Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets

Mahimna Kelkar* Cornell Tech

Kushal Babel* Cornell Tech

Philip Daian* Cornell Tech

James Austgen Cornell Tech

Vitalik Buterin Ethereum Foundation

Ari Juels Cornell Tech

ABSTRACT

Most cryptographic protocols model a player's knowledge of secrets in a simple way. Informally, the player knows a secret in the sense that she can directly furnish it as a (private) input to a protocol, e.g., to digitally sign a message.

The growing availability of Trusted Execution Environments (TEEs) and multiparty computation (MPC), however, undermines this model of knowledge. Such tools can *encumber* a secret sk and permit a chosen player to *access* sk *conditionally, without actually* knowing sk. By permitting selective access to sk by an adversary, encumbrance of secrets can enable vote-selling in cryptographic voting schemes, illegal sale of credentials for online services, and erosion of deniability in anonymous messaging systems.

Unfortunately, existing proof-of-knowledge protocols fail to demonstrate that a secret is unencumbered. We therefore introduce and formalize a new notion called *complete knowledge* (CK). A proof (or argument) of CK shows that a prover does not just know a secret, but also has fully unencumbered knowledge, i.e., unrestricted ability to use the secret.

We introduce two practical CK schemes that use special-purpose hardware, specifically TEEs and off-the-shelf mining ASICs. We prove the security of these schemes and explore their practical deployment with a complete, open-source, end-to-end prototype with smart-contract verification that supports both. We show how CK can address encumbrance attacks identified in previous work. Finally, we introduce two new applications enabled by CK that involve proving ownership of blockchain assets.

1 INTRODUCTION

Most cryptographic protocols are designed under a simple model of knowledge. If a player $\mathcal{P}$ can explicitly furnish a secret value sk as a (private) protocol input, then she *knows* sk. In a digital signature scheme, for example, $\mathcal{P}$ inputs a private key sk to a locally executed algorithm to sign a message.

This basic, intuitive model of knowledge, however, can break down when sk is not controlled by a single player, but an *interactive functionality*. For example, sk might be stored exclusively in a trusted execution environment (TEE) such as Intel SGX [50, 51], AMD SEV [9], or AWS Nitro Enclaves [8]. The TEE could then *encumber* $\mathcal{P}$'s access to sk, by only allowing selective use. For instance, in the previous digital signatures example, a TEE could generate (and store) a private signing key sk for a user Alice, but only allow Alice to sign messages approved by an adversary.

Such encumbrances can also be realized by multi-party computation (MPC) [38, 69] over sk among a committee that restricts its use. Encumbrance of secrets can undermine security in many cryptographic protocols, as we will show.

It may seem counterintuitive that a user / prover might *want* to encumber her own secret sk. Encumbrance of secrets, it turns out, can paradoxically *benefit* a user. For example, as highlighted in [31, 57], voters that choose to encumber secret keys used in a voting scheme can sell their votes to an adversary trying to subvert an election. Here, Alice might encumber her voting key sk so that she can only sign a ballot with candidate Bob, the choice of adversary Mallory. Alice can then sell Mallory an enforceable promise that if she votes, she will vote only for Bob. Remarkably, even techniques that specifically aim to prevent such vote-selling—e.g., so-called coercion-resistant voting schemes [28, 29, 43]—fail in the presence of such key encumbrance.

In this paper, we introduce and explore a new notion of knowledge called *complete knowledge* (CK). Complete knowledge embodies a strong notion of possession meant to rule out encumbrance of e.g., the secret key. CK by a prover $\mathcal{P}$ of a secret sk means, informally, that it has *unencumbered* access to sk and *can use it for any desired purpose*, e.g., can sign any message of her choice.

CK can be leveraged in our voting example, by requiring Alice to prove that she has complete knowledge of her secret key sk before allowing her to vote. Here, CK would imply that Alice can always cast any desired vote and therefore, cannot sell Mallory an enforceable promise to vote only for Bob.

Our goals in this work are to formalize CK, implement CK schemes end-to-end, and shed light on its various applications.

The problem with proofs of knowledge. To understand CK, we build on the classical formalism of *proofs of knowledge* (PoKs). PoKs are interactive protocols in which a prover demonstrates knowledge of some kind to a verifier. PoKs play an important role in many cryptographic constructions and have found widespread applications in e-voting [25, 26, 29, 49], encryption [58, 59], group signatures [13], and private cryptocurrency transactions [20, 62].

More formally, a PoK involves two players: a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. The goal is for $\mathcal{P}$ to convince an honest $\mathcal{V}$ that it knows a valid *witness* sk for some (public) statement x. In practice, PoKs are often *zero-knowledge* [35], meaning that the protocol hides any information about sk from $\mathcal{V}$. (A PoK need not be zero-knowledge, though, and this is true of some CK schemes we propose.)

As observed above, however, $\mathcal{P}$ could have access to sk intermediated by an interactive protocol with another entity or device (e.g., a TEE or an MPC committee). In this case, classical PoK formalism breaks down, because it lacks a notion of encumbered knowledge.

In our voting example above, for instance, Alice has a secret key sk encumbered in a TEE that only allows her to cast a vote for Bob in an election. This same TEE, however, might allow unencumbered

*The first three authors contributed equally to this work.

use of sk for completely different purposes, e.g., signing to authorize cryptocurrency transactions. In these contexts, Alice along with the TEE *can successfully prove knowledge* of sk. Yet *neither Alice nor any other entity truly knows* sk, in the sense of being able to use it for *any* desired purpose. In fact, the TEE can allow Alice to take any action using sk *except for the one to vote against Bob*. This allows a powerful collusion between Alice and the bribing adversary Mallory; Mallory can bribe Alice without risk since she provably cannot vote against Bob, while Alice is willing to accept the bribe since Mallory learns no information about the key and at the same time Alice is guaranteed to be able to utilize her key for all other functionalities.

This situation underscores a mismatch between the existing formalism for proofs of knowledge and knowledge in a critical, real-world sense. In this paper, we show the potential practical impact of this mismatch, describing coercive attacks that exploit encumbrance of secret keys in voting schemes, deniable messaging, and blockchain systems.

We show how to remedy the resulting problems by introducing the notion of CK.

Proofs of complete knowledge (PoCKs). In this work, we introduce and formalize proofs of complete knowledge (PoCKs). (Our definitions also cover *arguments* of complete knowledge (ACKs), for which $\mathcal{P}$ is polynomial time. We often informally use the term PoCK to denote both.)

The core idea in our formalization is intuitively as follows: A PoCK scheme ensures complete knowledge if it is the case that when an honest $\mathcal{V}$ accepts a proof by $\mathcal{P}$, *$\mathcal{P}$ can learn her own witness* sk *fully during the proof execution.*

Specifically, in a PoCK, $\mathcal{P}$ must be able to *eavesdrop* on an unencrypted channel carrying sk. To ensure this "self-eavesdropping" capability, $\mathcal{P}$ is given access to a special *resource* $\mathcal{R}$ required for successful execution of the proof.

$\mathcal{R}$ will typically be a local piece of hardware within the trust domain of $\mathcal{P}$. $\mathcal{R}$ may paradoxically itself be a TEE that stores sk—as a way of preventing encumbrance using another TEE. Alternatively, $\mathcal{R}$ could be a resource, such as an ASIC, with special computational capabilities. Informally, an *eavesdropper* $\mathcal{E}$ must be present on the channel between $\mathcal{P}$ and $\mathcal{R}$. Abstractly, $\mathcal{E}$ can be visualized as the *physical manifestation* of a straight-line extractor [37] from PoK literature, in the sense that it will allow for extraction in practice rather than just as a proof construct (see Section 4 for details).

In the case of $\mathcal{R}$ being a TEE, $\mathcal{E}$ can be constructed by simply requiring $\mathcal{P}$ to submit sk in plaintext to $\mathcal{R}$, or alternatively by having a function within the TEE application that reveals sk. Even if $\mathcal{P}$ uses a TEE or MPC committee in an attempt to encumber sk—thus hiding sk from herself—exposure on $\mathcal{E}$ means that $\mathcal{P}$ can still recover sk. Thus a PoCK ensures that sk is unencumbered. The PoCK setting is shown in Figure 1.

Let's return to our voting example now. Intuitively, if a PoCK is used here, then Mallory will no longer have any guarantee on Alice's vote since Alice can use $\mathcal{E}$ to fully recover sk without being detected by Mallory.

Realizing CK. We consider two practical schemes for realizing PoCKs. These schemes enable any proof of knowledge protocol to be converted into a PoCK protocol.

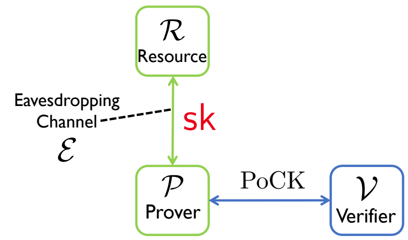

Figure 1: *Proof of complete knowledge* (PoCK) setting. A PoCK is a proof of knowledge, but has two additional requirements: (1) $\mathcal{P}$ can only execute the proof successfully by accessing a special local *resource* $\mathcal{R}$; and (2) $\mathcal{P}$ must have access to the witness sk as it is transmitted to $\mathcal{R}$ over a plaintext *eavesdropping channel* $\mathcal{E}$. Green lines / boxes in the figure indicate entities local to / within the trust domain of $\mathcal{P}$, while blue lines / boxes indicate those outside.

Our first scheme is a conceptually straightforward one that generalizes an idea proposed by Gunn et al. [40] for preventing TEE-based attacks on deniability in messaging protocols. This scheme paradoxically realizes the resource $\mathcal{R}$ as a TEE *that prevents encumbrance* by, e.g., another TEE. The idea is that $\mathcal{P}$ generates sk inside or inputs sk to a special TEE application that outputs sk to the user on demand, thus realizing an eavesdropping channel $\mathcal{E}$. To prove complete knowledge of sk, $\mathcal{P}$ has the special TEE application generate an attestation proving that sk is output to $\mathcal{P}$. We prove the security of this CK scheme under the assumption that it is not practical to run a TEE instance inside another TEE instance.

TEEs have important drawbacks, though. Notably, one widely available, fully-featured hardware TEE is Intel SGX, in which a number of serious vulnerabilities have been discovered (and later patched), e.g., [18, 65, 66].[1]

We thus explore a second PoCK realization involving *proof of work* (PoW) [33, 41], using an ASIC as the resource $\mathcal{R}$.

An ASIC (Application-Specific Integrated Circuit) is special-purpose hardware for a specific computation. ASICs are widely used for cryptocurrency mining [63], specifically hashing (e.g., double SHA-256 for Bitcoin) and thus widely available as an off-the-shelf component for our PoCK scheme. ASICs also have an important feature for our purposes: they accept only *unencrypted* inputs.[2]

Our ASIC-based PoCK requires $\mathcal{P}$ to solve a specially crafted PoW puzzle within a certain time period. Successfully solving this puzzle will reveal sk to $\mathcal{P}$ with high probability. Complete description of our scheme and its security analysis is given in Section 6.

Interestingly, our approach to CK not only contributes to the theory of proof systems, but draws on techniques from the literature on this theory, specifically the *straight-line, non-programming extractor* construction of Fischlin [37].

To show practicality of our CK schemes, we implement an end-to-end system SMACK (Section 7) in which the verifier is a smart contract on the Ethereum blockchain. This implementation both shows

[1] While a break of SGX (or other TEEs) may seem to *help* achieve CK by exposing sk, it can in fact *undermine* CK proofs by enabling the generation of fake TEE attestations. At the same time, a TEE break does not necessarily prevent encumbrance of secrets, as the adversary who has broken a TEE may be distinct from one who is using the TEE to encumber secrets. Also, there are alternative ways to encumber secrets, e.g., MPC.

[2] Even if an ASIC supported encrypted connection, without a built-in enclave, it would still be eavesdroppable.

how minimal a CK verifier can be and supports the blockchain applications we discuss in Section 2. The ASIC-based implementation (Section 7.1) uses an off-the-shelf mining ASIC. In order to make CK widely accessible, we also implemented our TEE-based scheme as a mobile app (Appendix D.1) using the mobile phone's TEE.

Contributions. In brief, our contributions are:

- *Complete Knowledge:* We introduce a new notion of knowledge for cryptographic protocols, called *complete knowledge* (CK), that addresses theoretical and practical limitations of classical PoKs arising from encumbrance of secrets.
- *CK applications:* We revisit known attacks on existing protocols that lack CK and offer a unified treatment and discussion of countermeasures through the lens of CK (Section 2). We also introduce two new applications of CK that help prove ownership of blockchain assets.
- *Formalization:* We formalize CK as a strengthening of standard proofs / arguments of knowledge (Section 3 and 4).
- *Practical CK schemes:* We present two practical schemes to convert proofs of knowledge to proofs of complete knowledge (PoCKs), using a TEE (Section 5) or off-the-shelf mining ASICs (Section 6). Our schemes work for a very broad class of Σ-protocols, a common type of ZK proof of knowledge in practice. *We implement both our CK schemes in a full, end-to-end system in which verification is performed in a smart contract (Section 7).*

2 MOTIVATION: ATTACKS & APPLICATIONS

To motivate our exploration, we begin by briefly reviewing three key-encumbrance attacks from existing literature and explain how CK can serve as a countermeasure. The result is a new, unified treatment of these attacks that identifies their common root cause as a lack of CK for private credentials. We also describe two *new applications* enabled by CK.

2.1 Attacks and CK-Based Countermeasures

Authentication protocols that lack CK are vulnerable to various attacks in which credentials, such as secret keys or passwords, are made available fully or conditionally to an adversary. Three concrete examples are described below.

Deniable Messaging: Deniability in messaging protocols is the (desirable) property that a participant Alice cannot prove the authenticity of a transcript of her communication with another participant Bob, to an outsider, rendering their communications inadmissible as evidence against Bob. Several widely deployed messaging protocols, e.g., Signal [1, 30] (which is based on OTR [17]),advertise such deniability as a key feature. These protocols accomplish deniability by exposing key material to two communicating parties, Alice and Bob, that allows either of them to forge a transcript unilaterally. E.g., Alice can do so using her long-term private key A and an ephemeral private key a.

Gunn et al. [40] show how a TEE can erode the deniable authentication [17, 32] that underpins messaging protocols. The use of a TEE enables a simple attack on deniability by *either* communicating party in isolation. It suffices for Alice, for example, to generate her ephemeral private key a in a special TEE application that does not permit her to forge messages from Bob. Through this technique, Alice can show a judge that she lacks the ability to forge making it so that Bob loses deniability for the messages he sends; notably this happens *without Bob realizing that his deniability has been eroded.*

Gunn et al. describe some countermeasures, the simplest counterintuitively involving use of a small TEE program that attests that a user's private key is present in unprotected memory, i.e., *outside* the enclave, and thus made available to the user for transcript forgery.

Our work, by introducing CK, formalizes and extends ideas which allow the countermeasure from [40] to work.

Electronic Voting: Electronic voting is becoming important in decentralized systems such as permissionless blockchains [44, 49]. Those systems make use of user-generated keys. They are vulnerable to attacks—like those initially described in [31] and subsequently in [57]—in which a voter encumbers her private key used for voting in a TEE upon generation. The voter then offers a briber or coercer *exclusive and verifiable access* to her key either permanently or for certain elections. Access is verifiable because the TEE can present a proof of encumbrance to the adversary. Note that the key can be encumbered for specific well defined tasks and does not for example, compromise the voter's cryptocurrency if the same key also controls her cryptocurrency.[3]

CK offers a countermeasure to such attacks. If a voter is required to prove CK for her private key sk, then she cannot encumber it. While prior literature has proposed *coercion-resistant* (which subsumes receipt-free [14]) voting protocols [29, 43, 48] for the explicit purpose of countering adversarial coercion, we emphasize that they are insecure against TEE-based *encumbrance attacks* [57]. CK is a *necessary* requirement to restore coercion-resistance in these protocols under encumbrance attacks.

Coercion-resistant protocols involve either: (1) An authority that sends keys to voters over an untappable channel, along with the ability of users to present fake versions of these keys to adversaries or (2) The ability for the voter to re-vote. CK *cannot* fully remedy the problem on its own for approach (1), as the TEE can still identify true keys to adversaries. But CK *can* at least prevent an adversary from gaining *exclusive* access to voting keys and *can* restore the coercion-resistance for protcols using approach (2). We leave it as an open problem to understand whether coercion-resistance can be easily restored in the first approach[4].

Similarly, CK offers a path to shoring up security in *minimal anti-collusion infrastructure* (MACI), a scheme proposed by Buterin [22] that is designed to provide bribery-resistance in voting and other applications in a way loosely analogous to coercion-resistance [14, 29, 43, 48].[5] Recent strides have been made toward MACI deployment on Ethereum [42]. As explicitly noted in [22], MACI is vulnerable to TEE-based encumbrance of keys. Use of CK, however, can restore the scheme's bribery-resistance properties.

[3]Because a TEE can be taken offline, encumbrance in a TEE alone only ensures a briber that either the briber will be able to cast a vote or no one will. While limited, this property is still valuable to the adversary. Networks of TEEs, e.g., [6, 31] can in fact further help ensure liveness.

[4]One potential approach is to use CK along with placing all voter functionality in a TEE application. This works in theory but comes at the cost of an application-specific realization and bloated trusted-computing base.

[5]The basic idea is to allow users to switch their registered keys secretly; this prevents an adversary from knowing whether a key presented by a user is valid or not. The scheme allows for a race condition between valid users and adversaries with whom keys are shared. It thus does not strictly meet formal definitions of coercion-resistance, e.g., JCJ [43], although use of deposits acts as a practical disincentive to key sharing.

2.2 New Applications Enabled by CK

We introduce two new applications enabled by CK, i.e., through use of PoCKs. We show how CK enables users to prove facts about asset ownership that they could not easily prove without CK. Specifically, *these applications cannot be securely realized with standard PoKs.*

Key-coupling: In some systems, it is valuable to be able to ascertain that two different private keys, sk_1 and sk_2 (with respective public keys pk_1 and pk_2), are known simultaneously by the same user. This is possible with a CK witness consisting of two private keys. That is, the prover furnishes a CK proof of knowledge of $\mathsf{sk}_1 \parallel \mathsf{sk}_2$, i.e., a concatenation of the two private keys. We refer to such proof as *key-coupling*.

We emphasize that key-coupling is not possible using, e.g., a conventional PoK of $\mathsf{sk}_1 \parallel \mathsf{sk}_2$. Two distinct holders of sk_1 and sk_2 could jointly generate such a proof using secure function evaluation and avoid mutual key disclosure.

Key-coupling has a number of applications, particularly in blockchain systems. They include:

(1) *KYC dilligence:* Cryptocurrency users often transfer control of their own coins from one key to another, e.g., from hot to cold wallets or vice versa. Users often must undergo know-your-customer (KYC) to transact with exchanges. It can be helpful for a user who has undergone KYC diligence with respect to the address associated with pk_1 to be able to transfer assets to an address associated with pk_2 *without having to undergo diligence again.* By proving simultaneous knowledge of sk_1 and sk_2, the user provides strong evidence that assets transferred between pk_1 and pk_2 belong to the same user—or at least fall under the control of a single entity. The same approach can be used by the owner of the address associated with pk_2 to prove that *she doesn't owe tax on funds* sent from pk_1, as the funds didn't change hands.

(2) *Privacy-preserving credential linkage:* Suppose that a user who controls private keys for pk_1 and pk_2 has a public credential attached to pk_2 (e.g., proof of KYC diligence, as above). The user can construct a CK proof for sk_1 *plus* possession of some public key / address (pk_2) with an associated KYC credential. She can do this *without revealing* pk_2.

(3) *Enforcing NFT royalty payments:* Non-fungible tokens (NFTs) are blockchain objects that often represent ownership of digital artistic works. Some NFT platforms enforce *royalty payments* to artists (e.g., 5% of sale price) upon resale of an NFT. However, those platforms also support direct, royalty-free transfer between addresses so as to support transfer between addresses belonging to a single user. As there is no way (prior to our work) to determine single-owner possession of two distinct addresses, users can exploit this royalty-free transfer feature to *bypass* royalty payments. Controversially, for example, a popular marketplace called Sudoswap facilitates royalty-free NFT sales [36]. Key-coupling can, however, enforce *true single-owner possession of addresses in royalty-free transfers*, thereby closing the loophole that deprives NFT creators of ongoing royalty payments.

Note that the ability of key-coupling to distinguish between within-owner and between-owner transfers may not be future-proof: User wallets could eventually support key changes, whether for key rotation or social recovery [19, 21]. This feature could be abused to transfer control between two distinct users while maintaining an appearance of consistent ownership by a single user. Still, the friction against cheating created by key-coupling might still be sufficient to protect, e.g., small-royalty NFT transfers.

CK addresses and Atomic NFTs: Blockchains enable new mechanisms for joint ownership of indivisible digital assets. Such ownership regimes are referred to as *fractionalization.* This is a particularly popular approach to distributing ownership of expensive NFTs among a collection of users.

Fractionalization, however, introduces problems such as price volatility and attractiveness to scammers [64]. As NFTs are essentially financial instruments—and usable not just for digital art, but for real-world assets such as real estate—preventing of fractionalization can also help with know-your-customer (KYC) / anti-money-laundering (AML) compliance, as it ensures that on-chain ownership representation is accurate [67]. Finally, certain types of NFTs, e.g., *soulbound tokens* (SBTs) [68] are intended by design for exclusive ownership; fractionalization would undermine their utility.

Fractionalization may be prevented on chain, i.e., on the blockchain itself, by allowing ownership only from a user address and not a smart contract. But there exists no mechanism (prior to our work) to prevent *off-chain* fractionalization of NFTs by means of secret sharing or TEEs.

CK offers a novel way to prevent fractionalization of any kind through a concept we call *CK addresses.* A CK address is one whose secret key is guaranteed to be unencumbered through the use of a PoCK protocol. Through CK addresses, we can create what we refer to as *Atomic NFTs*—NFTs which are designed to only permit ownership by CK addresses. This ensures that at all times *only one entity controls the NFT.*

3 PRELIMINARIES AND BACKGROUND

Computational model for interactive proofs. An interactive proof system [39] is a pair $(\mathcal{P}, \mathcal{V})$ of Interactive Turing Machines (ITMs) that communicate with each other in rounds. $\mathcal{P}$ and $\mathcal{V}$ may be given auxiliary inputs z_1 and z_2 respectively, along with a common input x. $\mathcal{V}$ outputs a single-bit at the end of the execution. We use $\langle \mathcal{P}(x, z_1), \mathcal{V}(x, z_2) \rangle$ to denote the random variable for $\mathcal{V}$'s output and $\mathrm{VIEW}_{\mathcal{V}}(\mathcal{P}(x, z_1), \mathcal{V}(x, z_2))$ to denote the random variable for $\mathcal{V}$'s view of the execution.

Interactive proofs of knowledge. Consider a language $L \in \mathrm{NP}$ with witness relation R_L, i.e., $x \in L$ iff. $(x, w) \in R_L$ for a *witness* w. Informally, the goal of a proof of knowledge system is to have the verifier output 1 iff $x \in L$ and the prover "knows" some witness w for x. We say that $(\mathcal{P}, \mathcal{V})$ is an interactive PoK system (for R_L) if it is *complete* and a *proof of knowledge.*

(1) *Completeness* means that the honest prover $\mathcal{P}$ can always convince the honest verifier $\mathcal{V}$ when $(x, w) \in R_L$. Concretely, for all $(x, w) \in R_L$, we have $\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle = 1] > 1 - \mathrm{negl}(\lambda)$ where λ is the security parameter.

(2) A *proof of knowledge* is a protocol in which, if a (malicious) prover $\mathcal{P}^*$ convinces $\mathcal{V}$ that $x \in \mathcal{L}$ (i.e., $\mathcal{V}$ outputs 1), then the prover "knows" a witness w for x. This is formalized by requiring the existence of an extractor $\mathcal{E}$ that can extract the witness given the description of $\mathcal{P}^*$. More formally, we require that for all $\mathcal{P}^*$ and

$x, \Pr[w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x) : (x, w) \in R_L] + \mathrm{negl}(\lambda) > \Pr[\langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = 1]$. If this property holds only for a computationally bounded (PPT) adversarial prover, $(\mathcal{P}, \mathcal{V})$ is called an interactive *argument* of knowledge system.

(3) We further say that a proof of knowledge is *zero-knowledge* if, informally, the verifier learns nothing from a proof execution. Specifically, for any PPT verifier $\mathcal{V}'$, there exists a PPT machine $\mathcal{S}$ (called the *simulator*) such that for all $(x, w) \in R_L$, it holds that $\mathrm{VIEW}_{\mathcal{V}'}(\langle \mathcal{P}(x, w), \mathcal{V}'(x) \rangle) \approx \mathcal{S}(x)$, where $\approx$ means computational indistinguishability.

For a more detailed introduction, we refer the reader to [54]. As stated before, in a PoK system, there is no guarantee that the prover actually has unencumbered access to the witness.

Σ-protocols. A popular type of zero-knowledge proofs of knowledge (ZKPoK) used in practice are Σ-protocols. They are a key building block in our ASIC-based construction. A Σ-protocol is a three-move, interactive ZKPoK with the following structure:

(1) $\mathcal{P}$ sends a message a (often called a *commitment*) to $\mathcal{V}$.
(2) $\mathcal{V}$ sends a challenge $c \leftarrow_\$ \{0,1\}^\ell$ to $\mathcal{P}$.
(3) $\mathcal{P}$ sends a response s to $\mathcal{V}$.

For a given x, $\mathcal{V}$ decides whether to accept $\mathcal{P}$'s proof based on the proof transcript, which consists of the triple (a, c, s).

Σ-protocols have a property called *special soundness*, whereby an extractor can efficiently compute a witness w from a pair of accepting transcripts (a, c, s) and (a, c', s') where $c \neq c'$. (They also have a property called *special honest-verifier zero-knowledge*, which means that a simulator given x and c can generate a transcript (a, c, s) distributed like that in a real execution without access to the witness.)

An additional property of many Σ-protocols is *quasi-soundness*, which means that no efficient prover can produce any two transcripts of the form $(a, c, s), (a, c, s')$, where $s \neq s'$, i.e., two different responses for the same commitment-challenge pair. Our construction applies only to quasi-sound Σ-protocols.[6] We assume such Σ-protocols throughout.

The extractor $\mathcal{E}$ for a proof-of-knowledge protocol may in general *rewind* the prover $\mathcal{P}$. In a Σ-protocol, the special-soundness property gives rise to a simple extractor construction. After the first move in the protocol, i.e., the commitment a by $\mathcal{P}$, $\mathcal{E}$ issues a challenge c, and obtains response s. $\mathcal{E}$ then rewinds $\mathcal{P}$ to the point just after the first move and issues a second challenge c', recovering a second response s'. The two transcripts allow extraction of w.

The best-known Σ-protocol—and a practical choice for proving knowledge of a discrete-log-based public key in ASIC-ZKPoCK—is the Schnorr protocol [60], which proves knowledge of a discrete log. Here, $x = g^w$ (where g is a published generator of some suitable group $\mathbb{G}$), and the goal is to prove knowledge of the exponent w.

Straight-line extraction. The need to rewind the prover $\mathcal{P}$ results in loose security reductions for various signature schemes, e.g., [56] and is typically incompatible with concurrent protocol composition [23]. These issues with rewinding motivated the exploration of *straight-line extractors* (a.k.a. *online extractors*), which do not require rewinding.

[6] Given the ability to generate two transcripts (a, c, s) and (a, c, s') for $s \neq s'$, for instance, which is permissible in strongly-sound Σ-protocols [46], a prover can cheat in our protocol.

Online extractors may observe calls to a hash function H by $\mathcal{P}$, where H is modeled as a *random oracle* (RO), i.e., responses are independent and uniformly random. Of particular interest in our setting are *non-programming* extractors, which involve a strong model where the extractor cannot program the RO, i.e., determine its responses during extraction.

Fischlin [37] proposed a non-interactive ZKPoK scheme with a straight-line extractor whose techniques we adapt to our ASIC construction. In this construction the extractor has access to RO queries. The key idea is that $\mathcal{P}$ uses Σ-protocol transcripts as inputs to the RO in a proof of work (PoW). The scheme is parameterized in such a way that solving the PoW requires with high probability that $\mathcal{P}$ feeds a pair of valid Σ-protocol transcripts $(a, c, s), (a, c', s')$ with $c \neq c'$ to the RO. By observing these transcripts as RO inputs, an extractor $\mathcal{E}$ can extract the witness w.

To provide more detail, Fischlin's scheme builds on a Σ-protocol involving a prover / verifier pair $(\mathcal{P}_\Sigma, \mathcal{V}_\Sigma)$. Parameters include challenge domain cardinality k, proof-of-work difficulty b, global proof-of-work target S, and number of rounds of execution n. Specifically, $H : \{0,1\}^* \rightarrow \{0,1\}^b$, and a valid PoW solution is such that $H(z) = 0^b$ for an input z that includes a valid Σ-transcript (a, c, s). The protocol is as follows:

- Prover $\mathcal{P}$: On input (x, w), $\mathcal{P}$ does the following:
  - Runs the first step of n executions of $\mathcal{P}_\Sigma$ to obtain n commitments $\vec{\mathrm{com}} = (a_1, a_2, \ldots, a_n)$. Let $q_i = ([x, i]; \vec{\mathrm{com}})$.
  - For each given a_i, $\mathcal{P}$ completes an execution of $\mathcal{P}_\Sigma$ on challenges $c_j \in [0, k-1]$. Each execution of $\mathcal{P}_\Sigma$ yields a corresponding response s_j. $\mathcal{P}$ sets $\pi_i = (a_i, c_j, s_j)$ for $H((a_i, c_j, s_j) \parallel q_i) = 0^b$ if one exists; otherwise it sets $\pi_i = (a_i, c_j, s_j)$ for the minimal result, i.e., j for $\mathrm{argmin}_j\, H((a_i, c_j, s_j) \parallel q_i)$.
  - Sends $\vec{\mathrm{com}}, x, \vec{\pi} = \{\pi_i\}_{i=1}^n$ to the verifier.
- Verifier $\mathcal{V}$: On input $(x, \vec{\pi})$, $\mathcal{V}$ checks: (1) For all $i \in [1, n]$ that $\mathcal{V}_\Sigma$ accepts π_i; and (2) $\sum_{i=1}^n H(\pi_i \parallel q_i) < S$.

In order to compute PoW results that $\mathcal{V}$ will accept, $\mathcal{P}$ must with high probability hash *at least two* Σ transcripts of the form (a_i, c_j, s_j) for some a_i. That means that the random oracle H will be called on a pair of inputs that include distinct pairs (a_i, c_j, s_j) and (a_i, c'_j, s'_j). From special-soundness, it follows that w can be extracted.

As will be seen later, while our ASIC ACK protocol draws on the technique of combining a Σ-protocol with a PoW, our setting differs considerably. In straight-line extractors, extraction is a theoretical capability used to prove knowledge of w. In our ASIC ACK, extraction is a practical capability used to show that $\mathcal{P}$ can access w. Additionally, in our ASIC ACK, rather than using a *cryptographic* resource in the form of an RO, $\mathcal{P}$ uses a *computational* resource in the form of an ASIC. The result is a protocol that differs somewhat from Fischlin's in terms of the form of oracle queries involved and the resulting security analysis.

Trusted Execution Environments (TEEs). A TEE runs applications with strong confidentiality and integrity protections. Some TEE platforms can issue a type of statement, known as an *attestation*, to untampered execution of a particular application, along with application outputs.

One popular TEE with attestation capabilities is Intel Software Guard eXtensions (SGX) [10, 50, 51]. Trust in the hardware—and in Intel, which authenticates attestation keys—means that only an SGX platform can generate a valid attestation, i.e., attestations are existentially unforgeable. We make use of formalism for SGX-like TEEs in the universal composability (UC) framework from [53]. TEEs are nearly universal today in mobile devices as well, but without built-in attestation capabilities. Google and Apple, however, generate attestations for devices in their ecosystems [11, 12].

We make use of both SGX (Section 5) and mobile-device TEEs (Appendix D.1) in different CK variants.

4 PROOFS OF COMPLETE KNOWLEDGE

As we have explained, the standard proof-of-knowledge property does not guarantee that the prover actually has unencumbered access to the witness w. This is because in order to recover w, the (knowledge) extractor gets oracle access to the *entire* prover, including parts that may be controlled by separate entities or even the adversary. The issue is that the prover is modeled as a single machine $\mathcal{P}$ even if in reality it is not owned by a single entity. For instance, if w is secret-shared between two independent parties, $\mathcal{P}$ will correspond not to the individual machines, but to the combined system with both parties, and the extractor is given access to this system. Here, even though the system as a whole knows the witness and the extractor is able to recover it, intuitively, it is clear the neither party alone really has unencumbered access to the witness.

We now formalize *proofs of complete knowledge* (PoCKs), which ensure that a single party has full access to the witness.

4.1 Building Intuition

The basic idea behind PoCKs is to design protocols where the knowledge extractor $\mathcal{E}$ is actually runnable in practice, rather than simply a proof construct. We therefore begin with a careful analysis of what can make an extractor fail if practically run, and then use these insights to guide our PoCK formalism. As a first point, since we want $\mathcal{E}$ to be able to run in practice, it is obvious that $\mathcal{E}$ should not rewind $\mathcal{P}$, i.e., it must be *straight-line*. We will further restrict our attention to straight-line (i.e., non-rewinding) extractors that are also *non-programmable* (i.e., unable to program e.g., the random oracle (RO); this is important since real hash functions used to instantiate the RO cannot be programmed). This style of knowledge extractor has been used previously in e.g., [37, 46, 52].

Recall that for standard PoK extraction in the random oracle model, the extractor $\mathcal{E}$ is given two quantities: (1) A transcript of the interaction between the prover and the verifier; and (2) A list of queries (and corresponding responses) made by the prover to the random oracle. What does it mean exactly for $\mathcal{E}$ to be given these inputs in practice? Unfortunately, we find that both of these inputs are problematic to assume in practice (due to encumbrance by MPC or a TEE) which makes designing protocols for our setting particularly challenging.

The first input seems obtainable—any entity present on the communication channel between the prover and the verifier can observe this transcript (even if the prover is composed of multiple entities and only one communicates with the verifier). This however implicitly assumes that the communication is not encrypted and can therefore be observed by $\mathcal{E}$. Such an assumption fails if $\mathcal{V}$ is also run inside a TEE, in which case, a prover TEE holding the witness w would be able to convince $\mathcal{V}$ without w being known in plaintext to any non-TEE entity. This observation has a surprising consequence: the trivial PoK of simply sending w to $\mathcal{V}$ *cannot* be a PoCK unless it can be enforced that $\mathcal{V}$ is not run in a TEE .

The second input—the list of random oracle queries—is also challenging to enforce in light of TEE or MPC encumbrance. This is because a hash function, which will typically be used to instantiate the random oracle, can be computed easily in trusted hardware or MPC. In turn, providing the oracle queries to $\mathcal{E}$ in practice would effectively translate to breaking the trusted hardware or unravelling the MPC protocol to figure out the hash function inputs. Intuitively, the gap here arises from the fact the *physical* instantiation of the random oracle may not comply with the way the extractor functions in theory for the proof of knowledge to go through.

Key technique. To surmount these challenges, we must ensure that the extractor always obtains the inputs required for knowledge extraction in practice. To do so, we consider a physical *resource oracle* functionality $\mathcal{R}$ such that the prover's interaction with $\mathcal{R}$ leaks a witness to $\mathcal{E}$. We seek to deploy $\mathcal{R}$ so that no prover can perform a successful CK proof without $\mathcal{R}$; in a sense, the resource abstraction separates out the part of the protocol responsible for the proof of knowledge.

Concretely, we model $\mathcal{E}$ as a man-in-the-middle entity for $\mathcal{R}$ that can snoop on the queries made to $\mathcal{R}$. $\mathcal{E}$ will use these queries to extract the witness, thereby giving it (and whoever can see its output) *complete knowledge* of the witness (see Remark 1). To emphasize its physical presence and its non-rewinding nature, we refer to $\mathcal{E}$ as the *eavesdropping extractor*, or simply the eavesdropper.

Looking ahead, in our protocols, $\mathcal{R}$ models special hardware available to $\mathcal{P}$—either a global SGX functionality that attests to seeing the witness, or an untrusted ASIC whose computational speed is superior to that of a trusted environment that can potentially conceal the witness from $\mathcal{E}$. In a sense, the SGX instance represents a *physical manifestation* of a trusted third party that furnishes the witness w directly to $\mathcal{E}$; the ASIC similarly may be thought of as a *physical manifestation* of an RO accessible by $\mathcal{E}$.

Here, $\mathcal{E}$ can be thought of as the machine where $\mathcal{R}$ physically resides; as an example, for the SGX resource, $\mathcal{E}$ represents the host machine of the SGX which may simply be one of the entities within $\mathcal{P}$ or even a different external machine. A crucial point here is that by design, we use a resource $\mathcal{R}$ such that there is no practical resource $\mathcal{R}'$ which provides an identical functionality and allows for encrypted queries to be made directly to $\mathcal{R}'$: this enables $\mathcal{E}$ to view the plaintext queries made to the resource.

Remark 1 (Complete knowledge for *some* entity). Notice that PoCK only guarantees that the eavesdropper $\mathcal{E}$ can extract w. If for some reason, the prover does not have access to the output of $\mathcal{E}$ in practice, then it may not be able to recover w.

As a consequence, PoCK protocols can only guarantee that *some* entity (specifically $\mathcal{E}$ and anyone who can see its output) has complete knowledge of w. For instance, if $\mathcal{R}$ is connected to a different outsourced machine $\mathcal{M}$ instead of $\mathcal{P}$, then $\mathcal{E}$ will correspond to $\mathcal{M}$ and be able to recover w while $\mathcal{P}$ might not. We note that this subtlety is not accounted for in standard PoK formalism since $\mathcal{E}$ does not represent a physical entity.

This property, however, is sufficient to deter the collusion and bribery attacks which motivate our work. For instance, a user will not willingly accept a bribe for her vote if it reveals her key (which also controls her money) to another entity.

4.2 Formal PoCK Security

We now formally define proofs-of-complete knowledge. We use λ throughout to denote the security parameter.

Basic setting. Similar to standard PoKs, we consider a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ (modeled as ITMs). Specific to our setting, we model a *resource oracle* $\mathcal{R}$ that can be queried by $\mathcal{P}$. We will often work in a *timed* setting where $\mathcal{P}$ must complete its proof within some time $T(\lambda)$. To model the concrete computational speed of a resource $\mathcal{R}$, we associate with it a function $t_{\mathcal{R}}(\cdot)$ that defines the time taken by the resource to compute responses to its queries. Note that this execution by $\mathcal{R}$ can be concretely faster than by $\mathcal{P}$.

Resource formalism. Abstractly, a resource $\mathcal{R}$ is a randomized and stateful functionality $\mathcal{F}_{\mathcal{R}}$. $\mathcal{R}$ is initialized with an internal state $\mathsf{st}_{\mathsf{initial}} \leftarrow\$ \mathcal{R}.\mathsf{Setup}(1^\lambda)$. Upon input inp from $\mathcal{P}$, $\mathcal{R}$ computes $\mathcal{F}_{\mathcal{R}}(\mathsf{st}, \mathsf{inp}) \rightarrow (\mathsf{st}', \mathsf{out})$ where st is the current state of $\mathcal{R}$, st' is the state after the computation, and out is the output returned to prover. We also model the time taken by $\mathcal{R}$ as the randomized function $t_{\mathcal{R}}(\mathsf{st}, \mathsf{inp})$. Note that $t_{\mathcal{R}}(\cdot)$ may be smaller than if the computation was done by $\mathcal{P}$ itself. The tuple $(\mathcal{R}.\mathsf{Setup}, \mathcal{F}_{\mathcal{R}}, t_{\mathcal{R}})$ represents $\mathcal{R}$.

We parameterize our PoCK protocols by a set $\mathfrak{R}$ of "honest" resources, i.e., resources that an honest prover can utilize to convince an honest verifier. We use $\mathfrak{R}_{\mathsf{all}} \supseteq \mathfrak{R}$ to denote the global set of all *practically instantiable* resources—only resources in this set are *assumed* to exist in practice. This explicitly enables modeling our assumption regarding resource practicality (e.g., the non-existence of SGX inside SGX).

PoCK formalism. A T-timed PoCK (when T is unspecified, there are no additional timing constraints) for language $L \in \mathrm{NP}$ with witness relation R_L, a set $\mathfrak{R}$ of honest resources, and a set $\mathfrak{R}_{\mathsf{all}}$ of practical resources is a tuple $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where:

- $\mathsf{Setup}(1^\lambda) \rightarrow pp$ generates public parameters.
- $(\mathcal{P}, \mathcal{V})$ is an interactive proof system where $\mathcal{P}$ is given (pp, x, w) and $\mathcal{V}$ is given (pp, x). $\mathcal{V}$ outputs a single bit indicating whether the prover has complete knowledge of a valid witness for x. For non-interactive proofs, $\mathcal{P}$ will output a proof π which will be given to $\mathcal{V}$ to verify. For T-timed protocols, $\mathcal{P}$ will be required to run in time T.

We can also consider the standard relaxation of *arguments* (instead of proofs) of knowledge for which only PPT provers are considered (although *argument* and *proof* are often used interchangeably in the literature). Our concrete protocols will be "Arguments of Complete Knowledge" (ACK).

Now, for PoCK security, we define in the subsequent paragraphs, two properties—completeness, and forced-revelation (or CK-soundness) that are required to hold.

Completeness. The first PoCK property of *completeness* mirrors the analogous property for PoKs. Recall that completeness states that an honest prover who holds the witness can convince the verifier to output the success bit. The only difference now for PoCK completeness is that the prover is endowed with a resource oracle $\mathcal{R}$. Formally, PoCK completeness states that for all pp, $\mathcal{R} \in \mathfrak{R}$, and $(x, w) \in R_L$:

$$\Pr[\langle \mathcal{P}^{\mathcal{R}}(pp, x, w), \mathcal{V}(pp, x)\rangle = 1] > 1 - \mathrm{negl}(\lambda).$$

Furthermore, $\mathcal{P}^{\mathcal{R}}$ runs in time at most $T(\lambda)$.

Forced revelation or CK-soundness. The second PoCK property of *forced revelation* is similar in spirit to the knowledge-soundness property of standard PoKs. Abstractly, if a prover is able to convince an honest verifier using any practical resource (i.e., within $\mathfrak{R}_{\mathsf{all}}$ and not just within $\mathfrak{R}$), then the eavesdropper will be able to output a valid witness. Let $\mathsf{out}_{\mathcal{E}}$ denote the output of $\mathcal{E}$. A party able to view the output of the eavesdropper obtains the full witness. Formally, for all pp, inputs x, provers $\mathcal{A}$, and resources $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$ such that $\mathcal{A}^{\mathcal{R}'}$ runs in time $T(\lambda)$,

$$\Pr[\langle \mathcal{A}^{\mathcal{R}'}(pp, x), \mathcal{V}(pp, x)\rangle = 1] < \Pr[x \in L \ \wedge \ (x, w = \mathsf{out}_{\mathcal{E}}) \in R_L] + \mathrm{negl}(\lambda).$$

Forced revelation directly implies a couple of nice properties. First, it implies the usual soundness notion since no prover can convince $\mathcal{V}$ of an $x \notin L$ (except with negligible probability). Second, it also implies that if the prover does not make use of any resource, making it so that $\mathcal{E}$ cannot eavesdrop, then it cannot convince $\mathcal{V}$ except with negligible probability *even if it has the witness*. Intuitively, this property is necessary because otherwise it would imply the ability to prove CK through e.g., a 2PC protocol where the witness is encumbered. As illustrated in the remark that follows, forced revelation also has surprising ramifications, which underscore the nuances of working in our PoCK setting.

Forced revelation implies an interesting separation between trivial protocols for PoK and PoCK (see App. A for details).

4.3 Zero-Knowledge PoCK

Most applications require the prover's witness to not be leaked to the verifier. A strong property often considered in the PoK realm is that of *zero-knowledge* (ZK) [39]. Informally, this ensures that *no additional information* is leaked to the verifier. We will add a similar requirement to PoCKs to formalize "Zero-Knowledge Proofs of Complete Knowledge" or ZKPoCKs.

Zero-Knowledge property for PoCKs. Formally, we adapt the ZK property to our setting as follows: For any PPT verifier $\mathcal{V}'$, there exists a PPT machine $\mathcal{S}$ (called the simulator) such that for all $\mathcal{R} \in \mathfrak{R}$, $(x, w) \in R_L$ and auxiliary input $z \in \{0, 1\}^*$, it holds that:

$$\mathrm{VIEW}_{\mathcal{V}'}(\mathcal{P}^{\mathcal{R}}(pp, x, w), \mathcal{V}'(pp, x, z)) \approx \mathcal{S}^{\mathcal{R}}(pp, x, z).$$

4.4 Eavesdropper Undetectability

The PoCK formalism models $\mathcal{E}$ as a man-in-the-middle entity which eavesdrops on queries made to $\mathcal{R}$. We implicitly assume that $\mathcal{E}$ is *always run*; in other words, we do not model a scenario where w was not extracted even though it *could have been*. While this distinction is not important in the standard cryptographic context, and therefore not part of our core PoCK formalism, it uncovers subtleties in the context of side channels and incentive compatibility. We formalize this as the property of "eavesdopper undectability" and remark on its utility in Appendix A.1.

SGX-PoCK Protocol for $\mathcal{R} = \mathcal{G}_{\mathrm{SGX}}$

Setup(1^λ): Output mpk $\leftarrow \mathcal{R}$.getpk()

$\mathcal{P}^{\mathcal{R}}((sid, \mathrm{mpk}), x, w)$:

$eid \leftarrow \mathcal{R}$.install$(sid, \mathrm{prog}_{\mathrm{CK}})$

$(\mathrm{out}, \sigma) \leftarrow \mathcal{R}$.resume$(eid, (\text{"expose"}, x, w))$

Send (eid, σ) to $\mathcal{V}$

$\mathcal{V}((sid, \mathrm{mpk}), x)$:

Await (eid, σ) from $\mathcal{P}$

$m \leftarrow ((sid, eid), \mathrm{prog}_{\mathrm{CK}}, (\text{"exposed"}, x))$

Output $b \leftarrow \mathrm{S.ver}_{\mathrm{mpk}}(m, \sigma)$

$\mathrm{prog}_{\mathrm{CK}}$

On input("expose", x, w):

Assert $(x, w) \in R_L$

Return ("exposed", x)

Figure 2: SGX-PoCK **Protocol Description.**

5 SGX-BASED POCK PROTOCOL

We now describe SGX-PoCK, a simple but illustrative PoCK protocol which uses an SGX TEE as its resource. Intuitively, the SGX models a physical manifestation of a trusted third party to whom the prover will submit the witness. We use SGX for concreteness but note that a similar PoCK can be realized through any TEE which admits remote attestation, including those in mobile devices, as discussed in Appendix D.1.

SGX resource. We use the formalism for TEEs with attested execution from Pass et al. [53]. Abstractly, SGX attestation is modeled using a global functionality (i.e., with global setup [24]) $\mathcal{G}_{\mathrm{SGX}}$. $\mathcal{G}_{\mathrm{SGX}}$ models all valid SGX processors and is initialized with a master key pair (mpk, msk) with signature scheme S = (S.kg, S.sign, S.ver); this intuitively allows the modeling of anonymous attestation which prevents identifying the SGX which signed an attestation.

$\mathcal{G}_{\mathrm{SGX}}$ permits SGX-equipped parties (denoted by the set Reg) to install programs on their SGX and compute outputs. When a party X provides input inp to an installed program prog, $\mathcal{G}_{\mathrm{SGX}}$ computes its output out and a signature σ on (id, prog, out) where id denotes any relevant session identification information. The tuple (out, σ) is then sent to X as the attested output. For completeness, we detail the full $\mathcal{G}_{\mathrm{SGX}}$ functionality in Fig. 8 in Appendix B.

SGX-PoCK **description.** We describe the full SGX-PoCK protocol in Fig. 2. The public parameter pp output by Setup is just the SGX public key mpk. For the proof protocol, the prover $\mathcal{P}$ first installs the program $\mathrm{prog}_{\mathrm{CK}}$ through $\mathcal{G}_{\mathrm{SGX}}$. Now, given (x, w) in the relation R_L as input, $\mathcal{P}$ submits the tuple ("expose", x, w) to $\mathcal{G}_{\mathrm{SGX}}$ and gets back a signature σ on (id, $\mathrm{prog}_{\mathrm{CK}}$, ("exposed", x)) which it forwards to $\mathcal{V}$. By checking the validity of the signature, $\mathcal{V}$ can convince itself of complete knowledge of a witness corresponding to x.

5.1 SGX-PoCK Properties

It is easy to see that an honest $\mathcal{P}$ given w can always convince $\mathcal{V}$; in other words, completeness holds for SGX-PoCK. For CK-soundness, we require an assumption on the infeasibility of specific types of resources, as we describe below:

Resource assumptions. To ensure that the witness w exposed to the SGX can be eavesdropped upon, intuitively we need to assume that it is not practical to run $\mathrm{prog}_{\mathrm{CK}}$ in an SGX within another SGX. This is because otherwise, the outer SGX could be in possession of w which it could expose to the inner SGX to obtain a CK proof without w ever being accessible outside of a trusted enclave. We briefly remark on how this assumption can be removed in Appendix B.

In the context of the $\mathcal{G}_{\mathrm{SGX}}$ formalism, this means that no program prog installed by a party can install its own program prog′—this is implicitly assumed within [53] since only a fixed registration set Reg is considered for SGX devices.

CK-soundness proof. Now, assuming that there is no practical resource $\mathcal{R}' \subseteq \mathfrak{R}_{\mathrm{all}}$ that models such a 2-layer SGX, it is easy to show that SGX-PoCK satisfies CK-soundness. We provide a sketch below and defer the full game-based proof to Appendix B.1.

Suppose that some $\mathcal{P}'$ is able to convince the honest $\mathcal{V}$ that it knows the witness to a statement x. This can happen in only one of two ways: (1) $\mathcal{P}'$ submits (x, w) to the honest resource $\mathcal{R} = \mathcal{G}_{\mathrm{SGX}}$ as one of its queries; (2) $\mathcal{P}'$ does not query $\mathcal{G}_{\mathrm{SGX}}$ with (x, w)—it either does not use $\mathcal{G}_{\mathrm{SGX}}$ at all (potentially uses a different $\mathcal{R}' \in \mathfrak{R}_{\mathrm{all}}$) or queries it with different values. In the first case, w will be sent in plaintext to $\mathcal{G}_{\mathrm{SGX}}$ allowing $\mathcal{E}$ to easily output it (since we assume that $\mathcal{G}_{\mathrm{SGX}}$ cannot be simulated inside another TEE). The second case implies that $\mathcal{P}'$ was able to forge a valid $\mathcal{G}_{\mathrm{SGX}}$ signature on (x, w) using either (i) a different resource without the msk of $\mathcal{G}_{\mathrm{SGX}}$ or (ii) valid signatures from $\mathcal{G}_{\mathrm{SGX}}$ on different $x' \neq x$. Both of these contradict the SUF-CMA security of the signature scheme S used and therefore only arise with $\mathrm{negl}(\lambda)$ probability.

Privacy properties. Observe that SGX-PoCK as described is not zero-knowledge since the attestation can be forwarded. Still, the SUF-CMA security of S implies non-trivial privacy properties over the basic PoCK. In particular, given many SGX-PoCK proofs (which are just signatures under the master key), an adversary still cannot forge a different SGX-PoCK proof for another statement x. Appendix B remarks on modifications to make SGX-PoCK satisfy ZK.

6 ASIC-BASED POCK CONSTRUCTION

In this section, we explore the design of a (ZK)PoCK using a cryptocurrency mining ASIC as the prover resource $\mathcal{R}$—a protocol we call ASIC-ZKPoCK. Our construction is quite general. It can transform a broad class of Σ-protocols [61]—a common class of three-move, honest-verifier ZKPoK—into an (honest-verifier) ZKPoCK through the use of an ASIC. The only requirement is that the Σ-protocol be quasi-sound.

Intuition. ASIC-ZKPoCK makes use of the *performance gap* between computation in secure environments (e.g., SGX) or secure multi-party computation (MPC) and computation using fast ASIC hardware. By running as a time-constrained protocol, ASIC-ZKPoCK ensures that it is only feasible to compute a correct, timely proof using a mining ASIC.

As required, mining ASIC hardware has an eavesdropping channel $\mathcal{E}$. (Mining ASICs, as we explain below, don't support encryption, so eavesdropping is straightforward.) This channel $\mathcal{E}$ allows the prover to extract the witness during the proof generation process, ensuring complete knowledge.

In short, a mining ASIC may be viewed as a computing resource $\mathcal{R}$ that is special in that it is fast—faster than a CPU—and has an eavesdropping channel $\mathcal{E}$ on inputs. A mining ASIC thus fits our basic CK framework shown in Fig. 1.

To additionally show practicality, we show how ASIC-ZKPoCK can be parameterized to work with cheap outmoded ASICs (which are no longer practical for e.g., Bitcoin mining).

Why ASIC-based (ZK)PoCKs? There are two reasons, security and performance related respectively, for exploring ASIC-based PoCKs over TEE-based PoCKs. First, many TEE-based machines operate in the cloud. As noted in Section 1, TEE vulnerabilities could expose the private keys to cloud operators or remote adversaries with access to the cloud. Second, in the case of blockchain applications, TEE attestations can be expensive to verify. For example, EPID [3] attestations—an attestation type generated by Intel SGX without special provisioning and with optional privacy protection—is expensive to verify in the Ethereum Virtual Machine (EVM).

6.1 Background: Mining ASICs and PoWs

Cryptocurrencies like Bitcoin use *proof of work* (PoW) [41] for the safety of their underlying consensus mechanism for block generation, or *mining*. PoW involves solving puzzles (see below) by means of repeated cryptographic hashing.

A mining ASIC, designed for fast PoW puzzle solving can compute hashes 1,000,000× faster than a CPU—and thus achieves a performance gap compared to any SGX-protected application. Mining ASICs today take only *unencrypted* inputs, meaning that their inputs are exposed to users.[7]

A PoW puzzle is based on a particular hash function $\mathcal{H}$ (typically modeled as a random oracle) with ℓ-bit outputs, where ℓ is a security parameter. A puzzle instance has a *difficulty* d corresponding to the probability of correctly solving it with a single hash computation.[8] A puzzle instance may also include ancillary data B (e.g., the block header data for Bitcoin). Solving a puzzle with data B and difficulty $d \in [1, 2^\ell)$ involves finding a *nonce* v such that $\mathcal{H}(B \parallel v) < 2^\ell/d$. The probability of solving the puzzle for any random nonce is an independent and identically distributed Bernoulli random variable with success probability $1/d$ (for $d \mid 2^\ell$).

ASIC-ZKPoCK **intuition.** The idea behind ASIC-ZKPoCK is to require $\mathcal{P}$ to use an ASIC to find a solution π to a PoW puzzle. π is required to include a proof of knowledge of the witness w. Specifically, $\mathcal{P}$ specifies PoK commitment a and π includes the challenge c and response s for a Σ-protocol transcript (a, c, s) involving w. Since Σ-protocols are zero-knowledge, the transcript (and thus π) will not expose w.

Despite this, the process of *computing* π will involve $\mathcal{P}$ sending *multiple* Σ-protocol transcripts to the ASIC. Consequently, given the special soundness (and quasi-soundness) of the Σ-protocols we use, the eavesdropper $\mathcal{E}$ will be able to *extract* w *with high probability.*

While it is possible in principle to perform mining for an ASIC-ZKPoCK in an SGX enclave (in a CPU), this is not realistic, as ASICs are far more performant than CPUs. As an example, a top-of-the-line ASIC can outperform a (single-server) SGX application by a factor of more than 1,000,000.

[7] Even if such ASICs were ultimately to support encryption, if they do not also support enclaves, any value input to an ASIC will still be exposed to the user. While one could imagine reasons to support encryption in mining ASICs, there's no compelling reason to support enclaves.

[8] We refer to d generically in our protocol description as a difficulty parameter, without reference to the specific notion of "difficulty" in Bitcoin.

6.2 ASIC-Based PoCK: Protocol construction

Formally, in ASIC-ZKPoCK, $\mathcal{P}$ and $\mathcal{V}$ execute a Σ-protocol. $\mathcal{P}$ embeds a valid proof transcript (a, c, s) for the Σ-protocol in a PoW puzzle whose solution π constitutes a full ASIC-ZKPoCK proof. The key idea in our construction is to *require* $\mathcal{P}$ *to try out multiple puzzles, each with a different challenge c (and thus response s), in order to find a solution.* We accomplish this by carefully choosing parameters such that with high (but still constant) probability, a single randomly chosen c will not lead to a puzzle with a valid solution. As a result, $\mathcal{P}$ must *input different transcripts to the ASIC* (or in other words, create different puzzles for the ASIC), among which, by quasi-soundness, is a pair $(a, c, s), (a, c', s')$ with $c \neq c'$ and $s \neq s'$. From this pair, given the special soundness property of the Σ-protocol, $\mathcal{E}$ can extract w.

At the same time, however, π itself—the solution revealed to $\mathcal{V}$—contains *only one proof transcript.* Thus $\mathcal{V}$ does not learn w ensuring that the protocol remains zero-knowledge.

As $\mathcal{P}$ must complete the proof in a limited period, it can succeed only with a powerful resource. Given the right parameter choices, this means that $\mathcal{P}$ must employ an ASIC. We will consider the set $\mathfrak{R}$ of (honest) resources that work to be all PoW ASICs with hash rate at least some threshold Q_{asic}.

Preliminaries. ASIC-ZKPoCK involves a random PoCK *challenge* r from $\mathcal{V}$. Let $\pi = (B, v)$ denote a puzzle solution computed by $\mathcal{P}$ in response to a PoCK challenge r. The puzzle solution π consists of a block header B and nonce v corresponding to a valid proof of work. Let $\Sigma\mathsf{map}_r(\pi) \rightarrow (c, s)$ denote a function, dependent on r (as specified below), that maps π to a pair (c, s). We define two verification functions:

- $\mathsf{PoKAccept}(x, (a, c, s)) \rightarrow \{\mathsf{true}, \mathsf{false}\}$ checks the correctness of the Σ-protocol transcript (a, c, s) w.r.t. public PoK value x.
- $\mathsf{puzAccept}[d, \beta](\pi) \rightarrow \{\mathsf{true}, \mathsf{false}\}$ checks that (B, v) represents a correct puzzle solution with difficulty d, i.e., $\mathcal{H}(B, v) < 2^\ell/d$. puzAccept also checks that nonce v is of correct size ($v < \beta$), for a parameter β discussed below.

Our ASIC-ZKPoCK protocol involves n rounds of the above form where a challenge r is sent by the verifier and a solution π must be computed by the prover in the round time τ. For the verifier to accept the proof, the prover must compute the puzzle solution in more than y rounds. Let $\mathsf{puzAccept}^y[d, \beta](\pi_1, \ldots, \pi_n)$ denote whether more than y (out of n) puzzle solutions are valid. Looking ahead, setting y appropriately will ensure that the proof can be completed by an honest ASIC but not by an adversarial prover.

Efficient puzzle-solving vs. transcript extraction. As explained above, our protocol design must force $\mathcal{P}$ to try different values of c while computing a puzzle solution π. Changing c, however, carries the overhead of computing a new corresponding block header B and feeding B to the ASIC. While this is not relevant for a "theoretical ASIC," it poses a significant practical challenge. Therefore, to obtain a protocol which can be deployed, we don't want $\mathcal{P}$ to have to change c *too* frequently. This requires us to correctly set the nonce size (in particular $v < \beta$) as discussed below.

Our approach to resolving this tension is to: (1) map B to a distinct (c, s), so that changing B changes (c, s) *but* (2) allow $\mathcal{P}$ to explore a range of different nonces v for a given B.

ASIC-ZKPoCK **Protocol**

$\mathcal{P}$		$\mathcal{V}$
	$\xrightarrow{a_i}$	// Generate challenge $r_i \leftarrow\$ \{0,1\}^{\mathrm{poly}(\lambda)}$ $t \leftarrow$ time
	$\xleftarrow{r_i}$	

	$\xrightarrow{\pi_i}$	$t' \leftarrow$ time
		// Verify Σ transcript
		$(c_i, s_i) \leftarrow \Sigma\mathsf{map}_{r_i}(\pi_i)$
		$\mathsf{PoKAccept}(x, (a_i, c_i, s_i)) \stackrel{?}{=}$ true
		// Check response time
		$t - t' \stackrel{?}{\leq} \tau$

... for rounds $i \in [1, n]$		// Check solutions $\mathsf{puzAccept}^y[d, \beta](\pi_1, ..., \pi_n) \stackrel{?}{=}$ true

Figure 3: ASIC-ZKPoCK **protocol. The protocol executes over n rounds. $\mathcal{V}$ runs** PoKAccept **in each round to check that $\mathcal{P}$ given a valid Σ transcript (a_i, c_i, s_i) and that $\mathcal{P}$ has run within time bound τ. After n rounds, $\mathcal{V}$ runs** $\mathsf{puzAccept}^y$ **to check more than y out of the n PoW puzzle solutions $\pi_1, ..., \pi_n$ are correct with respect to the difficulty specified by d.**

We construct a mapping $\Sigma\mathsf{map}_r(\pi) \rightarrow (c, s)$ as follows. Recall that $\pi = (B, \nu)$. The function $\Sigma\mathsf{map}_r$ partitions $B \rightarrow B[1] \parallel B[2]$. It computes $c = \mathcal{H}(B[1] \parallel r)$ from $B[1]$. Here, $\mathcal{H}(\cdot \parallel r)$ can be viewed as a random hash function selected by $\mathcal{V}$ (using PoW challenge r) to prevent PoW precomputation.

Because s depends on c (and a), $\Sigma\mathsf{map}_r$ is constructed such that $\mathcal{P}$ can specify $s = B[2]$, i.e., encode s in a portion of B distinct from that for c. Given the collision-resistance of $\mathcal{H}$, changing $B[1]$ of course changes c. Changing $B[2]$ also changes $B[1]$: Given a fixed a, a given c has only one corresponding response s, due to the quasi-soundness property discussed in Section 3. In short, $\mathcal{P}$ cannot feasibly construct distinct blocks that map to the same (c, s). At the same time, we allow $\mathcal{P}$ to explore the space of possible nonces ν. To force $\mathcal{P}$ to try multiple puzzles (and thus multiple (c, s)), the nonce space should not be too large. Therefore, we impose in puzAccept the restriction $\nu < \beta$ for a protocol parameter β.

In summary, $\mathcal{P}$ can feed a block header B to an ASIC to solve a puzzle corresponding to some challenge c. Provided that the security parameter β is small enough—i.e., the space of valid ν is small—the probability of $\mathcal{P}$ finding a puzzle solution $\pi = (B, \nu)$ for any single value of c is low. Therefore, $\mathcal{P}$ must w.h.p. try multiple values of c to find a puzzle solution. Consequently, $\mathcal{P}$ is likely to use a pair of triples $(a, c, s) \neq (a, c', s')$ from which w can be extracted.

ASIC-ZKPoCK **protocol.** The $\mathsf{Setup}(1^\lambda)$ algorithm simply provides all the concrete parameters (described later) to both parties. The protocol between $\mathcal{P}$ and $\mathcal{V}$ is presented in Fig. 3. Two further points are worth highlighting. First, we reiterate that the protocol is *interactive*; $\mathcal{V}$ supplies a PoW challenge r as described above. Second, the protocol is *timed*; $\mathcal{P}$'s response is only accepted by $\mathcal{V}$ if it is returned within time τ. The goal, again, is to ensure that computation has taken place in a (fast) ASIC, rather than a (relatively slow) CPU.

B	Block header
ν	PoW puzzle nonce
π	PoW solution
r	PoCK challenge
(a, c, s)	Σ-protocol transcript

d	PoW puzzle difficulty
y	PoW puzzle threshold
τ	Prover time bound
β	Bound on nonce ν size
$\lambda = \ell$	Security parameter
n	Total number of rounds

Figure 4: Protocol notation (left) and parameters (right)

6.3 Security Analysis

In this section, we prove security of ASIC-ZKPoCK by showing that it satisfies the PoCK properties of completeness, forced-revelation, and (honest-verifier) ZK. Later, we demonstrate its practicality by choosing concretely viable parameters. We begin with some simple results (Proofs in App. C.1).

Fact 6.1. Consider $Y \sim \mathsf{Binomial}(n, p)$ and let $\mathsf{F}(n, p, y) = \Pr[Y \leq y]$. If $y < np$, then $\mathsf{F}(y, n, p) \leq exp(-\frac{(np-y)^2}{3np})$. If $n, y = \Theta(\lambda)$, and $p, y/n = \Theta(1)$ with $y/n < p$, then there exists a negligible function $\mathrm{negl}(\lambda)$ such that $\mathsf{F}(y, n, p) \leq \mathrm{negl}(\lambda)$. Alternatively, if $y/n > p$, then $\mathsf{F}(n - y, n, 1 - p) \leq \mathrm{negl}(\lambda)$.

Lemma 6.2. *Let p^Q_{one} denote the probability of $\mathcal{P}$ with hash rate Q successfully computing one PoW puzzle solution in time τ. Define $k = \tau Q/d$. Then: $p^Q_{\text{one}} = 1 - (1 - 1/d)^{kd}$.*

Setting parameters. Looking ahead, to obtain asymptotic security, we set the parameters as follows: d, β are constants in λ. For simplicity, our analysis considers Q and τ to also be constants. We note however that since λ changes the output of the hash function, it may be the case that the hashrate Q degrades with λ (e.g. $Q = \Theta(1/\lambda)$). Here, the round time τ can be increased to maintain $\tau Q = \Theta(1)$; the analysis remains the same. Further, n, y will be $\Theta(\lambda)$ such that y/n is a constant.

Completeness proof. Consider the PoW threshold satisfying $y < np^{Q_{\text{asic}}}_{\text{one}}$. Now, if n and y are $\Theta(\lambda)$ with $y/n = \Theta(1)$, then by Fact 6.1, the completeness error for the honest prover will be $\mathrm{negl}(\lambda)$, which proves completeness of ASIC-ZKPoCK.

CK-soundness proof. Notice first that for ASIC-ZKPoCK to function correctly, it should be infeasible for an adversary to execute the protocol in an enclave (thus on a CPU). Unfortunately, mining puzzles are embarrassingly parallel, which means that in principle, an adversary can use a network of multiple TEE-enabled hosts to solve them. We must therefore characterize security in terms of the size of the network, say m. Appendix C remarks on the possibility of using sequential functions (e.g., VDFs) to remove this constraint.

Furthermore, even assuming that an ASIC is used, in order for $\mathcal{E}$ to extract the witness, we also need to show that at least two distinct challenges c and c' are used in the computation.

Concretely, for CK-soundness, we need to bound the adversarial success probability in two strategies: (1) Compute a valid puzzle solution using only one challenge; (2) Compute a valid puzzle solution using m TEE-enabled CPUs. Appendix C.1 shows through a game-based analysis why it is sufficient to consider these approaches.

We note that while the adversary can send one challenge each to m different ASICs (to prevent extraction caused by two challenges sent to a single ASIC), this requires the adversary to use separate machines for each ASIC and requires it to enforce strong non-collusion properties between these outsourced machines, which is infeasible. We note this in Remark 4 but still allow the adversary to use this strategy within our analysis in Appendix C.1. Additionally, as noted earlier, we assume, as is the case in practice, that the ASICs do not work over encrypted or secret-shared inputs, avoiding MPC over ASIC strategies[9] (see Appendix C.1 for details).

Lemma 6.3. *Let p^{onechal} be the probability (irrespective of hash-rate) of successfully computing a valid puzzle solution using a single challenge c. Then:* $p^{\text{onechal}} = 1 - (1 - 1/d)^{\beta}$.

Now, let $\mathbb{P}^{y,\text{onechal}}_{\text{succ}}$ denote the probability that this happens in more than y (out of n) rounds, i.e., the adversary wins using strategy (1). Notice that $\mathbb{P}^{y,\text{onechal}}_{\text{succ}} = \mathsf{F}(n-y, n, 1-p^{\text{onechal}})$, and so if $\beta = \Theta(1)$ is such that $p^{\text{onechal}} < y/n$, then $\mathbb{P}^{y,\text{onechal}}_{\text{succ}}$ is negligible in λ.

Following (2), the adversary seeks to boost its mining rate while preventing disclosure of w by using a network of m enclave-enabled CPUs. Let Q_{cpu} denote the fastest hash rate achievable in an enclave and consider an adversary that uses a network of m enclave-enabled CPUs, i.e., $Q_{\text{adv}} = mQ_{\text{cpu}} \ll Q_{\text{asic}}$. Consider $p^{Q_{\text{adv}}}_{\text{one}} < y/n = \Theta(1)$. Fact 6.1 directly implies that an adversarial prover succeeds in more than y rounds with at most negligible probability.

In essence, ASIC-ZKPoCK achieves both completeness and CK-soundness when we set the success threshold y such that $np^{mQ_{\text{cpu}}}_{one} < y < np^{Q_{\text{asic}}}_{one}$ and β such that $np^{\text{onechal}} < y$.

(Honest-Verifier) Zero-knowledge proof. Exactly as in [37], the final proof contains only one Σ-protocol transcript, and thus the HVZK is maintained (see App. C.1).

6.4 Practical ASIC-ZKPoCK Parameterization

Achieving both completeness and forced revelation introduces a tension in the tuning of d and τ. For completeness, ASIC-ZKPoCK requires moderately large k_{asic} (corresponding to Q_{asic}). To ensure forced revelation (specifically, to rule out use of CPUs) requires small k_{adv}. To better understand this tension, it is helpful to consider the ratio $N = Q_{\text{asic}}/Q_{\text{cpu}}$, i.e., the speed advantage conferred by an ASIC over a CPU. Given an adversary with a network of m CPUs, the ratio $k_{\text{asic}}/k_{\text{adv}} = (\tau Q_{\text{asic}}/d)/(\tau n Q_{\text{cpu}}/d) = N/m$. A secure and efficient (in terms of n and y) parameterization requires $m \ll N$, i.e., that an adversary cannot feasibly come close to approaching ASIC speeds with a network of CPUs.

Example practical parameterizations. We derive two concrete parameterizations for ASIC-ZKPoCK; one for a recent top-of-the-line mining ASIC [15] with $Q_{\text{asic}} \approx 2^{47}$ H/s and one for a cheap, outmoded mining ASIC with $Q_{\text{asic}} \approx 2^{43}$ H/s. The hashrate for a state-of-the-art CPU [55] is taken as $Q_{\text{cpu}} \approx 2^{26}$ H/s. We show that $n = 8$ and $n = 12$ rounds respectively for the the two settings yields negligible completeness and soundness error against an adversary with a network of 10,000 CPUs. Appendix C contains further details.

Figure 5 compares honest and adversarial prover success rates for the top-of-the-line ASIC. We also implement an end-to-end system using the outmoded ASIC (see Section 7).

[9]Even if future ASICs do support such MPC, we posit that ASICs which do not support this will always be faster, allowing us to appropriately parameterize our protocol.

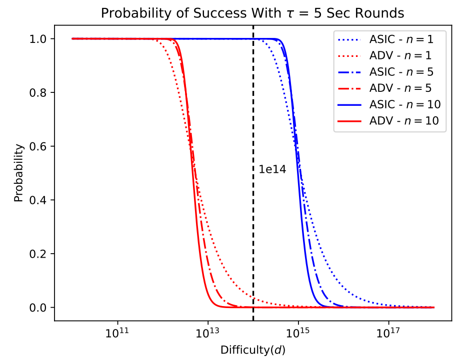

Figure 5: ASIC-ZKPoCK **allows for practical parameters to achieve overwhelming probability of completeness by an ASIC (hashrate 154 TH/s) and negligible probability of success by an adversary with 10,000 state-of-the-art CPUs (hashrate 72 MH/s).** n **denotes number of rounds. Here, prover is required to compute more than** $y = \lfloor n/2 \rfloor$ **valid PoW puzzle solutions. The black line represents a parameterization of** $d = 10^{14}$**, for which adversarial success probability is negligible but that of ASIC is close to 1.**

Smart Contract	LOC	Operation	Gas cost
ASIC-ZKPoCK Verification Contract	140	Register new job	366,485
		Initiate challenge	70,209
		Verify proof	7,620,401
CK Registry	130	Record a new proof	54,260

Figure 6: Lines of code (LOC) for SMACK **and gas costs for various operations. As of Oct 2023, the cost for 100,000 gas is around $1.55.**

7 SMACK: END-TO-END IMPLEMENTATION

To demonstrate CK proofs in a practical setting, we prototype SMACK (*SMArt-contract enabled CK*)—a *complete, end-to-end* CK system on Ethereum. SMACK offers a good proof of concept of CK practicality for two reasons. First, smart contracts are highly resource constrained, with limited, expensive computational power, coarse-grained, approximate measurement of time, and no ability to maintain secret state. Making CK proofs work in this austere environment strongly evidences their general practicality. Second, deployment of SMACK in Ethereum has the benefit of supporting a wide variety of blockchain-based services, e.g., voting, Atomic NFTs, enforced NFT royalties, etc., as described in Section 2.

SMACK allows for any desired CK method to be used. To demonstrate the practicality of our approaches, we implement two CK variants in SMACK: ASIC-ZKPoCK and a TEE-based CK proof system for Android devices that we call *lightweight CK* (see Appendix D.1). Fig. 6 reports the costs associated with SMACK's contracts.

CK Registry. SMACK supports CK for private keys associated with Ethereum addresses (public keys). CK proofs remain indefinitely valid (once an sk is exposed to $\mathcal{E}$, the fact of exposure remains true forever). Therefore we deploy a smart contract (at address 0x25B270...eE3966), called the CK *registry* that maintains a permanent record of addresses for which CK proofs have been provided (to the verification contract). This allows for applications to cheaply query whether a given address has an associated CK proof. As an example, we use this registry and verification contracts to mint

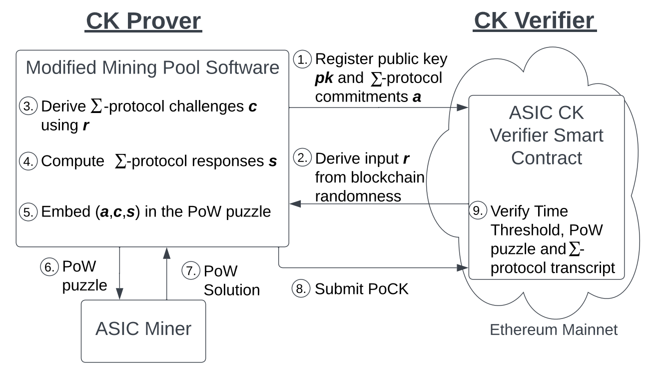

Figure 7: ASIC-ZKPoCK **system architecture. The encircled numbers correspond to steps in a protocol execution. Our CK Prover consists of a (modified) pool mining software and an ASIC miner. Our CK Verifier is implemented as a Smart Contract on Ethereum mainnet.**

Atomic NFTs (contract deployed at 0x6986fc...FcAC38) for CK addresses. We provide more details on the registry in Appendix D.

7.1 ASIC-ZKPoCK implementation

We now detail our ASIC-ZKPoCK implementation, which is the more technically challenging and intricate of the two currently supported CK methods in SMACK. We make use of a Bitcoin miner that is several years old—the Bitmain Antminer S9 with a hashrate of 13TH/s. While this hardware cannot competitively mine Bitcoin at this point, it is sufficient to achieve low error bounds in our scheme[10](details in Appendix C). It demonstrates that cheap, outmoded hardware (which cost us around $100) can be repurposed successfully for ASIC-ZKPoCK. We implement the verifier as an Ethereum smart contract, deployed at the address 0xAC86fD...B39f4b, and carry out an example verification in transaction 0xf67b1c...6f4106. All the source code and scripts can be found at https://github.com/CK-anon/SMACK.

System Architecture. Figure 7 shows our system architecture. The prover communicates with the ASIC miner using an open source pool mining software which implements the standard Stratum V2 protocol [4] for allocating work to the miner and fetching the PoW solution. The miner does some sanity checks on the input block data (e.g. for increasing block heights, correctly sized fields, etc.). Therefore, our prover makes use of a private Bitcoin network to generate valid PoW puzzles for the miner. This also allows us to configure the difficulty of the PoW puzzles. We use Schnorr's protocol [60] for our underlying PoK Σ-protocol. While our verification smart contract is used for verifying complete knowledge of the private keys of Ethereum addresses, it can be used for any general PoK value x. To generate the verfier's random challenge r, our smart contract uses randomness from the Ethereum proof-of-stake network [5].[11] For pool mining, the Merkle root in the block header is expanded into a special coinbase transaction along with the adjacent Merkle branches. We place s (32 bytes in our case) inside this coinbase transaction, which is allowed to carry arbitrary data. Appendix D describes how to correctly set the nonce range β.

[10]As $\tau = 12s$ for our implementation, any time advantage that an adversary can have in communication with the Ethereum blockchain does NOT meaningfully affect the prover's completeness probability.
[11]This randomness is biasable to small extent but is not material.

8 RELATED WORK

Concurrent work: Individual Cryptography [34]. In recent work concurrent to the first online version of our paper, Dziembowski et al. [34] propose the notion of *individual cryptography*, with a goal similar to ours—to prove that e.g., a key is held unencumbered by an individual party instead of within trusted hardware, or an MPC committee. This however is the extent of similarities. We highlight key differences below:

First, and most importantly our CK formalism and setting is more general—the main focus of their paper is on a simpler setting where the verifier also has access to the witness w—this does not model any of our usecases. Furthermore, [34] explicitly formalizes "slow" vs "fast" hash function computation as part of their primitive to model the computational difference between trusted (MPC or TEE) and untrusted (ASIC) computation; this reduces the scope of their primitive. In contrast, we provide a general resource-based formalism which captures other CK constructions—e.g., our SGX-based construction which *cannot* be modeled through their formalism. Our ASIC-based scheme also substantially differs from theirs. It further shows connections to Fischlin [37], allows the usage of generic Σ-protocols, and can be practically parameterized.

Second, while [34] positions their work as a theoretical exploration, we provide significant practical motivations: bribery attacks through key encumbrance on voting protocols, as well as new applications enabled by CK. Furthermore, we demonstrate practicality by not only implementing our constructions but also deploying them via Ethereum smart contracts to enable CK-based applications.

Preventing sharing of cryptogrpahic functionalities. Traitor-tracing [27] and leakage-deterring [45] schemes can both model the following abstraction: If an adversary with key sk attempts to share some cryptographic functionality (e.g., signing, encryption) that uses sk, then any entity that it shares this with could also recover sk. Note that this abstraction is orthogonal to CK; it could be feasibly used for voting but *does not apply to other CK applications.* If black-box recovery is possible, this can deter adversarial encumbrance (and bribery) since the user could always recover her own key.

Even then, these techniques do not capture all kinds of encumbrance we consider for several reasons: First, and most importantly, they are *purpose-built* for not only a specific functionality (e.g., signing) but also a specific scheme—given one scheme, a different scheme can be constructed that deters sharing as above. That is, they cannot *generically* prevent any type of encumbrance; in contrast, this is achieved by CK. Additionally, for signature schemes (required for voting) in particular, recovery is non-black-box. Second, they *do not prevent MPC-based encumbrance* where no party has the secret or a fully-operational functionality for even a single input. Finally, we note that requiring modification of the encryption/signature scheme would be a non-starter in the typical blockchain settings we consider where the voting key also controls, e.g., digital assets.

9 CONCLUSION

We have shown a fundamental limitation in traditional proofs of knowledge (PoKs)—the fact that they do not actually prove knowledge by a prover in light of encumbrance-based attacks using TEEs or MPC. This gap in the PoK model introduces a range of coercive attacks, many explored in earlier works. We therefore formalized complete knowledge (CK) as a stronger version of PoKs. CK can help in the design of practical protocols—using TEEs and ASICs—that are resistant to coercive attacks. We hope that our work will stimulate the development of new CK constructions and their use in e-voting, deniable authentication, lease-resistant credentials, and many other applications.

REFERENCES

[1] 2020. Signal website. https://signal.org.
[2] 2022. Discussion: Play Integrity API. https://forum.xda-developers.com/t/discussion-play-integrity-api.4479337/.
[3] 2022. Intel Enhanced Privacy ID (EPID) Security Technology. sintel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html.
[4] [Accessed December 2022]. Ethereum Consensus Notes. https://en.bitcoin.it/wiki/Stratum_mining_protocol.
[5] [Accessed December 2022]. Ethereum Consensus Notes. https://eth2book.info/bellatrix/part2/building_blocks/randomness/.
[6] [Accessed June 2022]. Oasis Labs website. https://www.oasislabs.com/.
[7] Referenced Nov. 2022. Android Developers: Verifying hardware-backed key pairs with Key Attestation. https://developer.android.com/training/articles/security-key-attestation#root_certificate.
[8] Amazon Web Services. Referenced Nov. 2022. AWS Nitro Enclaves website. https://aws.amazon.com/ec2/nitro/nitro-enclaves/.
[9] AMD. Jan. 2020. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. *White Paper* (Jan. 2020).
[10] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *HASP*. 7.
[11] Android Open Source Project. Referenced Nov. 2022. Android Open Source Project: Key and ID Attestation. https://source.android.com/docs/security/features/keystore/attestation.
[12] Apple Inc. Referenced Nov. 2022. Apple Developer Website: Establishing your app's integrity. https://developer.apple.com/documentation/devicecheck/establishing_your_app_s_integrity.
[13] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. 2000. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*. 255–270.
[14] Josh Benaloh and Dwight Tuinstra. 1994. Receipt-free secret-ballot elections. In *STOC*. 544–553.
[15] Bitmain. 11 Feb. 2022. Specifications of T19/S19 Liquid-Cooling Miner. https://support.bitmain.com/hc/en-us/articles/4418373232153-Specifications-of-T19-S19-Liquid-Cooling-Miner.
[16] Remco Bloemen, Leonid Logvinov, and Jacob Evans. 2017. EIP 712. https://eips.ethereum.org/EIPS/eip-712.
[17] Nikita Borisov, Ian Goldberg, and Eric Brewer. 2004. Off-the-record communication, or, why not to use PGP. In *WPES*. 77–84.
[18] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *USENIX Security*. 3917–3934.
[19] John Brainard, Ari Juels, Ronald L Rivest, Michael Szydlo, and Moti Yung. 2006. Fourth-factor authentication: somebody you know. In *CCS*. 168–178.
[20] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*. 315–334.
[21] V. Buterin. 11 Jan 2021. Why we need wide adoption of social recovery wallets. vitalik.ca blog post at https://vitalik.ca/general/2021/01/11/recovery.html.
[22] V. Buterin. 2 May 2019. Minimal anti-collusion infrastructure (MACI). Ethereum Research blog post at https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413.
[23] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*. 136–145.
[24] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. In *TCC*. 61–85.
[25] David Chaum, Peter YA Ryan, and Steve Schneider. 2005. A practical voter-verifiable election scheme. In *ESORICS*. 118–139.
[26] David L Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.
[27] Benny CHor, AMos Fiat, and Moni Naor. 1994. Tracing Traitors. In *CRYPTO*. 257–270.
[28] Jeremy Clark and Urs Hengartner. 2011. Selections: Internet voting with over-the-shoulder coercion-resistance. In *FC*. 47–61.
[29] Michael R Clarkson, Stephen Chong, and Andrew C Myers. 2008. Civitas: Toward a secure voting system. In *IEEE S&P*. 354–368.
[30] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2017. A formal security analysis of the Signal messaging protocol. In *EuroS&P*. 451–466.
[31] Philip Daian, Tyler Kell, Ian Miers, and Ari Juels. 2018. On-Chain Vote Buying and the Rise of Dark DAOs. https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/.
[32] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. 2006. Deniable authentication and key exchange. In *CCS*. 400–409.
[33] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *CRYPTO*. 139–147.
[34] Stefan Dziembowski, Sebastian Faust, and Tomasz Lizurej. 2023. Individual Cryptography. In *CRYPTO*. 547–579.
[35] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-knowledge proofs of identity. *Journal of cryptology* 1, 2 (1988), 77–94.
[36] O. Fernau. 13 Aug. 2022. Royalty-Free Sudoswap Is Finding Favor With NFT Traders. *The Defiant* (13 Aug. 2022).
[37] Marc Fischlin. 2005. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*. 152–168.
[38] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *STOC*. 218–229.
[39] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The knowledge complexity of interactive proof systems. In *STOC*. 291–304.
[40] Lachlan J Gunn, Ricardo Vieitez Parra, and N Asokan. 2019. Circumventing cryptographic deniability with remote attestation. *PETS* 2019, 3 (2019), 350–369.
[41] Markus Jakobsson and Ari Juels. 1999. Proofs of work and bread pudding protocols. In *Secure information networks*. 258–272.
[42] K. Wei Jie. 12 Oct. 2021. Release Announcement: MACI 1.0. Medium Post at https://medium.com/privacy-scaling-explorations/release-announcement-maci-1-0-c032bddd2157.
[43] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *WPES*. 61–70.
[44] Aggelos Kiayias and Philip Lazos. 2022. SoK: Blockchain Governance. *arXiv preprint 2201.07188* (2022).
[45] Aggelos Kiayias and Qiang Tang. 2013. How to Keep a Secret: Leakage Deterring Public-key Cryptosystems. In *CCS*. 943–954.
[46] Yashvanth Kondi and abhi shelat. 2022. Improved Straight-Line Extraction in the Random Oracle Model With Applications to Signature Aggregation. In *ASIACRYPT*.
[47] Brad Linder. 11 Mar. 2020. Some apps may stop working on rooted Android phones due to SafetyNet update.
[48] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security*. 1553–1570.
[49] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. 2017. A smart contract for boardroom voting with maximum voter privacy. In *FC*. 357–375.
[50] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. 2016. Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave. In *HASP*. 1–9.
[51] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *HASP*. 10.
[52] Rafael Pass. 2003. On deniability in the common reference string and random oracle model. In *CRYPTO*. 316–337.
[53] Rafael Pass, Elaine Shi, and Florian Tramèr. 2017. Formal Abstractions for Attested Execution Secure Processors. In *EUROCRYPT*. 260–289.
[54] Rafael Nat Josef Pass. 2006. *A precise computational approach to knowledge*. Ph.D. Dissertation. Massachusetts Institute of Technology.
[55] Hoai Luan Pham, Thi Hong Tran, Tri Dung Phan, Vu Trung Duong Le, Duc Khai Lam, and Yasuhiko Nakashima. 2020. Double SHA-256 Hardware Architecture With Compact Message Expander for Bitcoin Mining. *IEEE Access* 8 (2020), 139634–139646. https://doi.org/10.1109/ACCESS.2020.3012581
[56] David Pointcheval and Jacques Stern. 2000. Security arguments for digital signatures and blind signatures. *Journal of cryptology* 13, 3 (2000), 361–396.
[57] Ivan Puddu, Daniele Lain, Moritz Schneider, Elizaveta Tretiakova, Sinisa Matetic, and Srdjan Capkun. 2019. TEEvil: Identity Lease via Trusted Execution Environments. *arXiv preprint 1903.00449* (2019).
[58] Charles Rackoff and Daniel R Simon. 1991. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*. 433–444.
[59] Amit Sahai. 1999. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*. 543–553.

[60] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *CRYPTO*. 239–252.
[61] C. P. Schnorr. 1991. Efficient Signature Generation by Smart Cards. *J. Cryptol.* 4, 3 (jan 1991), 161–174. https://doi.org/10.1007/BF00196725
[62] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. 2017. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *ESORICS*. 456–474.
[63] Michael Bedford Taylor. 2017. The evolution of bitcoin hardware. *Computer* 50, 9 (2017), 58–66.
[64] Langston Thomas. 26 Apr. 2022. Fractional NFTs: The Good, the Bad, and the Weird. *NFT Now* (26 Apr. 2022).
[65] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security*. 991–1008.
[66] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. SGAxe: How SGX fails in practice. https://sgaxe.com/files/SGAxe.pdf.
[67] Joseph Weinberg. 21 Jan. 2022. NFTs and compliance: Why we need to be having this conversation. *Cointelegraph* (21 Jan. 2022).
[68] E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. 2022. Decentralized Society: Finding Web3's Soul. *Available at SSRN 4105763* (2022).
[69] Andrew C Yao. 1982. Protocols for secure computations. In *FOCS*. 160–164.

A DEFERRED POCK FORMALISM

Remark 2 (Trivial PoK and PoCK protocols). Forced revelation implies an interesting separation between trivial protocols for PoK and PoCK. Recall that a trivial PoK protocol is for $\mathcal{P}$ to simply provide the witness to $\mathcal{V}$. Of course, such a protocol does not offer any privacy (e.g., zero-knowledge) properties. Notice however, that this would not be a PoCK protocol (since $\mathcal{E}$ does not come into play). This may seem surprising but is in fact an important consequence of the PoCK setting.

Abstractly, if the verifier receives the witness through an encrypted channel (e.g., through TLS), then it would also be possible for two parties constituting the prover and holding only shares of w to directly compute the required encryption of w for the channel. In such a case, the trivial PoK protocol could be simulated by two prover parties, neither of which has complete knowledge of w. Importantly, this is also possible if $\mathcal{V}$ is run inside a TEE since there is no point at which w is revealed in plaintext. Consequently, simply sending w will not be a PoCK protocol.

Intuitively, if we can guarantee that $\mathcal{V}$ is not run inside a TEE, then it will be possible to eavesdrop on the witness, making the trivial protocol a PoCK. One way this can be done is by having $\mathcal{V}$ itself be a TEE instance with the assumption that it is not practical to run a TEE inside another TEE.

This unearths a dependence of PoCKs on how the communication to $\mathcal{V}$ is defined, which is not seen for standard PoKs.

A.1 Eavesdropper Undetectability Details

Side channel on the usage of $\mathcal{E}$. To differentiate between whether $\mathcal{E}$ was used to recover the witness or not, we consider a side channel that provides a remote adversary $\mathcal{A}$ with this information (explicitly, this is added to the VIEW of an adversarial verifier). Consequently, $\mathcal{A}$ can now take actions based on whether the witness was extracted; notably, as we show later, this gives $\mathcal{A}$ additional advantages in our vote bribery scenario.

It is important to emphasize here that the witness can always be extracted, (thereby satisfying CK-soundness); the difference is in *whether it actually was*. We also note that this detection ability is a highly unconventional power given to the adversary; while $\mathcal{A}$ cannot itself recover the witness, it is still made aware of whether $\mathcal{P}$ did. This is non-standard in the context of existing literature—when $\mathcal{R}$ (and $\mathcal{E}$) is not in the domain of $\mathcal{A}$ (because otherwise $\mathcal{A}$ could extract the witness itself), we note that there is likely no practical side channel that reveals to $\mathcal{A}$ whether $\mathcal{P}$ chose to extract.

Still, to ensure that PoCK protocols can be correctly deployed in applications where side channels need to be accounted for, we introduce an explicit assumption—*eavesdropper undetectability*—as the property that an adversary cannot detect whether $\mathcal{E}$ was run or not. We show how this property is critical for incentive-compatibility in our vote bribery example. This also serves to highlight the non-triviality of our CK setting.

Formal description. To define eavesdopper undectability, we relax our earlier modeling assumption that $\mathcal{E}$ exists as a man-in-the-middle entity for $\mathcal{R}$ and can snoop on any queries made to it. Instead, we will give the prover the ability to make queries to $\mathcal{R}$ *without the usage of* $\mathcal{E}$; we use $\mathcal{R} \setminus \mathcal{E}$ to denote this oracle. The honest prover $\mathcal{P}$ will still make use of $\mathcal{E}$.

We can now define eavesdopper undectability as the following property: For pp, inputs x, and $\mathcal{R} \in \mathfrak{R}$, for all (possibly malicious) provers $\mathcal{P}'$ and verifiers $\mathcal{V}'$, there exists $\widehat{\mathcal{P}}$ such that the following ensembles are indistinguishable:

$$\begin{aligned}&\mathrm{VIEW}_{\mathcal{V}'}\left(\mathcal{P}'^{\mathcal{R}\setminus\mathcal{E}}(pp,x),\mathcal{V}'(pp,x)\right)\\ \approx\ &\mathrm{VIEW}_{\mathcal{V}'}\left(\widehat{\mathcal{P}}^{\mathcal{R}}(pp,x),\mathcal{V}'(pp,x)\right).\end{aligned}$$

Intuitively, this means that no $\mathcal{V}'$ can distinguish whether it is interacting with a prover which uses $\mathcal{E}$ or one which does not.

In the example that follows, we briefly describe how a side channel which informs $\mathcal{A}$ of whether $\mathcal{E}$ was run breaks security in our vote bribery scenario. Eavesdropper undetectability is required here to prevent this attack. We leave further exploration of this property to future work.

Incentive compatibility example. Suppose that a PoCK Π is used in a voting application to mitigate the risk of key-encumbrance based bribery. As stated earlier, Π ensures that the eavesdropper $\mathcal{E}$ *can recover* the secret key but says nothing about *whether the recovery is actually carried out*—our previous CK formalism implicitly assumes that $\mathcal{E}$ will always output the recovered key.

Still, if there was some side channel through which a remote adversary $\mathcal{A}$ could detect whether the key was extracted (by $\mathcal{E}$ or $\mathcal{P}$ in general), then it could condition its bribe on this extraction action not being taken. The consequence of such a conditional bribe is that while $\mathcal{P}$ has the ability to learn her key, she will be *incentivized not to do so* in order to profit from the bribe. In particular, while $\mathcal{P}$ *can* always learn her full key, making the protocol satisfy CK-soundness, $\mathcal{A}$ will be able to detect such an action and refuse to pay $\mathcal{P}$; this incentivizes $\mathcal{P}$ to not learn her key even if she is able to. The key will therefore remain encumbered. Eavesdropper undetectability removes this side channel vulnerability.

B DEFERRED DETAILS FOR SGX-POCK

We detail the full $\mathcal{G}_{\mathsf{SGX}}$ functionality in Fig. 8.

B.1 Security Proof for SGX-PoCK

We now provide the full details of the CK-soundness proof for SGX-PoCK.

$\mathcal{G}_{\mathsf{SGX}}[S, \mathsf{Reg}]$

On initialization: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\$ \mathsf{S.kg}(1^\lambda), I \leftarrow \emptyset$.

On receive $\mathtt{getpk}^*()$ from some party $\mathcal{M}$:

Send mpk to $\mathcal{M}$.

On receive $\mathtt{install}^*(idx, \mathsf{prog})$ from some $\mathcal{M} \in \mathsf{Reg}$:

If $\mathcal{M}$ is honest, assert $idx = sid$.

Generate nonce $eid \in \{0,1\}^\lambda$.

Store $I[eid, \mathcal{M}] = (idx, \mathsf{prog}, 0)$ and send eid to $\mathcal{P}$.

On receive $\mathtt{resume}^*(eid, \mathsf{inp})$ from some $\mathcal{M} \in \mathsf{Reg}$:

Let $(idx, \mathsf{prog}, \mathsf{mem}) = I[eid, \mathcal{M}]$, abort if not found. Compute $(\mathsf{out}, \mathsf{mem}') = \mathsf{prog}(\mathsf{inp}, \mathsf{mem})$.

Update $I[eid, \mathcal{M}] = (idx, \mathsf{prog}, \mathsf{mem}')$.

Let $\sigma = \mathsf{S.sign}_{\mathsf{msk}}((idx, eid), \mathsf{prog}, \mathsf{out})$.

Send (out, σ) to $\mathcal{M}$.

Figure 8: $\mathcal{G}_{\mathsf{SGX}}$ global functionality from [53]. Starred operations are re-entrant activation points.

CK-Soundness proof. Let game G_1 denote the CK-soundness game, where $\mathcal{A}$ must convince $\mathcal{V}$ without the witness being extracted by $\mathcal{E}$. In particular, given $x \in L$ (with witness w) and pp, for adversary $\mathcal{A}$ using resource $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$, we define the CK-soundness advantage, i.e., the probability of winning G_1 as follows:

$$\mathsf{CK\text{-}Adv}(\mathcal{A}^{\mathcal{R}'}) = \Pr\left[\begin{array}{c}\langle \mathcal{A}^{\mathcal{R}'}(pp,x), \mathcal{V}(pp,x)\rangle = 1 \\ \wedge\ (x, w' = \mathsf{out}_{\mathcal{E}}) \notin R_L\end{array}\right]$$

Observe first that the CK-soundness definition holds whenever CK-Adv is bounded by $\mathsf{negl}(\lambda)$. We will show that for any $\mathcal{A}$ using a "practically instantiable" resource $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$, its CK-Adv is negligibly small. First, observe that $\mathcal{V}$ outputs 1 only on a valid signature from the honest resource $\mathcal{R} = \mathcal{G}_{\mathsf{SGX}}$, i.e., to convince $\mathcal{V}$, it must be the case that $\mathcal{A}^{\mathcal{R}'}$ constructed a valid $\mathcal{G}_{\mathsf{SGX}}$-signature on w. We will use this to bound CK-Adv by the SUF-CMA security of the signature scheme S—here, $\mathcal{G}_{\mathsf{SGX}}$ is instantiated with the key sampled by the challenger in the SUF-CMA game.

Suppose that $\mathcal{A}$'s usage of $\mathcal{R}'$ results in q queries made to $\mathcal{G}_{\mathsf{SGX}}$. Without loss of generality, we can assume that any queries to $\mathcal{G}_{\mathsf{SGX}}$ are not made directly by $\mathcal{R}'$ but instead made through $\mathcal{A}$ as instructed by $\mathcal{R}'$ (i.e., $\mathcal{A}$ has access to two resources—$\mathcal{R}'$ and $\mathcal{G}_{\mathsf{SGX}}$).

Now let game G_2 be the same as G_1 except now, if (x, w) $(\in R_L)$ is queried to $\mathcal{G}_{\mathsf{SGX}}$, $\perp$ is output instead of the signature; denote this by $\mathcal{G}^*_{\mathsf{SGX}}$. Notice that $\Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}_{\mathsf{SGX}}}$ wins in $G_1] = \Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_2]$ since whenever $\mathcal{A}$ queries $\mathcal{G}_{\mathsf{SGX}}$ with w, it will always be output by $\mathcal{E}$.

Let game G_3 be the same as G_2 except that the part with the extractor is removed (i.e., $\mathcal{A}$ wins if it just convinces $\mathcal{V}$). It is easy to see that $\Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_2] \leq \Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_3]$ since the adversary wins in G_3 whenever it wins in G_2.

Now, with the assumption that no practical resource in $\mathfrak{R}_{\mathsf{all}}$ (intuitively other than $\mathcal{G}_{\mathsf{SGX}}$) can give additional information about the signatures (this is equivalent to assuming that $\mathcal{R}'$ can be simulated without access to the msk of $\mathcal{G}_{\mathsf{SGX}}$), we can remove any additional power given by $\mathcal{R}'$. In particular, consider game G_4 where any queries to $\mathcal{R}'$ return $\perp$; denote this resource by $\mathcal{R}'_\perp$. Under the aforementioned assumption, it holds that $\Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_3] = \Pr[\mathcal{A}^{\mathcal{R}'_\perp, \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_4]$. Further, since $\mathcal{R}'_\perp$ only returns $\perp$, we can construct $\mathcal{B}$ which never queries $\mathcal{R}'$ such that $\Pr[\mathcal{A}^{\mathcal{R}'_\perp, \mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_4] \leq \Pr[\mathcal{B}^{\mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_4]$.

Recall from earlier that this $\mathcal{G}^*_{\mathsf{SGX}}$ returns $\perp$ when queried with the witness w and that $\mathcal{B}$ wins in G_4 if it convinces $\mathcal{V}$, i.e., outputs a valid signature on w. Now, let game G_5 be the standard SUF-CMA security game for the signature scheme S. We can construct an adversary C for this game as follows: C runs $\mathcal{B} \leftrightarrow \mathcal{V}$ internally and forwards any queries to $\mathcal{G}^*_{\mathsf{SGX}}$ from $\mathcal{B}$ to its own SUF-CMA signing oracle. If $\mathcal{V}$ outputs 1, then C outputs the last message sent by $\mathcal{B}$ as its own SUF-CMA forgery. Notice now that C wins the SUF-CMA game exactly when $\mathcal{B}$ convinces $\mathcal{V}$—i.e., it holds that $\Pr[\mathcal{B}^{\mathcal{G}^*_{\mathsf{SGX}}}$ wins in $G_4] = \Pr[C$ wins in $G_5] = \mathsf{SUF\text{-}CMA}_S(C)$. Further, C makes exactly as many queries to its signing oracle as $\mathcal{B}$ makes to $\mathcal{G}_{\mathsf{SGX}}$.

Finally, since the signature scheme S being used is SUF-CMA secure, we know that the adversarial advantage for any adversary is bounded by some $\mathsf{negl}(\lambda)$ which in turn implies that the CK-Adv for any $\mathcal{A}$ and $\mathcal{R}'$ is bounded by $\mathsf{negl}(\lambda)$. This completes the proof.

Making SGX-PoCK **satisfy zero-knowledge.** Intuitively, to make the protocol zero-knowledge, there must exist a simulator $\mathcal{S}$ that can simulate the protocol transcript without access to the witness. Further note that $\mathcal{S}$ should also not be able to program the master secret key of the SGX (this is accounted for since we model the SGX resource as a GUC functionality).

We now briefly describe how SGX-PoCK can be made zero-knowledge. As is the case with other GUC protocols, $\mathcal{S}$ will require a trapdoor in other to simulate transcripts. Towards this, intuitively, we introduce a trapdoor τ which allows the generation of arbitrary SGX attestations for our protocol.

Specifically, $\mathcal{V}$ first chooses a trapdoor τ and submits it to $\mathcal{G}_{\mathsf{SGX}}$ which returns an attestation σ_c on $c = \mathsf{owf}(\tau)$ where owf is a one-way function. $\mathcal{V}$ sends the (c, σ_c) to $\mathcal{P}$ who then submits (c, x, w) to $\mathcal{G}_{\mathsf{SGX}}$. Before providing an attestation that the witness w was seen, $\mathcal{G}_{\mathsf{SGX}}$ ensures that the correct c was input. Finally, the $\mathcal{V}$ can check the correctness of the attestation to complete the proof. Note that this protocol also requires $\mathcal{V}$ to possess an SGX device.

This construction will now be zero-knowledge since the simulator $\mathcal{S}$ can use the trapdoor τ to forge attestations and simulate the interaction between $\mathcal{P}$ and $\mathcal{V}$. The proof is analagous to the one from [53, Thm. 19].

Remark 3 (SGX inside SGX). SGX-PoCK can in fact be modified to work even when an SGX can be run inside another SGX as long as this can be done only a finite number of times. Specifically, if is practical to run k layers of SGX but not $k+1$ layers, then SGX-PoCK can be modified to use the k-layer SGX as the resource $\mathcal{R}$; $\mathcal{P}$ now obtains an attestation from this $\mathcal{R}$. By assumption, since a $(k+1)$-SGX is not practical, the witness submitted to $\mathcal{R}$ will be seen by $\mathcal{E}$ allowing for extraction.

This also serves to future proof our protocol in case of advances in TEE infrastructure.

C DEFERRED DETAILS FOR ASIC-ZKPOCK

C.1 Proofs

PROOF OF FACT 6.1. This follows directly from the lower-tail Chernoff bound. □

PROOF OF LEMMA 6.2. In time τ, $\mathcal{P}$ can compute $Q\tau = kd$ hash evaluations. Recall that the probability of finding a solution for a single evaluation is $1/d$. Therefore, the probability of finding a solution in kd evaluations i.e., p_{one}^{Q} can be given by $1 - (1 - 1/d)^{kd}$. □

PROOF OF LEMMA 6.3. The probability of finding a solution for a single evaluation is $1/d$. Since the nonce-space is of size β, the probability that no nonce works is $(1 - 1/d)^{\beta}$. Therefore the probability p^{onechal} that some nonce works is $1 - (1 - 1/d)^{\beta}$. □

ASIC-ZKPoCK **CK-soundness proof.** Similar to the SGX-PoCK CK-soundness proof, let G_1 denote the CK-soundness game where $\mathcal{A}$ must convince $\mathcal{V}$ (in more than y out of n rounds) without the witness being extracted by $\mathcal{E}$ and define CK-Adv($\mathcal{A}^{\mathcal{R}'}$) similarly as before given $\mathcal{A}$ and resource $\mathcal{R}'$.

Now define game G_2 to be exactly as G_1 except that only the first query made (by e.g., $\mathcal{A}$ or $\mathcal{R}'$) to each honest resource in $\mathfrak{R}$ is returned correctly; all other queries return $\perp$. Further, define game G_3 similarly except that all queries to resources in $\mathfrak{R}$ are returned as $\perp$. Recall that making two queries to an ASIC resource allows $\mathcal{E}$ to extract the witness because of the quasi-soundness of the sigma protocol. Therefore, we have:

$$\Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_1] \leq \Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_2] + \Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_3]$$

Now, for G_2, suppose that the adversary deploys one challenge each to m different ASICs on different machines to not have two queries to the same ASIC. While we do analyze this strategy in the security proof, we note that this actually requires the adversary to enforce strong non-collusion requirements in order to succeed, which is likely infeasible; see Remark 4 for details. For small enough $m = \Theta(1)$, we can set parameters such that the probability that $\mathcal{A}$ wins in G_2 is negligible in λ. To see why, note that similar to Lemma 6.3, the probability that m challenges are sufficient (in 1 round) is $p = 1 - (1 - 1/d)^{\beta m}$. For small enough m, we can set $\beta = \Theta(1)$ such that the $p < y/n$, and therefore $\Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_2]$ is negligible in λ.

Now, the probability that $\mathcal{A}$ wins in G_3 is exactly the probability that it wins without using the ASIC resource. Assume that there is no practical resource that is encrypted (or in other ways prevents eavesdropping by $\mathcal{E}$; this also prevents for instance, fast ASICs that can compute hashes in MPC with secret-shared inputs) and computes hashes faster than $Q_{\text{adv}} = mQ_{\text{cpu}}$ (i.e., equivalent to a network of m CPUs). We note that although their setting is simpler, [34] also makes a qualitatively identical assumption by modeling all MPC-based hash queries as "slow" as opposed to "fast" ASIC queries. Consequently, the probability that $\mathcal{A}$ wins in $\mathcal{G}_2$ can be bounded using Lemma 6.2 as follows: Suppose that parameters are set such that $p_{\text{one}}^{Q_{\text{adv}}} < y/n = \Theta(1)$. From Fact 6.1, it is easy to see that the the probability $\mathcal{A}$ succeeds in more than y rounds, i.e., wins in game G_3 is negligible in λ.

Finally, this means that we can bound the CK-Adv by negl(λ) which completes the proof.

ASIC-ZKPoCK **HVZK proof.** We now provide additional details on the honest-verifier zero-knowledge (HVZK) proof.

Recall that the proof $\pi = (B, v)$ sent to $\mathcal{V}$ contains only a single Σ-protocol transcript. We can define the zero-knowledge simulator $\mathcal{S}$ as follows: Given the public parameters pp, and the statement x, first randomly sample r. Intuitively, $\mathcal{S}$ will now use the ASIC-resource $\mathcal{R}$ to randomly sample $B[1]$ until a satisfying PoW solution v is found. In essence, $\mathcal{S}$ will act the same way as the honest prover $\mathcal{P}$ except in the computation of the Σ-protocol transcript (a, c, s)—instead of using the pre-committed a and the challenge c to compute s, it will run the simulator from the Σ-protocol to generate (a, s) given c.

Formally, given $B[1]$, let $c = \mathcal{H}(B[1] \parallel r)$. Run the simulator for the Σ-protocol on c to obtain a simulated transcript (a, c, s) for knowledge of a witness for x; this is possible since the Σ-protocol is also HVZK. Finally, use $\mathcal{R}$ to find whether there exists a nonce v for the PoW puzzle, i.e., satisfying $\mathcal{H}(B[1] \parallel B[2] = s, v) < 2^{\ell}/d$. Note that ℓ and d are given in the parameters pp. If such a v does not exist, then a different $B[1]$ is sampled and the process is repeated.

C.2 Remarks

Remark 4 (Network of machines with single-challenge ASICs.). A sophisticated strategy that the adversary might attempt in order to bypass our protocol and encumber the key in a TEE is to utilize an outsourced network of machines, each equipped with an ASIC, in such a way that each ASIC is given only one challenge to solve. This ensures that no machine gets access to two challenges that would enable extraction of the witness. This strategy is highly impractical however since the adversary will be required to make strong non-collusion assumptions on the outsourced network. In particular, if any two machines belong to the same entity or collude, they can reconstruct the witness which the adversary needs to avoid.

Remark 5 (Usage of sequential functions). Our usage of hash-based PoW mining comes with some unfortunate consequences: since evaluation is embarrassingly parallel, we need to rely on assumptions on the parallel processing capabilities (e.g., number of machines) available to the adversary.

A natural question towards removing this constraint is whether we can leverage sequential computation (e.g., through VDFs) instead of parallelizable hash computations. This turns out to be somewhat tricky however since intermediate values may first be computed in a TEE following which the rest of the computation can be done in faster untrusted hardware. We leave the exploration of this direction to future work.

C.3 Parameterization Details

Estimating Q_{asic} and Q_{cpu}. A top-of-the-line mining ASIC for Bitcoin, the Antminer S19 Pro Hydro, released in 2022, has a rated performance of 154 TH/s [15], i.e., about $Q_{\text{asic}} \approx 2^{47}$ H/s. Each hash in this case is a "Bitcoin" hash: a double invocation of SHA-256 on two 64-byte input blocks, as required for Bitcoin mining. The set $\mathfrak{R}$ of satisfying resources for our formalism will therefore consist of all ASICs with a hash rate of at least Q_{asic}.

Even under optimistic assumptions (including use of native hardware support for SHA), an SGX application[12] on a state-of-the-art 4.60 GHz Intel processor can execute at most about 72 MH/s, i.e, $Q_{\text{cpu}} \approx 2^{26}$ H/s (where hashes here are "Bitcoin" hashes) [55].

Example practical parameterization. Given $Q_{\text{asic}} \approx 2^{47}$ H/s and $Q_{\text{cpu}} \approx 2^{26}$ H/s, we might for instance target a forced revelation error and completeness error of $\epsilon < 10^{-6}$. We set $y = \lfloor n/2 \rfloor$ for the most efficient parameterization. While, the error bounds are also achieved for much smaller value of τ, we set the duration of each round τ to 5 seconds, allowing for any network communication delays. Difficulty $d = 2^{47}$ and number of rounds $n = 8$ allow the ASIC to succeed with probability more than $1-\epsilon$ while an adversary using a network of 10,000 CPUs only succeeds with probability less than ϵ. Note that, the range of nonce (security parameter β) can be set appropriately to minimize p^{onechal}. For example, by Lemma 6.3, a nonce of length 5 bytes ($\beta = 2^{40}$) would let an adversary which uses only one challenge, succeed with probability $<< \epsilon$. Figure 5 shows the completeness probability of the ASIC and the success probability of an adversary with 10,000 CPUs as a function of the difficulty and number of rounds (each round is set to 5 secs).

With a total execution time of 50s, it is possible to achieve an overwhelming probability of completeness as well as an overwhelming probability of failure for an adversary trying to bypass forced revelation with a network of 10,000 CPUs.

Concrete ASIC-ZKPoCK **implementation details.** We implement the ASIC-ZKPoCK prover using an outmoded Antminer S9 ASIC and the verifier using an Ethereum Smart Contract. While this ASIC has a smaller hashrate compared to the latest hardware in the market, and the Ethereum network inherently has a coarse granularity for measuring time, we can still achieve reasonable error probabilities for completeness and forced revelation. Below is one such example parameterization for our implementation:

$Q_{\text{asic}} = 13\text{TH/sec} \sim 2^{43.5}$.

$Q_{\text{cpu}} = 2^{26}$.

$\tau = 12$ sec (Ethereum inter block time).

$\beta = 2^{40}$ (Set nonce size to 5 bytes). Notice that the nonce range is exhausted by our ASIC in $2^{40}/Q_{\text{asic}} = 0.08$ seconds, so we queue up new work to the ASIC (with a new challenge) every 0.08 seconds.

For difficulty $d = 7 \times 10^{13}, n = 12, y = 6$, we have completeness probability > 99.9% and forced revelation error $< 10^{-4}$ for a network of 10,000 CPUs.

D DEFERRED DETAILS ON SMACK

Global system architecture. SMACK consists of different verification contracts, one for each type of CK proof method that is supported. For a given public PoK value x, the prover supplies a proof (potentially interactively) according to a certain CK method to the corresponding verification contract. The verification contract stores a boolean mapping from x, indicating whether the proof has been successfully verified. This mapping is leveraged by the CK Registry to provide a uniform and well managed interface to application developers.

[12]Note that secure use of SGX requires disablement of hyperthreading.

CK Registry specification. In Ethereum, public keys are associated with *addresses*. SMACK supports CK for the private keys associated with Ethereum addresses.

CK proofs have an important property: Once one has been generated for a given witness / address, it remains *indefinitely valid*. That is because once a private key sk has been exposed to $\mathcal{E}$, the fact of exposure remains true for all time.

SMACK therefore includes a smart contract, called the *CK Registry*, that maintains a permanent record of addresses for which valid CK proofs have been provided.

The CK Registry (deployed at 0x25B270...eE3966) includes a function that maps Ethereum addresses to the type(s) of CK proofs, if any, that have been verified successfully. When a user wishes to submit a proof of complete knowledge for their Ethereum address, it sends the proof to a verification contract (such as the one we describe in Section 7.1) that the CK Registry trusts and then asks the CK Registry to record the event. Applications can then cheaply query the CK Registry to see whether a CK address is verified rather than handling proofs themselves. The CK Registry also allows for extra data to be associated with each verification event (such as attestation from and certificate of a TEE, or the security parameters of the ASIC-ZKPoCK protocol) so that upstream applications can utilize important information about the parameters of verification.

The CK Registry is designed to allow the addition of more CK verification contracts in the future, possibly incorporating new classes of CK proofs as they are designed, and likewise to remove existing CK verification contracts (in case the parameters are deemed insecure in the future). As a demonstration of our work, we have used the CK Registry and verification contracts to successfully mint Atomic NFTs for CK addresses. The Atomic NFT contract is deployed at 0x6986fc...FcAC38.

Setting the Nonce range β. For exploring the PoW puzzle solutions, the Bitcoin pool mining protocol allows the miners to try different values for the *nonce* and *extranonce* fields. The nonce field in Bitcoin header is fixed to 4 bytes and has proven to be too small for the mining market. Therefore, extranonce was introduced whose size can be set by the pool software. Thus, for our case where the range is $\beta (> 2^{32})$, we set extranonce to $\lfloor log_2[\beta]/8 - 4 \rfloor$ bytes. Note that B in the proof transcript π denotes the portion of the Bitcoin block header that excludes the extranonce field, i.e. we treat that field as part of the space of possible nonces.

D.1 Lightweight CK

A PoCK system will be most useful if it is widely accessible. Any requirement for expensive specialized hardware—such as an SGX-enabled machine or, worse still, a Bitcoin-mining ASIC—could place CK beyond the reach of most users. While users could in principle outsource CK-proof execution, this would require them to entrust their private keys to third-party services, which could create new risks of key compromise.

Increasingly many users, however, *do* in fact own devices with trusted hardware: their mobile phones. Almost all newly manufactured mobile phones come with TEEs: recent Android devices often ship with Trusty TEE, while iOS devices have Apple's Secure Enclave.

To show how these devices can be used to implement CK proofs, we prototype a protocol design that we call *lightweight CK*. The term "lightweight" here reflects two features of our design: (1) It uses common consumer hardware, but (2) embodies a weaker security model than CK variants using SGX or ASIC. As such, lightweight CK is most suitable as a defense-in-depth layer or for applications where the impact of compromise is not high—e.g., Atomic NFTs (see Section 2.2).

Android implementation and workflow. We design a simple lightweight CK tool for Android devices that uses the hardware key attestation API [11] to produce lightweight CK proofs for Ethereum addresses. This API provides a hardware-backed assurance of boot integrity and, by extension, an application's integrity. (Apple's iOS analog is its App Attest Service [12], which can support a CK tool like the one we've implemented for Android.)

Our application itself is simple from a user's standpoint: the user enters a private key sk—exported from a crypto wallet—into a text field[13] and taps a button to generate a TEE key. The application copies the necessary CK proof, described below, to the Android clipboard for the user to paste into a dApp that creates a transaction to a CK verifier smart contract.

The app creates an attestation challenge for the freshly generated TEE key through the key attestation API, and the TEE signs a new certificate containing the challenge. The attestation challenge contains the signature of a static message ("Android CK Verification"), including an Ethereum message prefix [16] signed by the user's private key. While not strictly needed for a CK proof, the signature serves as a hedge against device compromise. Even if a compromised Android device could generate a seemingly valid integrity verdict, it cannot do so for an arbitrary pk—cooperation from the holder of the corresponding sk would be required.

The API then returns a certificate chain from the new TEE key containing the challenge to the TEE itself to the device manufacturer's certificate authority and, finally, to a root of trust—the Google Hardware Attestation Root Certificate [7]. Within the new TEE key's certificate is an attestation to the integrity of the operating system running on the device as well as a hash of the signing certificates of the application that made the request.

The certificate chain includes all the necessary information to create a CK proof, so submitting a lightweight CK proof to the Android CK Verifier smart contract involves creating a transaction that includes the intended prover address and the complete certificate chain to some root of trust. To ensure the authenticity of a lightweight proof, the contract first checks that the newly created certificate describes an adequately protected Android operating system: a verified boot from a trusted state and a key attestation for an app that matches the package name and signing key of our app. It also checks the attestation challenge embedded within the certificate to verify that sk signed the message. Next, it validates the certificate chain to a root certificate that the verifier trusts by verifying the signatures of each certificate, each one signed by the next in the chain. If everything passes validation, a record of the proof is created in the contract for use by the CK Registry. The cost of verification is approximately 1.5 million gas per certificate.

[13]Great care will be required in guiding users, as malicious software could dupe unsophisticated users into revealing their private keys. This can be mitigated if common wallets natively implement lightweight CK.

Signed messages cannot protect against a compromised device producing valid verdicts for others' keys, so we rely on per-device limits to mitigate the effect of a compromised device. Until mobile devices support on-device attestations, integrity measurements of the operating system are the closest way of verifying that the application ran as intended and a complete private key was entered into the device.

Limitations. Key attestation appears not to be foolproof. For example, there have been reports of a broken TEE keystore implementation in the ASUS ROG 3, compromising system integrity and still allowing a "strong" hardware-based integrity check to pass irrespective of bootloader status [2]. Google itself recommends a defense-in-depth approach, with attestation services as only one of several signals of abuse. To mitigate the problems caused by an entire class of devices containing faulty TEEs, the Android CK Verifier contract allows individual CA certificates to be revoked or trusted.

Privacy. Publishing a complete TEE certificate chain to a public blockchain comes with its own privacy issues. Each TEE certificate must be signed by a device manufacturer's public key for the certificate chain to be complete, which means that device manufacturers could easily associate the Ethereum address of a lightweight CK participant with the mobile device used to create the TEE certificate chain. This is because the Ethereum address being verified is contained within the TEE-signed certificate, and device manufacturers can associate a TEE's public key with its corresponding device both during manufacturing and whenever the device requires an updated certificate chain. As a point of reference, the first intermediate certificate of our sample device expires on a monthly basis, so at least once a month, the device must contact its manufacturer for a fresh intermediate certificate.

In order to prevent this type of privacy leak, rather than submitting the certificate chain directly to a smart contract, the certificate chain could instead be submitted to an application running within an off-device attesting TEE, such as Intel SGX. The attesting TEE application verifies the most sensitive part of the chain—including the mobile device TEE's public key—and the integrity state of the mobile device embedded inside the leaf certificate. The smart contract would then only need to verify the application's attestation to establish whether the lightweight CK attempt was successful, thereby keeping the end of the certificate chain from being disclosed. Then, collusion between both the mobile device manufacturer and the organization hosting the off-device TEE, as well as a feasible attack on the off-device TEE itself, would be necessary to deanonymize accounts. An attack on the TEE could reveal the tail of the certificate chain which, when revealed to the device manufacturer, could be mapped to a device.

In an alternative to the TEE-based privacy approach, the prover's CK proof could be a redacted certificate chain along with a ZK proof (preferably a *succinct* such proof) of the correctness of the redacted part of the chain.

Target applications. Google's Play Integrity API is currently the more commonly used method for developing assurance of application integrity inside Android apps, and its "strong" category of integrity verdicts include similar hardware-backed key attestations from a TEE. Although its integrity verdicts can range in strength from basic compatibility tests to hardware-backed key attestation, it has seen wide use by consumer services (e.g., Netflix), mobile games (e.g., Pokémon Go), banking applications, etc. for application integrity [47].

Given the security limitations discussed above, lightweight CK is most suitable as a defense-in-depth layer or where the impact of compromise is limited. For example, if CK is compromised for an Atomic NFT, that NFT can be fractionalized: an undesirable but not catastrophic outcome. The same is true of key-coupling for royalty payments. In contrast, only strong CK would meet the levels of security envisioned for soulbound tokens [68], which are identity documents.

Through the CK Registry, individual applications can support the specific CK proof types that match their security models.