc(pk_E, q^{\beta})
                                                                                   4:
              if g^y = D
 3:
                                                                                   5: return Enc(pk_E, g^z)
                  return y
        return Ø
```

Fig. 6.1: Our Construction ElGamalHEC

Proof. Let \mathcal{A} be a PPT adversary playing the HEC correctness game with ElGamalHEC. Let $\epsilon_{\mathcal{A}}(1^{\lambda})$ be the probability that the challenger accepts. Below, we (1) provide modified games G_0 and G_1 such that the probability that the challenger in G_0 accepts is also $\epsilon_{\mathcal{A}}(1^{\lambda})$; (2) prove that the probability $\epsilon'_{\mathcal{A}}(1^{\lambda})$ that the challenger in G_1 accepts is negligible; (3) give a reduction \mathcal{B}_{HEC} that breaks the security of the ElGamal cryptosystem with advantage $\epsilon_{\mathcal{A}}(1^{\lambda}) - \epsilon'_{\mathcal{A}}(1^{\lambda})$. Since the ElGamal cryptosystem is secure under the DDH assumption, it follows that, under the DDH assumption, $\epsilon_{\mathcal{A}}(1^{\lambda})$ is negligible.

(1) First, consider the following game G_0 , which is the same as the HEC correctness game with our ElGamal instantiation, except that actual polynomial evaluation instead of homomorphic evaluation. G_0 first obtains $(f, \mathbf{x}, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)$; if $f \in F, \mathbf{x} \in domain_{f,\mathbf{x}}$, then it computes X as HECENC $(hecpar, f_k, \mathbf{x})$, except that it renames P into P_0 and s into s_0 , i.e., it computes a polynomial $P_0(\chi) = s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i)$, where $\{x_i\} = \mathbf{x}$. Next, it invokes \mathcal{A} again to receive $(y, r_Z) \leftarrow \mathcal{A}(\mathsf{state}, X)$; from it, it computes $y' \leftarrow P_0(\mathsf{lobits}_k(y))r_Z + y$. If $(\mathsf{lobits}(y') \in \mathbf{x}) \land (\mathsf{lobits}(y) \notin \mathbf{x})$, accept. That is, instead of a homomorphic evaluation of P_0 using the ciphertexts $C_0, \ldots, C_{|\mathbf{x}|}$, followed by decrypting the resulting ciphertext, it performs an actual evaluation of P_0 . Observe that the probability that the challenger in G_0 accepts is the same as in the original correctness game due to the correctness of homomorphic polynomial evaluation.

Second, consider the game G_1 that proceeds similarly to G_0 : in addition to polynomial P_0 it computes a polynomial $P_1(\chi) = s_1 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i)$ with its own random value s_1 that it uses within HECENC (instead of P_0). Thus, X consists of the ciphertexts that correspond to coefficients of P_1 . Running HECEVAL followed by HECDEC on input y would correspond to homomorphically evaluating $P_1(y)$; instead, G_1 (like G_0) uses P_0 to compute $y' \leftarrow P_0(\mathsf{lobits}_k(y))r_Z + y$ and accepts if $(\mathsf{lobits}(y') \in \mathbf{x}) \land (\mathsf{lobits}(y) \notin \mathbf{x})$.

- (2) Let us prove that the probability $\epsilon'_{\mathcal{A}}(1^{\lambda})$ that the challenger in G_1 accepts is negligible. The challenger will accept only if $|\mathbf{bobits}(y)| \notin \mathbf{x}$, so let us consider this case. Then $P_0(|\mathbf{bobits}_k(y)|) \neq 0$, and, since s_0 is random, $y' = P_0(|\mathbf{bobits}_k(y)|) \neq 0$ is independent of \mathcal{A} 's view. Thus, for any $x \in \mathbf{x}$, $\Pr[|\mathbf{bobits}(y')| = x] \approx 2^{-k}$, thus the probability that G_1 accepts is $\approx |\mathbf{x}|2^{-k}$.
- (3) We construct the reduction \mathcal{B}_{HEC} to the security of the ElGamal cryptosystem. Recall that, under the DDH assumption, the ElGamal cryptosystem is CPA-secure using the formulation of Boneh and Shoup (see Sect. 2); we use this version of (multi-instance) CPA-security in our reduction (this makes the proof simpler as it avoids the hybrid argument). \mathcal{B}_{HEC} creates both polynomials $P_j \leftarrow s_j \prod_{i=1}^{|\mathbf{x}|} (\chi x_i), \ j \in \{0,1\}$. Let $P_{j,i}$ be their coefficients. It obtains the encryption of the coefficients of one of these polynomials via the ElGamal challenger: $\mathbf{C}_i \leftarrow \mathsf{O}_b(g^{P_{0,i}}, g^{P_{1,i}})$. This is described in more detail in Fig. ??. Observe that, $\mathcal{B}_{\text{HEC}}^{\mathsf{O}_b(\cdot,\cdot)}(1^\lambda, \mathsf{pk}_E)$ creates the same view for \mathcal{A} as G_b . Therefore, $\Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}},0}(1^\lambda) = 0] = \epsilon_{\mathcal{A}}(1^\lambda)$ and $\Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}},1}(1^\lambda) = 0] = \epsilon'_{\mathcal{A}}(1^\lambda)$. Since ElGamal is CPA-secure under the DDH assumption, $\epsilon_{\mathcal{A}}(1^\lambda) \epsilon'_{\mathcal{A}}(1^\lambda) \leq \Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}},0}(1^\lambda) = 0] \Pr[\text{IND-CPA}_{\mathcal{B}_{\text{HEC}},1}(1^\lambda) = 0]$ is negligible as required.

Lemma 8. ElGamalHEC satisfies Security of x

Proof. As in the proof of Lemma 7, we will use Boneh and Shoup's multi-use formulation of CPA security. Let \mathcal{A} be a PPT algorithm. Consider the reduction \mathcal{B}_X described in Fig. 6.3; it is constructed so that $p_{A,b}^{\text{SecX}}(\lambda) = \Pr[\text{IND-CPA}_{\mathcal{B}_X,b}(1^{\lambda}) = 0]$. Since ElGamal is secure under the DDH assumption, under the DDH assumption, $|p_{A,0}^{\text{SecX}}(\lambda) - p_{A,1}^{\text{SecX}}(\lambda)| = |\Pr[\text{IND-CPA}_{\mathcal{B}_X,0}(1^{\lambda}) = 0] - \Pr[\text{IND-CPA}_{\mathcal{B}_X,1}(1^{\lambda}) = 0]| = \nu(\lambda)$ for a negligible function ν .

Lemma 9. ElGamalHEC achieves security of x and y from third parties.

Proof. Let \mathcal{A} be a PPT algorithm. Consider the reduction $\mathcal{B}_{XY}^{O_b(\cdot,\cdot)}(1^\lambda,\mathsf{pk}_E)$ in Fig. 6.4. To prove the lemma, it is sufficient to show that $\mathcal{B}_{XY}^{O_b(\cdot,\cdot)}(1^\lambda,\mathsf{pk}_E)$ acts identically to the challenger $\operatorname{SecXY}_b^{\mathcal{A}}$ that \mathcal{A} interacts with in the SecXY game in Fig. 4.1. To see that this is the case, observe that X is computed identically to how \mathcal{A} 's challenger would compute it (for both b=0 and b=1). Note that in both experiments, Z is a random ElGamal encryption of the value z_b which is g^{y_b} if $y_b \in \mathbf{x}_b$ and a random element of $\mathbb G$ otherwise. Thus The value Z as computed by $\mathcal{B}_{XY}^{O_b(\cdot,\cdot)}(1^\lambda,\mathsf{pk}_E)$ is distributed identically to that computed by $\operatorname{SecXY}_b^{\mathcal{A}}$. \square

Lemma 10. ElGamalHEC achieves security of Directz.

Proof. (Sketch) Consider the experiments DIRECTZ₀^A and DIRECTZ₁^A (in Fig. 4.1) for any PPT \mathcal{A} , instantiated with ElGamalHEC. In DIRECTZ₀^A, Z is computed from X and y by homomorphic evaluation that ensures (by the correctness of homomorphic evaluation \oplus), that Z is an encryption of g^y if $f_k(\mathbf{x}, y) = y$ or of a random element of $\mathbb G$ otherwise. In DIRECTZ₁^A, Z is computed by directly encrypting g^y if $f_k(\mathbf{x}, y) = y$, and a random element of $\mathbb G$ otherwise. Thus, in the two experiments, Z is distributed identically.

6.2 From ElGamalHEC to an Efficient Secure Blueprint Scheme

In order to use our HEC construction in Fig. 6.1 to construct our Generic f-blueprint scheme in Fig. 5.1, we need a BB simulation extractable proof system for Ψ_1 to prove knowledge of the witness in the following relation:

$$\left\{ \begin{array}{l} \mathbb{x} = (X, hecpar, f, C_{\mathsf{A}}, cpar), \\ \mathbb{w} = (\mathbf{x}, d, r_X, r_{\mathsf{A}}) \end{array} \right. | \left. \begin{array}{l} (X, d) = \mathrm{HECenc}(hecpar, f, \mathbf{x}; r_X) \land \\ C_{\mathsf{A}} = \mathsf{Commit}_{cpar}(\mathbf{x}; r_{\mathsf{A}}) \end{array} \right\}$$

The building blocks of this relation are statements about the message and randomness of ElGamal encryption and the opening of Pedersen commitments that can be expressed as statements about discrete logarithms representations in R_{eqrep} . By Theorem 1, we have a BB simulation-extractible NIZK proof system for R_{eqrep} and in extension Ψ_1 .

For our specific construction, we assume that the auditor's commitment $C_{\rm A}$ contains commitments to coefficients of the polynomial $P' = \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_i)$. To prove that we encrypted some polynomial P = sP' involves proving that P = sP'. We first prove that we have properly encrypted the coefficients of P'. Then, we can exponentiate these encrypted values by s, effectively multiplying the coefficients by s.

See Appendix B for more details.

Additionally, we require that there exists a f'-BB-PSL simulation extractable proof system for Ψ_2 such that there exists an efficiently computable function f^* where $f^*(\mathbf{x}, f'(y)) = f(\mathbf{x}, y)$ for all $(f, \mathbf{x}, y) \in F \times domain_{f, \mathbf{x}} \times domain_{f, \mathbf{y}}$. Recall that Ψ_2 is used to prove the following relation:

$$\left\{ \begin{aligned} & \mathbf{x} = (\hat{Z}, hecpar, f, X, C, cpar), \\ & \mathbf{w} = (y, r, r_{\hat{Z}}) \end{aligned} \right. \middle| \begin{vmatrix} \hat{Z} = \text{HECeVal}(hecpar, f, X, y; r_{\hat{Z}}) \land \\ & C = \text{Commit}_{cpar}(y; r) \end{aligned} \right\}$$

We need a range proof to prove that $\mathsf{lobits}_k(y)$ is used to generate Eval in \hat{Z} . This can be done using Bulletproofs [13]. The rest of the building blocks for the relation involves statements about ElGamal encryption and Pedersen commitments, we can again be expressed as egrep relation statement.

Theorem 1 guarantees a $f(J, \cdot)$ -BB-PSL simulation extractable NIZK system for eqrep, and in extension Ψ_2 . Recall that $f(J, \mathbf{w}) = \{g^{\mathbf{w}_j} : j \in J\}$. Here, if we choose J to be a singleton containing just the index corresponding to y in \mathbf{w} , we get a g^y -BB-PSL simulation extractable NIZK system. Luckily, knowing \mathbf{x} and

y is sufficient to compute $f(\mathbf{x}, y)$. Here, $f^*(\mathbf{x}, g^y)$ can be computed similar to HECDEC in Fig. 6.1. We first iterate over all y' values such that $\mathsf{lobits}_k(y') \in \mathbf{x}$. If $g^{y'} = g^y$, we return y'. If no such value exists, we return \emptyset . Since $|\mathbf{x}| 2^{l_y - k}$ is polynomial in λ , f^* is efficiently computable.

See Appendix B for more details.

7 HEC for any f from Fully Homomorphic Encryption

Definition 6 (Circuit-private (CP) fully homomorphic encryption (FHE)). A tuple of algorithms (FHEKeyGen, FHEEnc, FHEDec, FHEEval) constitute a secure fully homomorphic public-key encryption scheme [47,11,10,48] if:

Input-output specification: FHEKeyGen(1^{λ} , Λ) takes as input the security parameter and possibly system parameters Λ and outputs a secret key FHESK and a public key FHEPK. FHEEnc(FHEPK,b) takes as input the public key and a bit $b \in \{0,1\}$ and outputs a ciphertext c. FHEDec(FHESK,c) takes as input a ciphertext c and outputs the decrypted bit $b \in \{0,1\}$. FHEEval(FHEPK, Φ , c_1, \ldots, c_n) takes as input a public key, a Boolean circuit Φ : $\{0,1\}^n \mapsto \{0,1\}$, and n ciphertexts and outputs a ciphertext c_{Φ} ; correctness (below) ensures that c_{Φ} is an encryption of $\Phi(b_1, \ldots, b_n)$ where c_i is an encryption of b_i .

Correctness of evaluation: For any integer n (polynomial in λ) for any circuit Φ with n inputs of size that is polynomial in λ , for all $x \in \{0,1\}^n$, the event that $\mathsf{FHEDec}(FHESK,C) \neq \Phi(x)$ where (FHESK,FHEPK) are output by $\mathsf{FHEKeyGen}, c_1,\ldots,c_n$ are ciphertexts where $c_i \leftarrow \mathsf{FHEEnc}(FHEPK,x_i)$, and $c_{\Phi} = \mathsf{FHEEval}(FHEPK,\Phi,c_1,\ldots,c_n)$, has probability 0.

Security: FHE must satisfy the standard definition of semantic security.

Compactness: What makes fully homomorphic encryption non-trivial is the property that the ciphertext c_{Φ} should be of a fixed length that is independent of the size of the circuit Φ and of n. More formally, there exists a polynomial $s(\lambda)$ such that for all circuits Φ , for all (FHESK, FHEPK) output by FHEKeyGen(λ) and for all input ciphertexts c_1, \ldots, c_n generated by FHEEnc(FHEPK, \cdot), c_{Φ} generated by FHEEval(FHEPK, Φ , c_1, \ldots, c_n) is at most $s(\lambda)$ bits long.

An FHE scheme is, additionally, **circuit-private** [47,60,9,36] for a circuit family C if for any PPT algorithm $A |p_{A,0} - p_{A,1}| = \nu(1^{\lambda})$ for a negligible ν , where for $b \in \{0,1\}$, $p_{A,b}$ is the probability that the following experiment outputs 0:

```
\begin{split} & \mathsf{FHECircHideExpt}(1^\lambda) \\ & (R, \Phi_0, \Phi_1, (x_1, r_1), \dots, (x_n, r_n)) \leftarrow \mathcal{A}(1^\lambda) \\ & \mathbf{if} \ \Phi_0 \notin \mathcal{C} \lor \Phi_1 \notin \mathcal{C} \lor \Phi_0(x_1, \dots, x_n) \neq \Phi_1(x_1, \dots, x_n) : \mathbf{reject} \\ & (\mathit{FHEPK}, \mathit{FHESK}) = \mathsf{FHEKeyGen}(1^\lambda; R) \\ & \mathbf{for} \ i \in \{1, \dots, n\} : \\ & c_i = \mathsf{FHEEnc}(\mathit{FHEPK}, x_i; r_i) \\ & Z_0 \leftarrow \mathsf{FHEEval}(\mathit{FHEPK}, \Phi_0, c_1, \dots, c_n) \\ & Z_1 \leftarrow \mathsf{FHEEval}(\mathit{FHEPK}, \Phi_1, c_1, \dots, c_n) \\ & \mathbf{return} \ \mathcal{A}(Z_b) \end{split}
```

Bibliographic note. Definitions of circuit-privacy in the literature come in different flavors; we chose the formulation that makes it easiest to prove Theorem 5 below. The strongest, malicious circuit-privacy [60,36], is strictly stronger than what we give here; therefore, constructions that achieve it automatically achieve the definition here. Constructions of circuit-private FHE from regular FHE have been given by Ostrovsky et al. [60] and by Döttling and Dujmović [36].

Similarly, we chose to formulate correctness as perfect correctness, rather than allowing a negligible probability (over the randomness for the key generation, encryption, and evaluation) of a decryption error. Our construction below also achieves HEC from schemes that are strongly correct, i.e. where the probability of a decryption error is non-zero, but with high probability, no efficient adversary can find a public key and a set of ciphertexts and a circuit that will cause a decryption error. Achieving strong correctness from the more standard notion of correctness with overwhelming probability can be done with standard techniques, see Appendix C

Construction of HEC for any f from CP-FHE. For a Boolean function $g: \{0,1\}^{\ell_x} \times \{0,1\}^{\ell_y} \mapsto \{0,1\}$, an ℓ_y -bit string g and a value $g \in \{0,1\}^2$, let $\Phi_{y,z}^g(x)$ be the Boolean circuit that outputs g(x,y) if g(x,y)

Recall that our goal is to construct a secure f-HEC scheme with a direct encryption algorithm; suppose that the length of the output of f is ℓ ; for $1 \le j \le \ell$, let $f_j(x,y)$ be the Boolean function that outputs the j^{th} bit of f(x,y). Suppose we are given an FHE scheme that is circuit-private for the families of circuits $\{C_j\}$ defined as follows: $C_j = \{\Phi_{y,z}^{f_j}(x) : y \in \{0,1\}^{\ell_y}, z \in \{0,1\}^2\}$.

 $\text{HECSETUP}(1^{\lambda}) \to \Lambda$: Generate the FHE parameters Λ , if needed.

HECENC(1 $^{\lambda}$, Λ , f, x) \rightarrow (X, d) : Generate (FHESK, FHEPK) \leftarrow FHEKeyGen(1 $^{\lambda}$, Λ). Let |x| = n; set $c_i \leftarrow$ FHEEnc(FHEPK, x_i). Output $X = (FHEPK, c_1, \ldots, c_n)$, and decryption key d = FHESK.

HECEVAL(hecpar, $f, X, y) \to Z$: Parse $X = (FHEPK, c_1, \ldots, c_n)$. For j = 1 to ℓ , compute $Z_j \leftarrow \mathsf{FHEEval}(FHEPK, \Phi^{f_j}_{y,00}, c_1, \ldots, c_n)$. Output $Z = (Z_1, \ldots, Z_\ell)$.

 $\text{HECDEC}(hecpar, d, Z) \rightarrow z : \text{Output } (\mathsf{FHEDec}(d, Z_1), \dots, \mathsf{FHEDec}(d, Z_\ell)).$

HECDIRECT(hecpar, X, z) $\to Z$: Parse $X = (FHEPK, c_1, \ldots, c_n)$. For j = 1 to ℓ , compute $Z_j \leftarrow \mathsf{FHEEval}(FHEPK, \Phi^{f_j}_{0^\ell, 1z_j}, c_1, \ldots, c_n)$. Output $Z = (Z_1, \ldots, Z_\ell)$.

Theorem 4. If (FHEKeyGen, FHEEnc, FHEDec, FHEEval) is a fully-homomorphic public-key encryption scheme that is circuit-private for circuit family $\{C_j^f: f \in F\}$ defined above, then our construction above constitutes a homomorphic-enough encryption for the family F.

Proof. (Sketch) Correctness follows from the perfect correctness of FHE. Security of x by semantic security of FHE. Security of x and y from third parties is also by semantic security. Finally, the security of the direct encryption algorithm follows by circuit privacy.

Combining the fact that circuit-private FHE exists if and only FHE exists, and (as we saw earlier) the fact that HEC and simulation-extractable NIZK [34] give us a secure blueprint scheme, we have the following result:

Corollary 1. If fully homomorphic encryption and simulation extractable NIZK exist, then for any function f, secure f-blueprint scheme is realizable.

8 HEC for any f from Non-Interactive Secure Computation

We show how to construct HEC from non-interactive secure computation as defined by [1] and adapted by [50].

Definition 7 (Non-interactive secure computation (NISC) [1]). A tuple of algorithms $N = (\text{GenCRS}, \text{NISC}_1, \text{NISC}_2, \text{Evaluate})$ constitute a secure (multisender) non-interactive secure computation (NISC) scheme for a function $f: \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, if protocol π_f (defined below) using these algorithms emulates the ideal functionality \mathcal{F}_f (also defined below) in the \mathcal{F}_{crs} -hybrid model.

 π_f uses a common reference string crs generated using GenCRS. The receiver takes as input $x \in X$ and runs $\mathsf{NISC}_1(crs, f, x; r)$ to learn the first message X and $\mathsf{NISC}_{private}$. A sender takes as input a value y and runs $\mathsf{NISC}_2(crs, f, y, X)$ to learn Z. Finally the receiver runs $\mathsf{Evaluate}(crs, Z, \mathsf{NISC}_{private})$ to learn the result.

 \mathcal{F}_f accepts $x \in X$ from the receiver and $y_i \in Y$ from the i^{th} sender, and outputs $f(x, y_i)$ to the receiver; senders receive no output. If $y_i = \bot$ is a special input error, then the output to the receiver is \bot .

Construction of HEC for any f from NISC.

Recall that our goal is to construct a secure f-HEC scheme with a direct encryption algorithm. Let f(x, y) be any function, expressed as a circuit. Define the corresponding function f'(x, y') as follows: if $y' = 0 \circ y$, then f'(x, y') returns f(x, y). Else, parse $y' = 1 \circ z$, and return z. (This may require that y' be padded or

truncated appropriately to account for the length of the output of f; without loss of generality, let us assume that f is such that |z| = |y|.) Suppose we are given an NISC scheme that is secure for the function family $F' = \{f' \text{ such that } f \in F\}$; from it, we construct an HEC for the function family F, as follows:

HECSETUP $(1^{\lambda}) \to \Lambda$: Generate the NISC reference string crs using GenCRS. Output $\Lambda = crs$.

HECENC($1^{\lambda}, \Lambda, f, x$) \rightarrow (X, d): Generate $(X, NISC_{private}) \leftarrow \mathsf{NISC_1}(crs, f', x)$. Output X and decryption key $d = NISC_{private}$.

HECEVAL(hecpar, $f, X, y) \to Z$: Compute $Z \leftarrow \mathsf{NISC}_2(hecpar, f', 0 \circ y, X)$. Output Z.

 $\text{HECDEC}(hecpar, d, Z) \rightarrow z : \text{Output Evaluate}(hecpar, Z, d).$

HECDIRECT $(hecpar, X, z) \to Z$: Compute $Z \leftarrow \mathsf{NISC}_2(hecpar, f', 1 \circ z, X)$. Output Z.

Theorem 5. If $(GenCRS, NISC_1, NISC_2, Evaluate)$ is a secure NISC scheme for function family F, then our construction above constitutes a homomorphic-enough encryption for the family F.

Proof. (Sketch) Correctness follows: if correctness didn't hold, then an adversarial environment interacting with an honest sender and receiver evaluating f' in NISC would distinguish the real world (in which the receiver's output is not $f(x,y) = f'(x,0 \circ y)$ with non-negligible probability) from the ideal world (in which the receiver's output is always f(x,y)).

Security for x follows: consider three experiments: $Real_0$, Ideal and $Real_1$. In the $Real_b$ experiment, the adversary is given $\Lambda \leftarrow \text{HECSETUP}(1^{\lambda}) = crs$, selects $f \in F$, x_0 and x_1 , and obtains the X part of the output of $\text{HECENC}(1^{\lambda}, \Lambda, f, x_b) = \text{NISC}_1(crs, f', x_b)$. In the Ideal experiment, crs as well as X are generated using the simulator that must exist because we know that N UC-realizes \mathcal{F}_f . If the adversary distinguishes $Real_b$ from Ideal for either b, then N does not UC-realize \mathcal{F}_f . Therefore, the adversary cannot distinguish $Real_0$ and $Real_1$ and so security for x holds.

Security for x and y from third parties: Let S be the ideal adversary (simulator) that, when interacting with the functionality $\mathcal{F}_{f'}$ for f'. Consider the value Z generated as follows: Z is the value S puts on the channel where the environment observes messages from the honest sender to the honest recipient as a result of their running the protocol. Since both the sender and the recipient are honest, this value is generated independently of their inputs x and y. Let SimZ be the algorithm that outputs Z in this manner. Security for x and y from third parties follows because for all pairs (x_0, y_0) , (x_1, y_1) , the output of SimZ is indistinguishable from the output of HECEVAL($hecpar, f, X_b, y_b$) for $X_b \leftarrow HECENC(1^{\lambda}, \Lambda, f, x_b)$ for $b \in \{0, 1\}$.

Security of HECDIRECT holds: if the property doesn't hold, then an adversary that forms X correctly can distinguish either the real world with $y'=0\circ y$ from the ideal world, or the real world with $y'=1\circ z$ from the ideal world. \square

As HEC and simulation-extractable NIZK [34] give us a secure blueprint scheme, we have the following result:

Corollary 2. If NISC and simulation extractable NIZK exist, then for any function f, secure f-blueprint scheme is realizable.

Acknowledgments

We thank Scott Griffy and Peihan Miao for helpful discussions, and the anonymous referees for constructive feedback. This research was supported by NSF awards #2154170 and #2154941, and by grants from Meta.

References

- Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (May 2014). https://doi. org/10.1007/978-3-642-55220-5_22
- Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (Aug 2000). https://doi. org/10.1007/3-540-44598-6_16
- Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 1087–1098. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516687
- Bangerter, E., Camenisch, J., Lysyanskaya, A.: A cryptographic framework for the controlled release of certified data. In: Security Protocols Workshop. Lecture Notes in Computer Science, vol. 3957, pp. 20–42. Springer (2004)
- Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and noninteractive anonymous credentials. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 356–374. Springer, Heidelberg (Mar 2008). https://doi.org/10. 1007/978-3-540-78524-8_20
- Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629.
 Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_38
- 7. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_38
- 8. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography. https://toc.cryptobook.us/
- Bourse, F., del Pino, R., Minelli, M., Wee, H.: FHE circuit privacy almost for free. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 62–89. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/ 978-3-662-53008-5_3
- 10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262
- 11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS. pp. 97–106. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.12

- Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/ 978-3-030-51280-4_23
- 13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
- 14. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_25
- Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: efficient periodic n-times anonymous authentication. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) Proc. 13th ACM Conference on Computer and Communications Security. pp. 201–210. ACM (2006)
- Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-cash. In: Cramer, R. (ed.) Advances in Cryptology Eurocrypt 2005. LNCS, vol. 3494, pp. 302–321. Springer (2005)
- Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Balancing accountability and privacy using e-cash (extended abstract). In: Prisco, R.D., Yung, M. (eds.) Proceedings of the 5th International Conference on Security and Cryptography for Networks (SCN). Lecture Notes in Computer Science, vol. 4116, pp. 141–155. Springer (2006)
- 18. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EU-ROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer Verlag (2001)
- 19. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: SCN 2002. LNCS, vol. 2576, pp. 268–289 (2003)
- Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: CRYPTO 2004. LNCS, vol. 3152, pp. 56–72 (2004)
- 21. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012. pp. 525–536. ACM Press (Oct 2012). https://doi.org/10.1145/2382196.2382252
- Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/ 978-3-540-45146-4_8
- 23. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052252
- Camenisch, J.L.: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. Ph.D. thesis, ETH Zürich (1998)
- Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 34–63. Springer, Heidelberg (Aug 2020). https://doi.org/ 10.1007/978-3-030-56877-1_2
- Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO '82. pp. 199– 203. Plenum Press (1982)

- 27. Chaum, D.: Blind signature systems. In: CRYPTO '83. pp. 153–156. Plenum (1983)
- 28. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM **28**(10), 1030–1044 (Oct 1985)
- Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: CRYPTO '90. LNCS, vol. 403, pp. 319–327 (1990)
- Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EURO-CRYPT'91. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_22
- Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94.
 LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_19
- Damgård, I.: On σ-protocols. Available at http://www.daimi.au.dk/~ivan/ Sigma.ps (2002)
- 33. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 552–576. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_23
- 34. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_33
- 35. Diaz, J., Lehmann, A.: Group signatures with user-controlled and sequential linkability. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 360–388. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75245-3_14
- 36. Döttling, N., Dujmovic, J.: Maliciously circuit-private FHE from information-theoretic principles. Cryptology ePrint Archive, Report 2022/495 (2022), https://eprint.iacr.org/2022/495
- 37. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (Aug 1984)
- 38. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_5
- 39. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
- Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152– 168. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_10
- 41. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.J.: Practical accountability of secret processes. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 657–674. USENIX Association (Aug 2018)
- Fraser, A., Garms, L., Lehmann, A.: Selectively linkable group signatures—stronger security and preserved verifiability. In: Conti, M., Stevens, M., Krenn, S. (eds.) Cryptology and Network Security. pp. 200–221. Springer International Publishing, Cham (2021)

- 43. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_1
- 44. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: CRYPTO '97. LNCS, vol. 1294, pp. 16–30 (1997)
- 45. Fujisaki, E., Okamoto, T.: Witness hiding protocols to confirm modular polynomial relations. In: The 1997 Symposium on Cryptograpy and Information Security. The Institute of Electronics, Information and Communication Engineers, Fukuoka, Japan (Jan 1997), sCSI97-33D
- Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/ 978-3-031-07085-3_14
- 47. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of STOC 2009. pp. 169–178 (2009)
- 48. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
- Goldwasser, S., Park, S.: Public accountability vs. secret laws: Can they coexist? Cryptology ePrint Archive, Report 2018/664 (2018), https://eprint.iacr.org/ 2018/664
- Green, M., Kaptchuk, G., Laer, G.V.: Abuse resistant law enforcement access systems. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 553–583. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77883-5_19
- Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (May 2011). https://doi. org/10.1007/978-3-642-20465-4_23
- Kilian, J., Petrank, E.: Identity escrow. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 169–185. Springer, Heidelberg (Aug 1998). https://doi.org/10. 1007/BFb0055727
- 53. Libert, B., Nguyen, K., Peters, T., Yung, M.: Bifurcated signatures: Folding the accountability vs. anonymity dilemma into a single private signing scheme. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 521–552. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77883-5_18
- 54. Lysyanskaya, A.: Signature schemes and applications to cryptographic protocol design. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (Sep 2002)
- Lysyanskaya, A., Rivest, R., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H.,
 Adams, C. (eds.) Selected Areas in Cryptography. LNCS, vol. 1758 (1999)
- 56. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable sigma-protocols in the global random-oracle model. Cryptology ePrint Archive, Report 2022/290 (2022), https://eprint.iacr.org/2022/290
- Maurer, U.M.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 09. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (Jun 2009)

- 58. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Proc. 8th ACM Conference on Computer and Communications Security. pp. 116–125. ACM press (Nov 2001)
- 59. Nguyen, K., Guo, F., Susilo, W., Yang, G.: Multimodal private signatures. In: CRYPTO 2022, Part II. pp. 792–822. LNCS, Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_27
- 60. Ostrovsky, R., Paskin-Cherniavsky, A., Paskin-Cherniavsky, B.: Maliciously circuit-private FHE. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 536–553. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_30
- 61. Ostrovsky, R., Skeith III, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_14
- Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140.
 Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
- 63. Sakai, Y., Emura, K., Hanaoka, G., Kawai, Y., Matsuda, T., Omote, K.: Group signatures with message-dependent opening. In: Abdalla, M., Lange, T. (eds.) PAIRING 2012. LNCS, vol. 7708, pp. 270–294. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-36334-4_18
- Scafuro, A.: Break-glass encryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 34–62. Springer, Heidelberg (Apr 2019). https://doi.org/ 10.1007/978-3-030-17259-6_2
- 65. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_18
- 66. Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC'98. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (Feb 1998). https://doi.org/10.1007/BFb0054019

A Deferred Preliminaries

A.1 IND-CPA Security

An encryption scheme (KGen, Enc, Dec) is IND-CPA secure if for any PPT adversary \mathcal{A} participating in the experiment described in Fig. A.1, the advantage

$$\mathsf{Adv}^{\mathrm{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) = \left| \Pr \Big[\mathrm{IND\text{-}CPA}^{\mathcal{A},0}_{\mathsf{Enc}}(1^{\lambda}) = 0 \Big] - \Pr \Big[\mathrm{IND\text{-}CPA}^{\mathcal{A},1}_{\mathsf{Enc}}(1^{\lambda}) = 0 \Big] \right| = \nu(\lambda)$$

is some negligible function ν .

Fig. A.1: IND-CPA game

Note that the standard definition for IND-CPA only allows an adversary to query the oracle once. It can be shown that our definition for IND-CPA is equivalent [8].

A.2 Simulation Soundness and Simulation Extraction

Simulation Soundness Let $\Phi = (S, P, V)$ be an NIZK proof system satisfying the zero-knowledge property above; let (SimS, Sim) be the simulator. Φ provides simulation soundness if for any PPT adversary \mathcal{A} participating in the game defined in Fig. A.2, the advantage function $\nu(\lambda)$ defined below is negligible. In Fig. A.2, \mathcal{Q} denotes the query tape that records the instances x for which \mathcal{A} has obtained proofs π via oracle access to the simulator Sim; this is explicitly recorded by O_{Sim} .

$$\mathsf{Adv}^{\mathsf{NISimSound}}_{\mathcal{A}}(\lambda) = \Pr\Big[\mathsf{NISimSound}^{\mathcal{A}}(1^{\lambda}) = 1\Big] = \nu(\lambda)$$

for some negligible function ν .

Straight-Line (SL) Simulation Extractability Let $\Phi = (S, P, V)$ be an NIZK proof system satisfying the zero-knowledge property above; let (SimS, Sim) be the simulator. Φ is simulation-extractable if there exists a polynomial-time extractor algorithm Ext such that for for any PPT adversary A participating in the game defined in Fig. A.3, the advantage function $\nu(\lambda)$ defined below is negligible.

$$\mathsf{Adv}^{\mathsf{NISimExtract}}_{\mathcal{A}}(\lambda) = \Pr \Big[\mathsf{NISimExtract}^{\mathcal{A}}(1^{\lambda}) = 1 \Big] = \nu(\lambda)$$

In Fig. A.3, Q denotes the simulator query tape. Q_S denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by O_S and \tilde{O}_S . As discussed below, \tilde{O}_S additionally reveals to A the extraction trapdoor τ_{Ext} ; this captures adaptive extraction from many proofs.

Fig. A.3: NISimExtract game: τ_{Ext} is only returned by $\tilde{\mathsf{O}}_{\mathsf{S}}(m)$

Black-Box (BB) Simulation Extractability Let $\Phi = (S,P,V)$ be an NIZK proof system satisfying the zero-knowledge property above; let (SimS, Sim) be the simulator. Φ is black-box simulation-extractable if there exists a polynomial-time extractor algorithm Ext such that for any PPT adversary \mathcal{A} participating in the game defined in Fig. A.4, the advantage function $\nu(\lambda)$ defined below is negligible. As before, \mathcal{Q} denotes the query tape. \mathcal{Q}_{Ext} denotes the setup query tape that records the queries, replies, and embedded trapdoors of the simulated setup; this is explicitly recorded by O_S . In fact, the game here is identical to that in Fig. A.3, except now Ext also gets access to BB(\mathcal{A}).

$$\mathsf{Adv}^{\mathsf{NISimBBExtract}}_{\mathcal{A}}(\lambda) = \Pr \Big[\mathsf{NISimBBExtract}^{\mathcal{A}}(1^{\lambda}) = 1 \Big] = \nu(\lambda)$$

for some negligible function ν .

Fig. A.4: NISimBBExtract game: τ_{Ext} is only returned by $\tilde{O}_{S}(m)$

It is straightforward to see that black-box simulation extractability is as strong or weaker than regular simulation extractability, because anything that an extractor can do without black-box access to \mathcal{A} it can also do with it (it just won't use it). It is somewhat less obvious that in fact the resulting flavor of knowledge extraction is inferior, in the sense that a protocol that uses regular simulation extractability may have better security properties than one that uses the black-box flavor. The problem arises when the proof of security of such a protocol tries to use the extractor: this extractor needs $\mathsf{BB}(\mathcal{A})$, which requires resetting the adversary to a previous state and replaying its view. Sometimes (in some proofs of security) that also requires resetting the overall security experiment, which is something that a security reduction may not always be able to do. For that reason, regular (straight-line) extraction is preferred, when it can be achieved.

B Technical Details of the NIZK proof systems for ElGamal Instantiation of Generic f-blueprint Scheme

Recall that in our Generic f-Blueprint scheme from HEC in Fig. 5.1, we need a BB-extractible NIZK system for the following relation for Ψ_1 :

$$\left\{ \begin{aligned} & \mathbb{X} = (X, hecpar, f, C_{\mathsf{A}}, cpar), \\ & \mathbb{W} = (x, d, r_X, r_{\mathsf{A}}) \end{aligned} \right. \mid \left. \begin{aligned} & (X, d) = \mathsf{HECenc}(hecpar, f, x; r_X) \land \\ & C_{\mathsf{A}} = \mathsf{Commit}_{cpar}(x; r_{\mathsf{A}}) \end{aligned} \right\}$$

In our instantiation found in Fig. 6.1, in HECENC we first compute an ElGamal key pair using KGen. Then, we sample some $s \in \mathbb{Z}_q \setminus \{0,1,\ldots,2^k-1\}$. Finally, we encrypt coefficients of the polynomial $P = (\chi - s) \prod_{i=1}^{|x|} (\chi - x_i)$.

Recall also that we assume the commitment C_A contains commitments to coefficients of $P' = \prod_{i=1}^{|x|} x_i$. More formally, let P'_i denote the i^{th} coefficient of P', Then we have commitments $C_A = \mathsf{Commit}_{cpar}(P'_0, P'_1, \dots, P'_{|x|+1}; r_A)$.

Let Ψ be a NIZK proof system for R_{eqrep} , we can construct Ψ_1 with P_{Ψ_1} as described in Fig. B.1. We assume r_X can be split into r_E used in generating pk_E and r_s used in generating the additional root s to the polynomial, and r_i for $i \in 0, \ldots, |x|$ used in encrypting \mathbf{C}_i . In Ψ_1 , we have a commitment to P', but have to prove that we have encrypted $(\chi - s)P'$. We can express $P_i = P'_{i-1} - sP'_i$, which gives us $P'_{i-1} = P_i + sP'_i$ that we can use to compute encryptions to coefficients of P'. We use this to compute an encryption of P' from the encryption of P and the root s and publicize it (effectively including it in the proof), along with proof that it was computed from encrypted coefficients of P. This does not give away information about s, otherwise we would violate the DDH assumption. Observe that, other than proving $s \geq 2^k$, which can be done using Bulletproofs, all other components of our proof system involves proof of equivalent discrete logarithm representation.

Recall that Ψ_2 is used to prove the following relation:

$$\left\{ \begin{aligned} & \mathbf{x} = (\hat{Z}, hecpar, f, X, C, cpar), \\ & \mathbf{w} = (y, r, r_{\hat{Z}}) \end{aligned} \right. , \left| \begin{aligned} & \hat{Z} = \text{HECeval}(hecpar, f, X, y; r_{\hat{Z}}) \land \\ & C = \text{Commit}_{cpar}(y; r) \end{aligned} \right. \right\}$$

In our instantiation found in Fig. 6.1, in HECEVAL we first evaluate our private input y on the encrypted polynomial \mathbf{C} to obtain eval. Then, we encrypt g^y to enc, and sample $r \leftarrow \mathbb{Z}_q$. Finally, we output $eval^r \oplus enc$. Our proof system Ψ_2 , detailed in Fig. B.2, also goes through its steps in this order, and again utilizes Ψ for R_{eqrep} . The verifier $\mathsf{V}_{\Psi_2}^{\mathbb{S}}$ works by first verifying the proof π , then verify:

$$\hat{Z} = (t^{-U}X\prod_{i=0}^{|x|+1}G_i^{AR}, t^{-U'}YZ\prod_{i=0}^{|x|+1}H_i^{AR})$$

And that (U, O) and (U', O') are openings to $\mathcal{X} \prod_{i=0}^{|x|+1} \mathcal{R}_i$ and $\mathcal{YZ} \prod_{i=0}^{|x|+1} \mathcal{R}'_i$ respectively, both under base (g, h).

Let's walk through P_{Ψ_2} . We deal with proving $a = \mathsf{lobits}_k(y)$ by first proving $a < 2^k$, then $a \equiv y \mod 2^k$, which is captured by $1 = g^{y-a}(g^{2^k})^{-m}$. Next, the idea is that we are computing Pedersen commitments A_i^R to $a^i r$, that can later be used to compute commitments to components of $\mathbf{C}_i^{a^i r}$ composed with some added noise $(t^{\rho_i}, t^{\rho_i'})$. Additionally, we compute Pedersen commitments X, Y, and Z where (X, YZ) correspond to $\mathsf{Enc}(\mathsf{pk}_E, y; r_{\mathsf{Enc}})$ composed with some noise $(t^{\gamma_X}, t^{\gamma_Y + \gamma_Z})$. After line 24, we reveal the cumulated noise's exponents U and U', which allows \hat{Z} to be revealed. This still perfectly hides our witness, since a Pedersen commitment is hiding. To prove that the U and U' we reveal are indeed the cumulated noise, we generate Pedersen commitments of the old noise captured in U and U', and reveal the cumulative new noise O and O', and we can verify that (U, O) and (U', O') are openings to the composition of the new Pedersen commitments.

When we refer to published commitments correctly computed in PoK, we are indicating that whenever P runs publish on a committed value of the form $A \leftarrow \mathsf{Commit}_{B,C}(D,E)$, we include the line $A = \mathsf{Commit}_{B,C}(D;E)$ in the PoK

statement on Line 38. In most of our calls to Commit, the arguments are all either public or members of the witness, so we can directly including them in the PoK statement. There is a slight nuance involving proving that G_i^{AR} and H_i^{AR} are correctly computed. Here, we know that $A_i^R = \operatorname{Commit}_{A_i,h}(r';\beta_i)$ can be expressed as $A_i^R = \operatorname{Commit}_{g,h}(a^ir;\beta_i + \sum_{j=0}^i \alpha_j)$, which can be proven by induction on i. We can instead construct a sigma protocol for equality of representation of two Pedersen commitments, which also reduces to an instance of R_{egrep} and therefore can be included in our proof.

Apart from the one range proof we need to use at the start to ensure a is within range, the rest of the statements within the proof are instances of eqrep. Therefore, this construction gives us a $f(J,\cdot)$ -BB-PSL simulation extractable NIZK proof system by Theorem 1.

C Achieving strongly correct FHE

The correctness requirement as stated in Definition 6 is stronger than standard correctness. Namely, we need to ensure that, for the specific circuit Φ used in our construction, the probability that the adversary can find random coins R for FHEKeyGen, and inputs x_1, \ldots, x_n , and randomness r_1, \ldots, r_n for FHEEnc such that FHEDec(FHESK, C) $\neq \Phi(x)$ where FHEPK, FHESK) = FHEKeyGen($1^{\lambda}; R$), $c_i = FHEEnc(FHEPK, x_i; r_i)$ and $C = FHEEval(FHEPK, \Phi, c_1, \ldots, c_n)$, is 0, i.e., the scheme is perfectly correct. Known constructions of FHE either have perfect correctness or can be adapted to achieve it by bounding the amount of random noise, and therefore stating correctness this way is justifiable.

If we allow a negligible probability that an adversary can make a decryption error happen, then we get another correctness notion, let us refer to it as *strong correctness*. It is easy to see that strong correctness is good enough for our construction of HEC from CP-FHE: Strong correctness ensures that even in the event when the adversary controls the choice of the keys and input ciphertexts, as long as he is following the protocol, he cannot (with more than negligible probability) create a situation in which decryption is incorrect; and therefore an incorrect decryption event cannot leak information about the inputs.

Regular (not perfect or strong) correctness requires only that a decryption error occurs negligibly often when keys are ciphertexts are selected honestly. In the common-random-string model, strong correctness can be achieved from regular correctness by standard techniques, which we describe here. Here is a standard technique first seen in Naor's commitment scheme. Let the circuit Φ be fixed. Suppose that for a random public key and a random set of n ciphertexts, the probability of incorrect decryption is p < 1/10. By the Chernoff bound, repeated encryption and taking the majority when decrypting, we can transform this cryptosystem into one whose probability (over the choice of randomness for the public key and the set of input ciphertexts) of incorrect decryption is $2^{-(n+2)(\lambda+1)}$.

Let the common random string consist of n+1 random strings $\rho_0, \rho_1, \ldots, \rho_n$. ρ_0 has length $\ell_{\mathsf{FHEKeyGen}}$, where $\ell_{\mathsf{FHEKeyGen}}$ is an upper bound on the number of

random bits needed for FHEKeyGen. For $1 \le i \le n$, ρ_i has length ℓ_{FHEEnc} , where ℓ_{FHEEnc} is an upper bound on the number of random bits needed for FHEEnc.

In order to generate a public key, sample a λ -bit random seed s_0 , and let $R=G(s_0)\oplus \rho_0$; output $(FHEPK,FHESK)=\mathsf{FHEKeyGen}(1^\lambda;R)$ where G is a PRG with appropriate lengths. In order to generate a the i^{th} ciphertext, sample a λ -bit random seed s_i , and let $r_i=G'(s_i)\oplus \rho_i$; output $c_i=\mathsf{FHEEnc}(FHEPK,x_i;r_i)$ where G' is a PRG with appropriate lengths. Note that there are only $2^{(n+1)(\lambda+1)}$ ways of setting these random strings, and for each of them the probability that truly random $\rho_i's$ are chosen so as to lead to a decryption error is $2^{-(n+2)(\lambda+1)}$ (since that's the probability of a decryption error). Thus, by the union bound, the probability that there exists a way to set the randomness for key generation and ciphertexts so as to lead to a decryption error is $2^{(n+1)(\lambda+1)}2^{-(n+2)(\lambda+1)}=2^{-\lambda-1}$, and the resulting FHE is strongly correct.

```
\mathcal{B}^{\mathsf{O}_b(\cdot,\cdot)}_{\mathrm{HEC}}(1^\lambda,\mathsf{pk}_E)
                                                                                        \mathcal{B}_{XY}^{\mathsf{O}_b(\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_E)
  1: hecpar \leftarrow HECsetup(\lambda)
                                                                                          1: hecpar \leftarrow HECsetup(1^{\lambda})
                                                                                         2: (f, \mathbf{x}_0, \mathbf{x}_1, \mathsf{state}) \leftarrow \mathcal{A}(hecpar)
  2: (f, \mathbf{x}, \mathsf{state}) \leftarrow \mathcal{A}(1^{\lambda}, hecpar)
 3: if f \in F, \mathbf{x} \in domain_{f,\mathbf{x}} 3: if f \in F, \mathbf{x}_0, \mathbf{x}_1 \in domain_{f,\mathbf{x}}
               s_0 \leftarrow \mathbb{Z}_q^*; \qquad s_1 \leftarrow \mathbb{Z}_q^* \qquad 4: \qquad s_0 \leftarrow \mathbb{Z}_q^*; \qquad s_1 \leftarrow \mathbb{Z}_q^*
P_0 \leftarrow s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i) \qquad \qquad 5: \qquad P_0 \leftarrow s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_{0,i})
P_1 \leftarrow s_1 \prod_{i=1}^{|\mathbf{x}|} (\chi - x_i) \qquad \qquad 6: \qquad P_1 \leftarrow s_1 \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_{1,i})
  6:
                                                                                         7: for i in\{0, |\mathbf{x}_0|\}
                 for i in \{0, ..., |x|\}
  7:
                                                                                         8: \mathbf{C}_i \leftarrow \mathsf{O}_b(g^{P_{0,i}}, g^{P_{1,i}})
                 \mathbf{C}_i \leftarrow \mathsf{O}_b(g^{P_{0,i}}, g^{P_{1,i}})
  8:
                                                                                                   X \leftarrow (\mathsf{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{|\mathbf{x}_0|})
                 X \leftarrow (\mathsf{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{|\mathbf{x}|})
  9:
                                                                                                          (y_0, y_1, \mathsf{state}) \leftarrow \mathcal{A}(X, \mathsf{state})
                                                                                        10:
                 (y, r_Z) \leftarrow \mathcal{A}(\mathsf{state}, X)
10:
                                                                                        11:
                                                                                                         if y_0, y_1 \in domain_{f,y}
                 if y \in domain_{f,y}
11:
                                                                                                              eval \leftarrow \mathsf{O}_b(\mathsf{pk}_E, g^{P_0(\mathsf{lobits}_k(y_0))},
                      parse r_Z = (r, r_{\sf Enc})
12:
                                                                                        12:
                       y' \leftarrow P_0(\mathsf{lobits}_k(y))r + y
                                                                                                                    g^{P_1(\mathsf{lobits}_k(y_1))}
13:
                                                                                        13:
                       if (\mathsf{lobits}_k(y') \in \mathbf{x})
14:
                                                                                                              enc \leftarrow \mathsf{O}_b(\mathsf{pk}_E, y_0, y_1)
                                                                                        14:
                             \land (\mathsf{lobits}_k(y) \notin \mathbf{x})
15:
                                                                                                              r \leftarrow \mathbb{Z}_q
16:
                                 return 1
                                                                                                               Z \leftarrow eval^r \oplus enc
                                                                                        16:
                       return 0
17:
                                                                                                               return A(Z, state)
                                                                                        17:
                  return 0
18:
                                                                                                          return A(\perp, state)
            {f return} \ 0
19:
                                                                                        19: return A(\perp, state)
\mathcal{B}_{X}^{\mathsf{O}_{b}(\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_{E})
  1: hecpar \leftarrow HECsetup(\lambda)
  2: (f, \mathbf{x}, \mathbf{x}', \mathsf{state}) \leftarrow \mathcal{A}(hecpar)
  s_1 : s_0 \leftarrow \mathbb{Z}_q^*; \qquad s_1 \leftarrow \mathbb{Z}_q^*
 4: P = s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_i); P' = s_1 \prod_{i=1}^{|\mathbf{x}'|} (\chi - \mathbf{x}'_i)
  5: for i in{0, |\mathbf{x}|}
                 \mathbf{C}_i = \mathsf{O}_b(g^{P_i}, g^{P'_i})
  7: X = (\mathsf{pk}_E, \mathbf{C}_0, \dots, C_{|\mathbf{x}|})
  8: return \mathcal{A}(hecpar, X, state)
```

Fig. 6.2: Reductions \mathcal{B}_{HEC} , \mathcal{B}_X and \mathcal{B}_{XY} for the proof of Theorem 3

$$\begin{array}{ll} \underline{ \text{IND-CPA}_{\mathcal{B},b}(1^{\lambda}) } & \underline{ O_b(\mathsf{pk}_E,m_0,m_1) } \\ 1: & (\mathsf{pk}_E,\mathsf{sk}_E) \leftarrow \mathsf{s} \ \mathsf{KGen}(1^{\lambda}) \\ 2: & \mathbf{return} \ \mathcal{B}^{O_b(\mathsf{pk}_E,\cdot,\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_E) \\ \\ \underline{ \mathcal{B}^{O_b(\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_E) \ \text{from} \ \mathcal{A} \ \text{attacking security of} \ \mathbf{x} \ \text{of} \ \mathsf{ElGamalHEC} } \\ 1: & hecpar \leftarrow \mathsf{HECSETUP}(\lambda) \\ 2: & (f,\mathbf{x},\mathbf{x}',\mathsf{state}) \leftarrow \mathcal{A}(hecpar) \\ 3: & s_0 \leftarrow \mathsf{s} \ \mathbb{Z}_q^* \\ 4: & s_1 \leftarrow \mathsf{s} \ \mathbb{Z}_q^* \\ 5: & P = s_0 \prod_{i=1}^{|\mathbf{x}'|} (\chi - \mathbf{x}_i) \\ 6: & P' = s_1 \prod_{i=1}^{|\mathbf{x}'|} (\chi - \mathbf{x}_i') \\ 7: & \mathbf{for} \ i \ \mathbf{in}\{0,|\mathbf{x}|\} \\ 8: & \mathbf{C}_i = \mathsf{O}_b(g^{P_i},g^{P_i'}) \\ 9: & X = (\mathsf{pk}_E,\mathbf{C}_0,\ldots,C_{|\mathbf{x}|}) \\ 10: & \mathbf{return} \ \mathcal{A}(hecpar,X,\mathsf{state}) \\ \end{array}$$

Fig. 6.3: Reduction from security of X to IND-CPA for the proof of Lemma 8

```
IND-CPA<sub>C,b</sub>(1^{\lambda})
                                                                                 \mathsf{O}_b(\mathsf{pk}_E, m_0, m_1)
 {\scriptstyle 1: \quad (\mathsf{pk}_E, \mathsf{sk}_E) \leftarrow \$ \; \mathsf{KGen}(1^\lambda)}
                                                                                  1: return Enc(pk_E, m_b)
 2: return C^{\mathsf{O}_b(\mathsf{pk}_E,\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_E)
\mathcal{B}^{\mathsf{O}_b(\cdot,\cdot)}(1^{\lambda},\mathsf{pk}_E)
 1: hecpar \leftarrow HECsetup(1^{\lambda})
 2: (f, \mathbf{x}_0, \mathbf{x}_1, \mathsf{state}) \leftarrow \mathcal{A}(hecpar)
  3: if f \in F, \mathbf{x}_0, \mathbf{x}_1 \in domain_{f,\mathbf{x}}
                s_0 \leftarrow \mathbb{Z}_q^*
                s_1 \leftarrow \mathbb{Z}_q^*
 5:
                P_0 \leftarrow s_0 \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_{0,i})
  6:
                P_1 \leftarrow s_1 \prod_{i=1}^{|\mathbf{x}|} (\chi - \mathbf{x}_{1,i})
                for i \text{ in}\{0, |\mathbf{x}_0|\}
  8:
                   \mathbf{C}_i \leftarrow \mathsf{O}_b(g^{P_{0,i}}, g^{P_{1,i}})
 9:
                 X \leftarrow (\mathsf{pk}_E, \mathbf{C}_0, \dots, \mathbf{C}_{|\mathbf{x}_0|})
10:
                 (y_0, y_1, \mathsf{state}) \leftarrow \mathcal{A}(X, \mathsf{state})
11:
                if y_0, y_1 \in domain_{f,y}
12:
                     eval \leftarrow \mathsf{O}_b(\mathsf{pk}_E, g^{P_0(\mathsf{lobits}_k(y_0))}, g^{P_1(\mathsf{lobits}_k(y_1))})
13:
                     enc \leftarrow \mathsf{O}_b(\mathsf{pk}_E, g^{y_0}, g^{y_1})
14:
                     r \leftarrow \mathbb{Z}_q
15:
                      Z \leftarrow eval^r \oplus enc
16:
                      return A(Z, state)
17:
18:
                 \mathbf{return}\ \mathcal{A}(\bot,\mathsf{state})
           return A(\perp, state)
19:
```

Fig. 6.4: Reduction from security of XY to IND-CPA

```
\begin{tabular}{lll} NISimSound^{\mathcal{A}}(1^{\lambda}) & & & & \\ \hline 1: & \mathcal{Q} \leftarrow [ \ ] & & & \\ 2: & (\mathbbm{x},\pi) \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{S}}(\cdot),\mathsf{O}_{\mathsf{Sim}}(\cdot)}(1^{\lambda}) & & & \\ 3: & \mathbf{return} \ \mathsf{V}^{\mathsf{O}_{\mathsf{S}}}(\mathbbm{x},\pi) \wedge (\mathbbm{x},\pi) \not\in \mathcal{Q} \wedge \forall \mathbbm{x}.(\mathbbm{x},\mathbbm{w}) \not\in \mathcal{R} & & \\ \hline \mathsf{O}_{\mathsf{S}}(m) & & & \mathsf{O}_{\mathsf{Sim}}(\mathbbm{x}) & & \\ 1: & \mathsf{state},h,\tau_{\mathsf{Ext}} \leftarrow \mathsf{SimS}(\mathsf{state},m) & & 1: & \mathsf{state},\pi \leftarrow \mathsf{Sim}(\mathsf{state},\mathbbm{x}) \\ 2: & \mathbf{return} \ h & & 2: & \mathcal{Q}.\mathsf{add}((\mathbbm{x},\pi)) & & \\ & & 3: & \mathbf{return} \ \pi & & & \\ \hline \end{tabular}
```

 ${\rm Fig.\,A.2:\,NISimSound\,\,game}$

```
\mathsf{P}_{\Psi_1}^{\mathsf{S}}((hecpar, f, cpar, X, C_{\mathsf{A}}), (x, d, r_X, r_{\mathsf{A}}))
 1: parse X = (pk_E, C_0, ..., C_{|x|+1})
 2: parse r_X = (r_E, r_s)
 3: parse cpar = (g_0, ..., g_{|x|}, H)
 4: \quad e \xleftarrow{r_E} \mathbb{Z}_q
 5: (\mathsf{pk}_E, \mathsf{sk}_E) \leftarrow \mathsf{KGen}(hecpar; r_E) = ((g, h = g^e), e)
 6: \quad s \stackrel{r_s}{\longleftarrow} \mathbb{Z}_q \setminus \left\{0, \dots, 2^k - 1\right\}
 7: \quad P' = \prod_{i=1}^{|x|} (\chi - x_i)
 8: \quad P = (\chi - s)P'
 9: \mathbf{C}'_{|x|} = \mathbf{C}_{|x|+1}
10: r'_{|x|} = r_{|x|+1}
11: for i \in \{|x|-1,\ldots,0\}
         \mathbf{C}_i' = \mathbf{C}_{i+1} \oplus (\mathbf{C}_{i+1}')^s
13: r'_i = r_{i+1} + s \cdot r_{i+1}
14: publish (\mathbf{C}_0',\ldots,\mathbf{C}_{|X|}')
\text{15}: \quad \pi \leftarrow \mathsf{PoK}_{\varPsi} \Big\{ \mathbb{w} = (P'_0, \dots, P'_{|x|}, r_0, \dots, r_{|x|+1}, r'_0, \dots, r'_{|x|}, s, r_{\mathsf{A}}, e) :
          h=g^e\wedge s\geq 2^k\wedge C_{\mathsf{A}}=H^{r_{\mathsf{A}}}\prod_{i=0}^{|x|}g_i^{P_i'}\wedge
          \mathbf{C}_i' = (g^{r_i'}, g^{P_i'} h^{r_i'}) 	ext{ for } i \in \{0, \dots, |x|\} \land
              \mathbf{C}_0 = (\mathbf{C}_0')^s \wedge \mathbf{C}_i = \mathbf{C}_{i-1}' \oplus (\mathbf{C}_i')^{-s} \text{ for } i \in \{1, \dots, |x|\}
19:
20: return \pi, any published values
```

Fig. B.1: Instantiation of Ψ_1 for ElGamal

```
\mathsf{P}^{\mathsf{S}}_{\Psi_2}((\hat{Z}, hecpar, f, X, C, cpar), (y, r, r_{\hat{Z}}))
                                                                                       P_{\Psi_2} continued
                                                                                      24: publish U \leftarrow \gamma_X + \sum_{i=0}^{|x|+1} \rho_i
 1: parse r_{\hat{Z}} = (r', r_{\mathsf{Enc}})
 2: parse X = (pk_E, C_0, ..., C_{|x|+1})
 3: \mathbf{parse}\ \mathsf{pk}_E = (g,h)
                                                                                                \text{publish } U' \leftarrow \gamma_Y + \gamma_Z + \sum_{i=1}^{|x|+1} \rho_i'
 4: t \leftarrow \mathbb{G}
 5: (\alpha_0, \ldots, \alpha_{|x|+1}, \beta_0, \ldots, \beta_{|x|+1})
                                                                                                for i \in \{1, ..., |x|\}
                                                                                       26:
 6: \rho_0, \ldots, \rho_{|x|+1}, \rho'_0, \ldots, \rho'_{|x|+1}
                                                                                       27:
                                                                                                     publish \mathcal{R}_i \leftarrow \mathsf{Commit}_{t,g}(\rho_i, \delta_i)
 7: \delta_0, \ldots, \delta_{|x|+1}, \delta'_0, \ldots, \delta'_{|x|+1}
                                                                                                     publish \mathcal{R}'_i \leftarrow \mathsf{Commit}_{t,q}(\rho'_i, \delta'_i)
                                                                                      28:
 8: \gamma_X, \gamma_Y, \gamma_Z, \delta_X, \delta_Y, \delta_Z) \leftarrow \$ \mathbb{Z}_q^{4|x|+10}
                                                                                                 publish \mathcal{X} \leftarrow \mathsf{Commit}_{t,q}(\gamma_X, \delta_X)
                                                                                       29:
 9: a \leftarrow \mathsf{lobits}_k(y)
                                                                                                 publish \mathcal{Y} \leftarrow \mathsf{Commit}_{t,q}(\gamma_Y, \delta_Y)
                                                                                      30:
10: m \leftarrow \lfloor \frac{y}{2^k} \rfloor
                                                                                                 \mathsf{publish}\ \mathcal{Z} \leftarrow \mathsf{Commit}_{t,g}(\gamma_Z, \delta_Z)
                                                                                      31:
       publish R \leftarrow \mathsf{Commit}_{q,h}(r')
                                                                                      32 : publish O = \delta_X \prod_{i=0}^{\infty}
         publish A_0 \leftarrow \mathsf{Commit}_{g,h}(1;\alpha_0)
         publish A_0^R = \mathsf{Commit}_{a,h}(r'; \beta_0)
                                                                                      33 : publish O' = \delta_Y \delta_Z \prod^{|x|+1}
        for i \in \{1, \dots, |x| + 1\}
              publish A_i = \mathsf{Commit}_{A_{i-1},h}(a;\alpha_i)
15:
                                                                                      34: \pi \leftarrow \mathsf{PoK}_{\Psi} \Big\{ w = (y, r_{\mathsf{A}}, r',
              publish A_i^R = \mathsf{Commit}_{A_i,h}(r';\beta_i)
                                                                                      35: r_{\mathsf{Enc}}, a, m, and all greek letters):
        for i \in \{1, \dots, |x| + 1\}
                                                                                                     a < 2^k \wedge 1 = q^{y-a} (q^{2^k})^{-m} \wedge
              parse C_i = G_i, H_i
18:
                                                                                      36:
              publish G_i^{AR} \leftarrow \mathsf{Commit}_{G_i,t}(a^i r'; \rho_i)
                                                                                      37:
                                                                                                     C = \mathsf{Commit}_{cpar}(y; r_{\mathsf{A}}) \wedge
19:
              publish H_i^{AR} \leftarrow \mathsf{Commit}_{H_i,t}(a^i r'; \rho_i')
                                                                                                     published commitments correctly computed
20:
                                                                                       38:
         publish X \leftarrow \mathsf{Commit}_{q,t}(r_{\mathsf{Enc}}; \gamma_X)
                                                                                      39:
         publish Y \leftarrow \mathsf{Commit}_{g,t}(y; \gamma_Y)
                                                                                      40: return \pi, any published values
         \mathsf{publish}\ Z \leftarrow \mathsf{Commit}_{h,t}(r_{\mathsf{Enc}}; \gamma_Z)
23:
24:
```

Fig. B.2: Instantiation of Ψ_2 for ElGamal