ZKASP: ZKP-based Attestation of Software Possession for Measuring Instruments

Luís T. A. N. Brandão[1], **Carlos E. C. Galhardo**[2], **René Peralta**[3]

[1]ORCID: 0000-0002-4501-089X. National Institute of Standards and Technology (NIST), USA, as a Strativia contractor.

[2]ORCID: 0000-0002-7398-8182. National Institute of Metrology, Quality and Technology (INMETRO), Brazil.

[3]ORCID: 0000-0002-2318-7563. National Institute of Standards and Technology (NIST), USA.

This is the author's version of the article submitted to Measurement Science and Technology, Special Section on the 20th International Congress of Metrology (CIM 2021). Submitted: 2021-August-20; Revised: 2022-March-01.

With permission, an earlier version with the same title appeared, with differences, in the International Organization of Legal Metrology (OIML) Bulletin Vol. 62, No. 3, July 2021.

Abstract

Software-controlled measuring instruments used in commercial transactions, such as fuel dispensers and smart meters, are sometimes subject to "memory replacement" attacks. Cybercriminals replace the approved software by a malicious one that then tampers with measurement results, inflicting a financial loss to customers and companies. To mitigate such attacks, legal metrology systems often require regular device attestation, where an auditor checks that the device possesses ("knows") the approved software. However, current attestation methods usually require the software to be known by the auditor, thus increasing the risk of inadvertent leakage or malicious theft of proprietary information, besides facilitating its malicious adulteration. We describe how this issue can be addressed in legal metrology systems by using **z**ero-**k**nowledge **p**roofs **o**f **k**nowledge (ZKPoK). These proofs enable attestation of possession of approved software, while ensuring its confidentiality from the auditor. To further provide publicly verifiable evidence of freshness, each such proof can be related to a fresh random value from a public randomness beacon. This article presents the basic conceptual idea, while also discussing pitfalls that should be avoided.

Keywords: device attestation, legal metrology, proof of knowledge, public auditability, randomness beacon, zero-knowledge proof.

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is neither intended to imply recommendation or endorsement by INMETRO or NIST, nor to imply that they are necessarily the best available for the purpose.

1. Introduction

1.1. Measuring instruments and legal metrology

In modern society, measuring instruments are a cornerstone of many activity sectors, including trade, safety, environment and health. For example, when filling an automobile fuel tank, a customer trusts that the volume of fuel dispensed by the pump corresponds to the displayed measurement result. Given the importance of measurements in many economic activities, each country develops a "legal metrology" framework composed of laws and regulations that strive to ensure that measurements are accurate (agree with corresponding standard units), reliable (stable against environmental changes), and incorruptible (impervious to malicious manipulation).

While scientific metrology is the science of measurement, legal metrology can be defined as the "practice and process of applying a regulatory structure and enforcement to metrology" [OIM12]. Laws concerning measuring instruments are needed when measurement errors or fraud can affect commercial transactions, public safety, health-related decisions, or the environment [Kel19]. Each country has at least one national metrological authority [OIM13] to enforce the metrological regulations and execute legal metrology procedures. (European harmonized standards rely on Notified Bodies [Gal13].)

In legal metrology, the accuracy requirement for computational systems calibrated for measurements should consider that they may be deployed in settings susceptible to adversarial attacks. Therefore, these systems should consider security properties such as integrity and authenticity [MANSV20]. In fact, digital/computer security has been proposed as one of the next-generation

"key enabling technologies" for Industry [PBOE+21] and has indeed played an increasing critical role in Industry [CLL20]. Correspondingly, its consideration is also required for improved legal metrology.

Each legal metrology authority enacts, through regulations, a set of software security requirements to protect from and detect non-authorized modifications in the software of measuring instruments. Then, for an embedded software to be approved by the metrological authority, it must comply with these requirements [PBMC+14]. The **integrity verification of the approved software** is one of the most critical concerns of software security regulation requirements [PPST15]. In legal metrology, measuring instruments must prove their software's integrity to a customer or an official auditor. The integrity check is typically obtained through a device attestation protocol.

1.2. Device attestation

A "device attestation" protocol is a technique that allows a software-controlled device to make a reliable statement about its memory content. The device attestation has two participants: the *verifier* $\mathcal{V}$ (the auditor) and the *prover* $\mathcal{P}$ (the instrument). The main goal is to enable $\mathcal{V}$ to confirm that $\mathcal{P}$ has some embedded software expected by $\mathcal{V}$. This is an essential component of metrological verification [OIM13].

Device attestation is, in general, implemented as a challenge-response protocol [CGLH+11]. Figure 1 shows the five main steps: (1) $\mathcal{V}$ prepares a challenge for $\mathcal{P}$, and (2) sends it to $\mathcal{P}$; (3) $\mathcal{P}$ computes a response and (4) sends it to $\mathcal{V}$; (5) $\mathcal{V}$ checks that the response is valid with respect to the challenge.

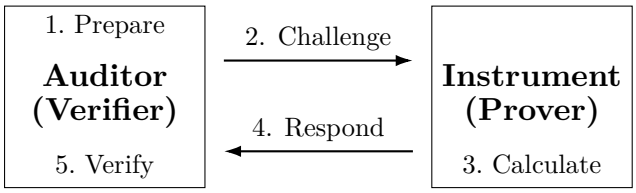

Figure 1. Device attestation

Often, the response is the output of a hash function that uses the memory content and the challenge. Using a hash allows succinctness, while still convincing that the challenge was used. In other words, the response can only be satisfied with access to the declared memory content after the challenge was made known. If $\mathcal{P}$ is an honest prover (i.e., it acts as specified by the protocol, and its memory content satisfies the statement being proven), it convinces the verifier about memory content. If $\mathcal{P}$ lacks the required memory content, then it should not be able to convince $\mathcal{V}$.

A device attestation protocol should reflect the current status of the prover. This property, called *freshness*, is ensured by including a *challenge* (information that the prover does not know in advance), which the prover requires in order to produce the *response*. Several approaches have been proposed to ensure freshness, such as based on random number [SPVK04], pseudo-random sequence [CFPS09], increasing integer [ISZ17], session keys [KBGK17], or timestamps [ERT17]. The approach proposed in this paper uses a public randomness beacon for this purpose.

A randomness beacon is a time-stamped source of public randomness [KBPB19]. At regular time intervals, it publishes a *pulse*, containing a fresh sequence of random bits. The pulse also contains a digital signature and various other fields. The pulses are also hash-chained. A beacon needs to have three critical properties: unpredictability, autonomy, and consistency [FIP11]. *Unpredictability* means no one can predict the random bits of new pulses. *Autonomy* means no one can influence the probability distribution of the output bits. *Consistency* means that all users accessing a particular pulse have the same view, namely the same random string. The pulses are archived forever and can be retrieved by anyone, using a well-defined interface (e.g., via the Internet).

A proof that is bound to the randomness of a pulse becomes bound to its timestamp, implying that the proof cannot have been created before the pulse was publicly known. The time interval between (i) the present moment and (ii) the timestamp of that beacon randomness can be used as a quantitative measure of freshness. This freshness is publicly verifiable, since each timestamped pulse from the beacon is signed and hash-chained in a publicly-readable database. Furthermore, a user can query the database to retrieve any previous pulse and its associated data.

Legal metrology tries to mitigate the malicious software replacement attack by requiring regular device attestation (software integrity verification), where an auditor checks that the device possesses ("knows") the original software by comparing it to a copy held by the auditor. However, if the attestation requires the auditor to view the software, there is a major drawback to confidentiality. The protection of the manufacturer's intellectual property is jeopardized, since the attack surface to steal the software has increased beyond the instrument. Also, this is inconvenient in practice since it narrows the set of possible auditors to only include auditors that are trusted to handle confidential material.

1.3. Enhanced security

State-of-the-art practical cryptography offers the opportunity for legal metrology frameworks to enable attestation of software possession with refined security properties. In particular, it is possible to enable a dual requirement of confidentially and auditability. For example, prior work has considered the case of confidentiality of measurements, sometimes in systems with many sensors (such as smart-grids), using cryptographic build-

ing blocks, such as homomorphic encryption and even functional encryption, to enable auditability of software functionality with respect to the combination and transformation of many measurement results [PYMS+20].

The present work is focused on confidentiality of the embedded software, for independent individual measurements, without concern for confidentiality of the measurement results. By using a **z**ero-**k**nowledge **p**roof (ZKP), more specifically a **ZKP o**f **k**nowledge (ZKPoK), an auditor unacquainted with the approved software can still check whether it is known by a measuring instrument. The proposed solution approach is dubbed ZKASP: **ZK**P-based **a**ttestation of **s**oftware **p**ossession for measuring instruments.

In the proposed ZKASP solution, the assurance of confidentiality becomes technical/cryptographic, instead of depending on a non-disclosure agreement by the auditor. Besides the official auditors, the customers can also run the protocol before a commercial transaction, to ensure that the instruments are able to prove knowledge of the correct software.

The proposed solution also enables freshness based on a "nonce", which is obtained from a public randomness beacon. The timestamped nonce creates an upper bound for the age of the proof, which can be publicly verified. This solution also prevents replay attacks, where a valid proof is maliciously repeated. Current public randomness beacons, such as the NIST one, have a pulsation period of one minute.

While the proposed solution is conceptually simple, it is important to avoid a number of pitfalls. For example, while the generation of a hash of the software requires knowledge of the software, a proof of knowledge of the said hash is not a proof of knowledge of the software. Thus, the ZKASP solution should simultaneously be succinct and require access to the full software. Different mechanisms are possible depending on the legal metrology system model and the allowed interaction between various parties.

2. System model and building blocks

2.1. Participants

For a given type of measuring instrument, there are various parties in the system of interest:

- **Measuring instruments (a.k.a. provers):** the individual instruments that are required to provide a proof of knowledge of the approved software.
- **Auditors (a.k.a. verifiers):** those that interact with the measuring instrument to verify that the instrument "knows" the approved software. Usually, these are official *auditors*, but in the proposed solution they can also be regular *customers*.
- **The authority:** a legal metrology authority (or notified body) providing public parameters that enable the device attestation protocol.
- **The randomness beacon:** an agreed trusted public randomness source [BeaconUS; BeaconBR; BeaconCL] that periodically outputs signed timestamped random values.
- **The vendor/manufacturer:** the entity that is responsible for setting (and possibly updating) the embedded software in the measuring instruments.

2.2. Building blocks

ZKASP uses the following cryptographic primitives:

- **Digital signature [FIPS 186-5].** Any party can create a pair (k, K) of keys: one private (k) for signing; one public (K) for verification. Given any string M, the party can use its private key to compute a signature $\sigma \leftarrow \text{Sign}[k](M)$. Given M and σ, any party with the public key can verify authenticity: $\text{VerSign}[K](\sigma, M) = \texttt{true}$. However, without the private key it is unfeasible for any party to compute σ that would pass the verification.
- **Hash function [FIPS 180-4].** A hash function H takes as input any string M, of arbitrary length, and outputs a (short) string $h_M = H(M)$. Any party can compute H. Given h, it is unfeasible for any party to find an M such that $h = H(M)$. Given M, it is unfeasible for any party to find another M' such that $H(M') = H(M)$. Furthermore (informally), $H(M)$ reveals no information about M (when that M is unpredictable).
- **ZKPoK [ZKProof]:** A ZKPoK (*zero-knowledge proof of knowledge*) is a cryptographic method by which a party (the prover) proves to another party (the verifier) that it knows a secret value x, but does not disclose x itself. For example, the prover may know the prime factorization of a large integer (e.g., used in some crypto-systems) and wish to prove this to a verifier, without revealing the prime factors. The method ensures that someone without the secret cannot successfully act as a prover.

ZKASP also uses various components of the infrastructure of the Internet, such as secure communication [RFC 8446] and public-key infrastructure [RFC 5280]. As for the public randomness server, our proposal is to use any server that is interoperable with the NIST Beacon. This server provides, every minute, a timestamped and cryptographically signed 512-bit random string. The signature authenticating these random strings can be verified off-line using the server's public key.

2.3. ZKPoK bound to time and identities

The type of ZKPoK we will use — let us call it Π — for proving knowledge of a secret S, with a certain property T (i.e., $T(S) = \texttt{true}$), is of the following basic form:

- The verifier sends an external challenge C to the prover. This external challenge includes a timestamp, identifiers, and fresh random values. (We

call it "external" to avoid confusion with a common [internal] "challenge" step in the generation of Π.)
- The prover replies with a value $\pi \leftarrow \Pi(C, S)$ that could only have been produced by someone who knows secret S satisfying T, and who simultaneously knows C.
- Given π and C, the verifier can efficiently compute $\text{VerProof}(\pi, C)$, which returns `true` if and only if the proof is valid, i.e., satisfies the two conditions expressed above.

Section 3.2 describes our solution, which amounts to designing the external challenge C and choosing a ZKPoK Π so as to ensure zero-knowledge, eliminate some attacks, and mitigate others.

It should not be possible to reuse the proof constructed by one instrument in order to produce a proof for a different instrument. It should also not be possible to use an old proof to construct a fresh proof, even if by the same instrument. More generally, a good solution must ensure that proofs are bound to the context in which they were produced.

To ensure this, we devise the external challenge C to be dependent on various strings that, together, specify the context. For example, if using $C = (t, \mu_t, \text{ID}_P, \text{ID}_V)$, where μ_t is a random string produced at time t by a public source of randomness, the proof $\pi \leftarrow \Pi(C, S)$ becomes bound to the time t and to the prover and verifier identities (ID_P, ID_V).

3. ZKASP protocols

3.1. Setup (initial deployment)

The typical setting for a ZKASP protocol is as follows:

- **Cryptographic keys.** All parties have private and public keys for a signature scheme. The j-th auditor's private/public key pair is (y_j, Y_j). The i-th instrument's key pair is (x_i, X_i). The authority's and vendor's key pairs are, respectively, (a, A) and (v, V). The public key of each party is also used as its identity.
- **Software approval.** Through an agreed protocol, during the type approval [OIM13], the authority approves the software S developed by a vendor for use with a type of measuring instruments.
- **Software commitment.** At time t_0, the authority publishes the hash $h = H(S)$ of the approved software S, along with a signature $\sigma_A = \text{Sign}[a](t_0 || r_{t_0} || h)$ by the authority, and a corresponding signature $\sigma_V = \text{Sign}[v](t_0 || r_{t_0} || h)$ by the vendor, where r_{t_0} is the most recent beacon randomness (from time t_0). On purpose, the signing of the same element, by the authority and by the vendor, requires coordination between the two.
- **Software deployment.** The vendor deploys the measuring instruments, each with a private-public key pair (x_i, X_i). Only the instrument knows its own secret key; the authority and the vendor both have a list of the public keys of all instruments.

3.2. The baseline ZKASP Protocol

Next we provide a textual description of an attestation interaction, with some simplifications. Some details are deferred to Figure 2 in Appendix A.

In an attestation interaction, started at time t, between an auditor j and an instrument i:

1. **External challenge.** The auditor obtains the most recent pulse B_t from the public randomness **b**eacon, verifies its consistency (e.g., that it is properly signed), and extracts its output random value r_t and the corresponding timestamp t. Then, the auditor determines the current time t' in its own local clock and locally-generates a random value r'_t. The auditor composes these elements, along with the public key X_i of the instrument to be audited, and signs them together, obtaining $\sigma_j = \text{Sign}[y_j]\,((t, r_t, t', r'_t, X_i))$ and then sends $C = (t, r_t, t', r'_t, X_i, Y_j, \sigma_j)$ to the instrument.
2. **Response.** The instrument checks that the received X_i (inside C) is indeed the instrument's public key. Then it checks the validity of the auditor's signature σ_j, with respect to the public key Y_j. An actual implementation needs to perform various other verifications, such as checking that the pair of timestamps is acceptable: well ordered, and consistent with a trusted local clock (if available). If any verification fails, then the instrument aborts the interaction. Otherwise it produces the proof $\pi = \Pi(C, S)$, i.e., a C_t-bound ZKPoK of "S satisfying $H(S) = h$". Finally, the instrument signs the proof, obtaining $\text{Sign}[x_i](\pi)$, and sends (π, σ_i) to the auditor.
3. **Local verification and transfer.** The auditor checks the validity of (π, σ_i), i.e., that $\text{VerProof}[h](\pi, C) =$ `true` and $\text{VerSign}[X_i](\sigma_i, \pi) =$ `true`. Note that this implies that the proof was produced after the randomness values r_t and r'_t were generated. The auditor then transfers the tuple (C, π, σ_i) to the authority.
4. **Central verification.** The authority makes the necessary checks, including that the proof is valid and is bound to the appropriate context — appropriate timestamps, requested by auditor j, and produced by instrument i after the beacon value r_t was generated. The authority then stores the proof and associated context. Conceivably, this can later be publicly audited.

The described protocol is agnostic to the ZKPoK technique used to prove knowledge of a hash pre-image. This may be based on a general-purpose ZKPoK technique to prove knowledge of the input of a function (a hash, in this case), which yields the output h. The details are beyond the scope of this paper. However, we note that

computing such ZKPoK entails not only every step of the computation of $h = H(S)$, which processes every bit of S, but also a multiplicative overhead of the ZKP to prove that each of those steps was correctly performed.

3.3. A lightweight ZKASP protocol

It is conceivable that low-resource devices may be unable to efficiently perform the generic ZKPoK described in Section 3.2. For those cases, we describe a much more efficient approach, essentially based on a simple discrete-log ZKPoK (a Schnorr proof [Sch91]). The tradeoff, as compared with the previous solution, is that it requires a more active role by the authority (or the vendor) and a corresponding synchronization by the auditor.

Elliptic curve parameters. The system uses global **e**lliptic **c**urve parameters [SP 800-186] agreed by every party. These include a cyclic group G_q of order q and generator G, e.g., based on `Curve25519` [RFC 7748; Ber06], which can be the same as already used for signatures.

Frequent fresh commitments. For this lightweight solution, we assume that a trusted party with knowledge of the software S periodically obtains the most recent public randomness pulse B_τ, with random output value ρ and timestamp τ, uses them to compute a secret value $h_\tau = H(\rho||S)$, and then commits to it by publishing $Q_\tau = h_\tau \cdot G$. The operation $\cdot$ represents a multiplication in the elliptic curve. Therefore, Q_τ is simply a point on `Curve25519`, which can be represented by a 256-bit integer. Under standard cryptographic assumptions, the publication of Q_τ does not reveal h_τ.

Since Q_τ changes frequently (the frequency can be as high as the beacon period allows, e.g., once per minute), the knowledge of $h_\tau = H(\rho||S)$ can be used as a proxy for the knowledge of S. However, note that the period for producing new values Q_τ may be larger than the beacon pulsating period. To prove knowledge of h_τ, the following (Schnorr-based) protocol suffices.

Auditor-Instrument interaction:

1. **External challenge.** In comparison with the baseline protocol described in Section 3.2, the external challenge prepared by the auditor (verifier) has two additional elements: τ, ρ. The challenge becomes $C = (\tau, \rho, t, r_t, t', r'_t, X_i, Y_j, \sigma_j)$, where $\sigma_j = \text{Sign}[y_j]\,((\tau, \rho, t, r_t, t', r'_t, X_i))$. Recall that t and t' are the timestamps of the beacon and the auditor.
2. **Response.** The instrument (prover) makes the necessary checks, including that the embedded identifiers and the signature σ_j are valid, and that the triplet of timestamps (τ, t, t') is acceptable. If any check fails, then the instrument aborts. Otherwise, it computes the hash $h_\tau = H(\rho||S)$ and its commitment $Q_\tau = h_\tau \cdot G$. Then, the instrument produces the proof π (a ZKPoK of the discrete-log of Q_τ):
   (a) selects a random number u and computes its "commitment" $U = u \cdot G$;
   (b) computes the internal "challenge" $c = H\,(C||U)$
   (c) computes the "answer" $z = u + c \cdot h_\tau \pmod q$;
   The prover defines $\pi = (U, z)$ and signs it $\sigma = \text{Sign}[x_i](\pi)$ and then sends (π, σ_i) to the verifier.
3. **Local verification and transfer.** The verifier accepts the proof if and only if $z \cdot G = U + c \cdot Q_\tau$, where c is computed as also prescribed in step 2(b) for the instrument. The transfer to a central authority and the central verification follow the same logic as in the baseline protocol.

Note that U and $(c \cdot Q_\tau)$ are points in the elliptic curve, and the last addition is an elliptic curve operation. This is known as a Schnorr proof [Sch91] and, under standard cryptographic assumptions, securely demonstrates knowledge of h_τ, the discrete-log of Q_τ. By proxy, assuming that the instrument did not receive the value h_τ from an adversary, it follows that the instrument must have known the entire software S to learn h_τ.

3.4. Clarifications for implementation

Next, we clarify some aspects of the presented protocols.

Timestamps. The described protocols include various timestamps. Ideally these would be very close to each other, but it is possible to have some discrepancies, namely if the period for generating new commitments Q_τ is larger than the beacon's period. One advantage of including the timestamp t' from the auditor's clock is that it constitutes an assertion of the auditor's responsibility. One advantage of including the timestamp(s) from the beacon pulse(s) (whose randomness is/are used) is that it facilitates verifying whether the time discrepancies (namely with the auditor's timestamp) are acceptable (according to some chosen rule). A concrete implementation can also decide to bind to the proof a timestamp from the instrument's local clock (if/when available).

Knowledge of the instrument's public key. The description assumes that the auditor knows in advance the instrument's public key X_i. This does not have to be the case. If unknown at start, X_i can be discovered through a standard handshake process or even from a physically stamped code on the instrument. This may be useful for the case of a member of the public acting as auditor.

External challenge. It is important to note that the external challenge C_t is not (and does not replace) what is sometimes called the [internal] "challenge" of a ZKPoK. In the presented protocols, we are prepending the external-challenge message to what can then be a non-interactive ZKPoK, whose internal challenge component is computed (by the instrument) using the Fiat-Shamir heuristic [FS87].

4. Security considerations

4.1. Basic properties

Important security properties for the legal metrology system stem essentially from the underlying building blocks: ZKPoK and beacon randomness. Informally:

- **Completeness.** If every party is honest, then the system leads the auditors to obtain valid proofs from instruments. These proofs can later be validated by the authority and even verified by the public.
- **Soundness.** A party without access to the software cannot produce a signature. This stems from the "extractability" property of the ZKPoK, i.e., that producing a valid proof implies being able to write down the full software string. Furthermore, the use of a signature also implies access to the secret key of the instrument names in the proof. In practice, we are satisfied with computational soundness (as is the case already with signatures), meaning that it relies on cryptographic assumptions (e.g., intractability of computing a discrete logarithm in some mathematical group).
- **Zero-knowledge (with transferability).** By definition, the ZKPoK does not reveal any information (in a cryptographic sense) about the software. However, it is noteworthy that the use of the Fiat-Shamir technique (for a non-interactive proof) makes the proof transferable, meaning that one gains the ability to prove that someone has knowledge of the software. This is an intentional feature. The digital signature also corresponds to a transferable proof of access to (i.e., of knowledge of) the private signing key that corresponds to the public verification key (the instrument's public key) that was bound to the proof of knowledge of software.
- **Verifiable freshness.** The use of public randomness from a randomness beacon binds the proof to a public timestamped value trusted to be unpredictable before the timestamp. This means that no one could have generated the proof before said timestamps. This enables placing an upper bound on the age of a proof.

The use of a digital signature scheme is crucial for ZKASP, to provide authenticity of the author of a proof. In turn, this prevents proxy attacks. Naturally, this relies on a trust model about public keys, which can be supported on a public key infrastructure and blockchains [MMACR19; MMPM20; PWTS18]. Furthermore, the legal metrology framework itself should promote a system of transparency, which may, for example, include making publicly accessible a list of the public keys of all measuring instruments.

A formal security analysis should consider an idealization of security, such as in the ideal/real simulation paradigm, e.g., in the universal composability (UC) framework [Can01].

4.2. Adversaries without the approved software

The following paragraphs discuss what can(not) be achieved by an adversary without the software, with respect to forging proofs. The considerations are intended as clarifying comments to convey intuition about security properties. However, they are not a proof of security. Concrete protocols following the ZKASP approach (i.e., building on fresh, transferable ZKPs of software possession) should be accompanied by a specific formulation of the properties intended of the system model (possibly derived from a so-called ideal model) and how they are achieved by the proposed concrete protocol.

Goal: A malicious instrument, without the valid software, wants to produce a valid proof of correct software possession. For example, either an adversary has replaced the instrument's software, or the instrument has not undergone a mandatory software update.

Capabilities: The adversary (e.g., a malicious auditor) has access to honest instruments and is able to request valid proofs from them, being able to completely determine the used challenges. However, the adversary is assumed to not be able to exfiltrate the private signing key or the approved software from deployed measuring instruments.

Impracticable attacks: ZKASP inhibits the well-known proxy attacks and precomputing attacks [SL16] against device attestation protocols.

- **Pre-computing attack (trying to reuse old proofs):** A malicious auditor is able to interact with an honest instrument and thereby obtain many proofs. In an unprotected system, the auditor could later try to reuse the pre-computed proofs, when, in fact, the instrument would already have been corrupted and no longer possess the correct software. ZKASP prevents this forgery of younger proof ages, because of its binding to timestamped (unpredictable) beacon randomness. If at a time t (with precision in minutes) the instrument loses access to the software, from that moment onward it will not be able to produce a proof with an acceptable claim that it was produced after time t.
- **Proxy attack (trying to use proofs from others):** In unprotected attestation protocols, the adversary may try to act as a malicious instrument (without the approved software), evading the attestation by redirecting the challenge to a nearby honest instrument, acting as a proxy [PSLP12]. This is not possible in ZKASP, since the proofs of software possession (ZKPoK) produced by other honest instruments (with access to the software) are bound to the instrument's identity, and will therefore not be valid if claimed by other instruments.

4.3. Adversaries with the approved software

It is important to bear in mind that the ZKASP approach proposes a proof of possession, which is not equivalent to a proof about the entire memory of the instrument. For example, an instrument capable of retaining the correct software and additional malicious code will still be an instance of possession of the correct software.

Possible attacks in which the adversary possesses the approved software are out of scope for resolution in this paper. Yet, it is instructive to consider their possibility.

- **Compression attack:** the adversary compresses the approved software and then adds a small malicious code in the instrument's memory [CFPS09].
- **Time-of-check to time-of-use (TOCTOU) attack:** a malicious instrument can download the approved software just in time for attestation, but all other times use a malicious software [NJRT20].
- **Collusion attack:** two malicious instruments have complementary pieces of the approved software; together they can reconstruct the entire approved software [YWZC07].
- **Proofs-as-a-service (PaaS):** an adversary that knows the approved software can create a proof that is bound to the public key of an instrument that does not possess the software; then, this adversary could provide (e.g., sell) this forged proof to a corrupted instrument that would simply sign the proof, thereby obtaining the two elements (π, σ) needed to trick the authority into believing that a valid attestation has been performed.

Despite the above attacks, the attestation provided by ZKASP addresses an important problem. It prevents malicious instruments from posing as honest, when oblivious of the confidential correct software and unaided by an adversary knowledgeable of the software.

Possible attacks in which the adversary possesses the approved software do not break the semantics of the proof, although it shows one limitation of the ZKASP approach. This case is out of scope of resolution in this paper, but for concrete schemes it is worth considering what complementing techniques may be possible to implement. For example:

- **System restrictions.** The compression attack can conceivably be prevented by a non-compressing software string that fills up the entire memory. The TOCTOU and collusion attacks may be mitigatable by hardware restricting the interaction capabilities of the instrument. While these are conceivable mitigations, ensuring the mentioned restrictions is not trivial in practice.
- **More sophisticated ZKPoK.** The described PaaS attack requires manipulating the instrument to sign a forged proof. Conceivably, this can be mitigated by a more sophisticated ZKPoK that would be verifiably bound also to the private key of the instrument, rather than only to the public key, in a way that prevents the malicious auditor from building the proof alone. In other words, the idea would be to require a much higher interaction between the holder of the key and the holder of the software. A proof of software possession would still be possible via an interactive secure computation between the instrument and the malicious auditor, each with the corresponding secret (private key and software, respectively). While this would not eliminate the possibility of the attack (after all, the pair of colluding parties knows the entire secret needed to produce the proof), it would increase the practical difficulty / deterrent for collusion.

5. Discussion

ZKASP is an approach to attestation of software possession by measuring instruments in a legal metrology framework. The approach enables public auditability and allows attestation to auditors not in possession of the software. The approach makes essential use of a ZKPoK in combination with a public randomness server.

5.1. Foreseen ZKASP deployment

An adoption of the ZKASP approach is expected to be suitable for deployment in instruments with micro-controllers running without an operating system. These micro-controllers, used in "built-for-purpose measuring instruments" — a type P computer, according to the European Cooperation in Legal Metrology (WELMEC) Guide 7.2 [WEL20] — would be chosen to support a chosen ZKPoK. Instruments such as fuel dispensers (gas pumps), utility meters, grain moisture meters, and non-automatic weighing instruments (grocery store scales) are examples of measuring instruments that could benefit from ZKASP. It is worth noting that ZKASP is a hybrid device attestation protocol [SL16] that depends on software and specialized hardware to secure the instrument's secret signing key. (General-purpose computers running with an operating system are more flexible for other attestation approaches [PPST15].)

Concrete implementations are beyond the scope of this paper but, even in settings where the general ZKASP approach (a direct ZKPoK of a pre-image of the hash of a long software) may be too expensive (say, due to low resources of the micro-controller), it is possible to consider the described lightweight version, based on a very efficient Schnorr proof (Section 3.3), albeit requiring a more active role by the authority and synchronization by the auditor.

5.2. Expectations about ZKASP

It is important to differentiate between what ZKASP solves and what it does not. To start, a proof of knowledge/possession of the correct software does not guarantee absence of an additional incorrect software. So,

in some cases an attacker may be able to subvert a measuring device by injecting code that controls the operation, and that has access to the correct software, in order to generate a valid proof of knowledge when requested. For example, it is conceivable that a malicious code injection would make the display screen always show an amount increased by a constant factor, in comparison with the correct measurement.

Despite the above, ZKASP at least impedes the generation of a valid proof by attackers that do not "know" (or do not have access to) the correct software. Therefore, if a client checks the proof as a condition of paying, then it will not be victimized by a device that does not possess the correct software. Also, if a physical system is able to guarantee that the software that executes in connection with a measuring operation is the software about which a proof of knowledge is performed, then the proof does provide additional assurance of which software has operated. (It may still be significantly difficult to guarantee that the software did not change between the check and the use.) Furthermore, one can also consider having the software (the one being attested) digitally signing the measuring results, so that the later signature verification (say, by the customer, using a personally-owned mobile device) serves as an additional check before deciding "what to pay". While such complementary procedures can serve as additional deterrents against corrupted measurement results, the remainder of the paper is focused only on the actual ZKASP proposal.

5.3. *ZKASP for a digital transformation of legal metrology*

The European Union is under a digital transformation of the legal metrology infrastructure, known as The European Metrology Cloud [Thi18]. In the Metrology Cloud, the manipulation of sensors became an important attack vector [OETS18]. ZKASP can mitigate this attack vector, by having the computer play the role of verifier. It can perform the attestation of several sensors in its neighborhood area network [BSK10] and report the results to the Metrology Cloud. It can also be applied to prove proper updating of software.

The ZKASP approach can also be used to empower customers as verifiers. The authority can enable a system (e.g., a mobile application) that allows customers to communicate with instruments, run the device attestation, and report proofs. The authority can crowdsource the collected data to better direct the placement of market surveillance operations. In this scenario, the approved software should enforce a period between subsequent attestations to prevent denial of service attacks.

6. Conclusions

This paper presented ZKASP, an approach for ZKP-based attestation of software possession, for measuring instruments in a legal metrology context. It combines a ZKPoK protocol, as well as randomness from a public beacon, to address challenges of device attestation. The ZKPoK allows the verifier (auditor) to remain oblivious of the content of the software embedded in the audited instrument. The use of beacon randomness establishes a publicly verifiable upper bound on the age of each proof.

ZKASP is a proposal where cryptography meets metrology. It combines a privacy-enhancing cryptographic tool (ZKPoK) and verifiable randomness to enhance the security guarantees of a real use-case requirement (attestation of software possession) of legal metrology. Being a high-level proposal, there are various aspects that deserve a closer look.

The choice of concrete instantiation options can depend on the exact context, e.g., the computational power of the processing unit (often micro-controllers) of the measuring instruments. Interesting options concern the choice of hash function, which may vary between the widely recognized secure hash algorithm (SHA) family standard and other more-recent proposals of ZKP-friendly hashes.

Acknowledgments. We thank Michael Davidson and Irena Bojanova from NIST and the anonymous reviewer from the Measurement Science and Technology journal for their useful editorial comments.

References

[**Ber06**] D. J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography — PKC 2006*. Ed. by M. Yung, Y. Dodis, A. Kiayias, and T. Malkin. Springer Berlin Heidelberg, 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[**BSK10**] R. Berthier, W. H. Sanders, and H. Khurana. "Intrusion detection for advanced metering infrastructures: Requirements and architectural directions". In: *2010 First IEEE International Conference on Smart Grid Communications*. IEEE. 2010, pp. 350–355. DOI: 10.1109/SMARTGRID.2010.5622068.

[**Can01**] R. Canetti. "Universally composable security: a new paradigm for cryptographic protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[**CFPS09**] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. "On the difficulty of software-based attestation of embedded devices". In: *Proceedings of the 16th ACM conference on Computer and communications security*. 2009, pp. 400–409. DOI: 10.1145/1653662.1653711.

[**CGLH+11**] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. "Principles of remote attestation". In: *International Journal of Information Security* 10.2 (2011), pp. 63–81. DOI: 10.1007/s10207-011-0124-7.

[**CLL20**] A. Corallo, M. Lazoi, and M. Lezzi. "Cybersecurity in the context of industry 4.0: A structured classification of critical assets and business impacts". In: *Computers in industry* 114 (2020), p. 103165. DOI: 10.1016/j.compind.2019.103165.

[**ERT17**] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik. "HYDRA: hybrid design for remote attestation (using a formally verified microkernel)". In: *WiSec'17: Proceedings of the 10th ACM Conference on Security and Privacy in wireless and Mobile Networks*. July 2017, pp. 99–110. DOI: 10.1145/3098243.3098261.

[**FS87**] A. Fiat and A. Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by A. M. Odlyzko. Springer Berlin Heidelberg, 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[**FIP11**] M. J. Fischer, M. Iorga, and R. Peralta. "A public randomness service". In: *Proceedings of the International Conference on Security and Cryptography*. IEEE. 2011, pp. 434–438. DOI: 10.5220/0003612604340438.

[**Gal13**] J.-P. Galland. "The difficulties of regulating markets and risks in Europe through notified bodies". In: *Eur. J. Risk Reg.* 4 (2013), p. 365. DOI: 10.1017/S1867299X00002634.

[**ISZ17**] A. Ibrahim, A.-R. Sadeghi, and S. Zeitouni. "SeED: secure non-interactive attestation for embedded devices". In: *WiSec'17: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 2017, pp. 64–74. DOI: 10.1145/3098243.3098260.

[**Kel19**] M. Kellermann. *Comprehensive Diagnostic Tool*. Annex to the QI Toolkit. The World Bank, 2019. Chap. 11, pp. 187–207. https://www.worldbank.org/en/topic/competitiveness/brief/qi.

[**KBPB19**] J. Kelsey, L. T. A. N. Brandão, R. Peralta, and H. Booth. *A Reference for Randomness Beacons: Format and Protocol Version 2. Draft NISTIR 8213*. 2019. DOI: 10.6028/NIST.IR.8213-draft.

[**KBGK17**] F. Kohnhäuser, N. Büscher, S. Gabmeyer, and S. Katzenbeisser. "Scapi: a scalable attestation protocol to detect software and physical attacks". In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 2017, pp. 75–86. DOI: 10.1145/3098243.3098255.

[**MMPM20**] W. Melo, R. C. Machado, D. Peters, and M. Moni. "Public-Key Infrastructure for Smart Meters using Blockchains". In: *2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT*. IEEE. 2020, pp. 429–434. DOI: 10.1109/MetroInd4.0IoT48571.2020.9138246.

[**MMACR19**] W. Melo Jr, R. Machado, B. Abreu, L. F. R. da Costa Carmo, and R. Ramos. "Certificação Digital como Ferramenta de Segurança para Medidores Inteligentes". In: *Anais Estendidos do IX Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. SBC. 2019, pp. 89–94. DOI: 10.5753/sbesc_estendido.2019.8641.

[**MANSV20**] T. Mustapää, J. Autiosalo, P. Nikander, J. E. Siegel, and R. Viitala. "Digital metrology for the Internet of Things". In: *2020 Global Internet of Things Summit (GIoTS)*. IEEE. 2020, pp. 1–6. DOI: 10.1109/GIOTS49054.2020.9119603.

[**NJRT20**] I. D. O. Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik. *On the TOCTOU problem in remote attestation*. arXiv: Cryptography and Security (cs.CR). 2020. arXiv:2005.03873.

[**OIM12**] OIML. *OIML D 1:2012: Considerations for a Law on Metrology*. Organization Internationale de Métrologie Légale. 2012. https://www.oiml.org/en/files/pdf_d/d001-e12.pdf.

[**OIM13**] OIML. *OIML V 1: International vocabulary of terms in legal metrology (VIML)*. Organization Internationale de Métrologie Légale. 2013. http://viml.oiml.info.

[**OETS18**] A. Oppermann, M. Esche, F. Thiel, and J.-P. Seifert. "Secure Cloud Computing: Risk Analysis for Secure Cloud Reference Architecture in Legal Metrology". In: *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE. 2018, pp. 593–602. DOI: 10.15439/2018F226.

[**PSLP12**] H. Park, D. Seo, H. Lee, and A. Perrig. "SMATT: Smart meter attestation using multiple target selection and copy-proof memory". In: *Computer Science and its Applications*. Springer, 2012, pp. 875–887. DOI: 10.1109/SECPRI.2004.1301329.

[**PPST15**] D. Peters, M. Peter, J.-P. Seifert, and F. Thiel. "A secure system architecture for measuring instruments in legal metrology". In: *Computers* 4.2 (2015), pp. 61–86. DOI: 10.3390/computers4020061.

[**PWTS18**] D. Peters, J. Wetzlich, F. Thiel, and J.-P. Seifert. "Blockchain applications for legal metrology". In: *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE. 2018, pp. 1–6. DOI: 10.1109/I2MTC.2018.8409668.

[**PYMS+20**] D. Peters, A. Yurchenko, W. Melo, K. Shirono, T. Usuda, J. Seifert, and F. Thiel. "IT security for measuring instruments: confidential checking of software functionality". In: *Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland* 1129 (2020), pp. 701–720. DOI: 10.1007/978-3-030-39445-5_51.

[**PBMC+14**] C. B. do Prado, D. R. Boccardo, R. C. Machado, L. F. da Costa Carmo, T. M. do Nascimento, L. M. Bento, R. O. Costa, C. G. de Castro, S. M. Câmara, L. Pirmez, et al. "Software Analysis and Protection for Smart Metering". In: *NCSLI Measure* 9.3 (2014), pp. 22–29. DOI: 10.1080/19315775.2014.11721691.

[**PBOE+21**] A. Przyklenk, A. Balsamo, D. O'Connor, A. Evans, T. Yandayan, A. Akgöz, O. Flys, D. Phillips, V. Zeleny, D. Czułek, et al. "New European Metrology Network for advanced manufacturing". In: *Measurement Science and Technology* (2021). DOI: 10.1088/1361-6501/ac0d25.

[**Sch91**] C.-P. Schnorr. "Efficient signature generation by smart cards". In: *Journal of cryptology* 4.3 (1991), pp. 161–174. DOI: 10.1007/BF00196725.

[**SPVK04**] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla. "SWATT: Software-based attestation for embedded devices". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.* IEEE. 2004, pp. 272–282. DOI: 10.1109/SECPRI.2004.1301329.

[**SL16**] R. V. Steiner and E. Lupu. "Attestation in wireless sensor networks: A survey". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–31. DOI: 10.1145/2988546.

[**Thi18**] F. Thiel. "Digital transformation of legal metrology — The European Metrology Cloud". In: *OIML Bulletin* 59.1 (2018), pp. 10–21. https://www.oiml.org/en/publications/bulletin.

[**WEL20**] WELMEC. *WELMEC 7.2: Software Guide (Measuring Instruments Directive 2014/32/EU).* 2020. https://www.welmec.org/guides-and-publications/guides.

[**YWZC07**] Y. Yang, X. Wang, S. Zhu, and G. Cao. "Distributed software-based attestation for node compromise detection in sensor networks". In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007).* IEEE. 2007, pp. 219–230. DOI: 10.1109/SRDS.2007.31.

[**BeaconBR**] INMETRO. *INMETRO Randomness Beacon.* https://beacon.inmetro.gov.br/. Accessed August 2020.

[**BeaconCL**] UChile. *Randomness Beacon — Random UChile.* https://random.uchile.cl/en/randomness-beacon/. Accessed August 2020.

[**BeaconUS**] NIST. *NIST Randomness Beacon.* https://beacon.nist.gov. Accessed August 2020.

[**FIPS 180-4**] National Institute of Standards and Technology (2015). *Secure Hash Standard (SHS).* (U.S. Department of Commerce) Federal Information Processing Standards Publication (FIPS PUBS) 180-4. Aug. 2015. DOI: 10.6028/NIST.FIPS.180-4.

[**FIPS 186-5**] National Institute of Standards and Technology (2019). *Digital Signature Standard (DSS).* (U.S. Department of Commerce) Draft Federal Information Processing Standards Publication (FIPS PUBS) 186-5. Oct. 2019. DOI: 10.6028/NIST.FIPS.186-5-Draft.

[**RFC 5280**] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. "Internet X.509 Public Key Infrastructure Certificate and CRL Profile". In: *RFC 5280.* Request for Comments 5280 (May 2008), pp. 1–151. DOI: 10.17487/RFC5280.

[**RFC 7748**] A. Langley, M. Hamburg, and S. Turner. "Elliptic Curves for Security". In: *RFC 7748.* Request for Comments 7748 (Jan. 2016), pp. 1–22. DOI: 10.17487/RFC7748.

[**RFC 8446**] E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC 8446.* Request for Comments 8446 (Aug. 2018), pp. 1–129. DOI: 10.17487/RFC8446.

[**SP 800-186**] L. Chen, D. Moody, A. Regenscheid, and K. Randall. *Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters.* (U.S. Department of Commerce) National Institute of Standards and Technology. Draft NIST Special Publication (SP) 800-186. Oct. 2019. DOI: 10.6028/NIST.SP.800-186-draft.

[**ZKProof**] ZKProof (many contributors). *ZKProof Community Reference. Version 0.2.* Editors: D. Benarroch, L.T.A.N. Brandão, E. Tromer. 2019. https://zkproof.org.

A. Figure for the Baseline ZKASP protocol

Figure 2 shows a detailed list of steps to implement the baseline ZKASP protocol.

The chosen flow with the specified "assignment of auditing task" phase is designed for a system where it is desirable that the auditor can justify (based on a request from the authority) every auditing request that it then makes to the instrument. Other variants can be considered. For example, if the auditor (possibly a customer) does not know h in advance and it learns it from the instrument, later the proof of correctness of the proof can only be verified against such value, deferring to the authority the verification of whether or not such h is legitimate. One can also consider the case where the instrument makes available at least the auditor's and vendor's signatures of h, i.e., (σ_A, σ_V), along with corresponding certified public keys.

Functions with self-explanatory names. The following labels are assumed to be self-explanatory: GetBeaconPulse(); GetLocalTime(); ExtractTimeRand(B_t); ExtractRand(B_t); GetLocalRand(); GetLocalState(sid); GetState(sid); GetTypeApprovalInfo(); VerProof[h](C, S); VerTimes($t, t'[, t'']$); VerSign[A](msg); StoreState(); Sign[a](msg), Store(...). Their implementation can vary based on locally decided requirements, e.g., for how long to keep state, what time-intervals to allow, etc.

Other functions/operations.

- NextAudit() selects an available auditor Y_i to audit the instrument X_i that has embedded the software h.
- GetSessionId(t_1, X_i, h, Y_j) must return a nonce (an element that never repeats across more than one session). If the system ensures that the time t_1 never repeats for two assignment tasks, the function (GetSessionId) can return a collision-resistant hash value. Otherwise, it could be a hash whose pre-image further contains a non-repeating counter or randomness.
- MsgXXXtoYYY[A, B](msg): entity of type XXX (Auth, Audi, or Inst), with public key A, sends message msg to entity of type YYY, with public key B.
- VerGetState(t') returns C and h (StoreState[t']) only if: C contains t'; C is recent; and C has not been used to produce a proof. Otherwise, it raises an exception.

Setup Inputs

Auth : (a, A) (own priv-pub key pair) (1)

Audi : (y_j, Y_j) (own priv-pub key pair), A (Auth's pub key), V (vendor's pub key) (2)

Inst : (x_i, X_i) (own priv/pub key pair), S (software) (3)

Auth: Assignment of auditing task

$(X_i, h, Y_j) = \text{NextAudit}()$ (4)

$(t_0, r_{t0}, \sigma_A, \sigma_V) = \text{GetTypeApprovalInfo}(h)$ (5)

$t_1 = \text{GetLocalTime}()$ (6)

$sid = \text{GetSessionId}(t_1, X_i, h, Y_j)$ (7)

$\sigma'_A = \text{Sign}[a]((sid, t_1, X_i, h, Y_j))$ (8)

$\text{StoreState}[sid] = (t_1, X_i, h, Y_j, \sigma'_A)$ (9)

$curr = (t_1, X_i, h, \sigma'_A)$ (10)

$approved = (t_0, r_{t0}, \sigma_A, \sigma_V)$ (11)

$\text{MsgAuthToAudi}[A, Y_j](\texttt{task}, sid, curr, approved)$ (12)

Audi (Y_j): External Challenge (on input (12))

If σ_A and σ_V are not previously checked: (13)

$\text{VerSign}[A](\sigma_A, (t_0, r_{t0}, h))$ (14)

$\text{VerSign}[V](\sigma_V, (t_0, r_{t0}, h))$ (15)

$\text{VerSign}[A](\sigma'_A, (sid, t_1, X_i, h, Y_j))$ (16)

$B_t = \text{GetBeaconPulse}[\text{latest}]$ (17)

$(t, r_t) = \text{ExtractTimeRand}(B_t)$ (18)

$t' = \text{GetLocalTime}()$ (19)

$r'_t = \text{GetLocalRand}()$ (20)

$\sigma_j = \text{Sign}[y_j]((sid, t, r_t, t', r'_t, X_i, h))$ (21)

$C = (sid, t, r_t, t', r'_t, X_i, Y_j, \sigma_j)$ (22)

$\text{StoreState}[t'] = (C, h)$ (23)

$\text{MsgAudiToInst}[Y_j, X_i](\texttt{chall}, C, h)$ (24)

Inst (X_i): Response(on input (24))

[parse] $(sid, t, r_t, t', r'_t, \overline{X_i}, Y_j, \sigma_j) = C$ (25)

$X_i =^? \overline{X_i}$ (26)

$\text{VerSign}[Y_j](\sigma_j, (sid, t, r_t, t', r'_t, X_i, h))$ (27)

$t'' = \text{GetLocalTime}()$ [optional] (28)

$\text{VerTimes}(t, t'[, t''])$ (29)

$\text{Hash}(S) =^? h$ (30)

$\pi = \Pi_{\text{hash}}(C, S) = \text{ZKPoK}_{\text{hash}}[S](h, C)$ (31)

$\sigma_i = \text{Sign}[x_i]((\pi, h, C))$ (32)

$\text{MsgInstToAudi}[X_i, Y_j](\texttt{proof}, t', \pi, \sigma_i))$ (33)

Audi (Y_j): Audit and Transfer (on input (33))

$(C, h) = \text{VerGetState}(t')$ (34)

$\text{VerProof}[h](\pi, C)$ (35)

$\text{VerSign}[X_i](\sigma_i, (\pi, h, C))$ (36)

$\text{MsgAudiToAuth}[Y_j, A](\texttt{transfer}, (C, \pi, \sigma_i))$ (37)

Auth: Central verification (on input (37))

$t_2 = \text{GetLocalTime}()$ (38)

[parse] $(sid, t, r_t, t', r'_t, \overline{X_i}, \overline{Y_j}, \sigma_j) = C$ (39)

$B_t = \text{GetBeaconPulse}[t]$ (40)

$\text{ExtractRand}(B_t) =^? r_t$ (41)

$(t_1, X_i, h, Y_j, \sigma'_A) = \text{GetState}(sid)$ (42)

$(X_i, Y_j) =^? (\overline{X_i}, \overline{Y_j})$ (43)

$\text{VerTimes}(t, t'[, t_2])$ (44)

$\text{VerSign}[Y_j](\sigma_j, (t, r_t, t', r'_t, X_i, h))$ (45)

$\text{VerProof}[h](\pi, C)$ (46)

$\text{VerSign}[X_i](\sigma_i, (\pi, h, C))$ (47)

$LE = (sid, t_1, \sigma'_A, t_2, h, C, \pi, \sigma_i, B_t)$ (48)

$\sigma''_A = \text{Sign}[a](LE)$ (49)

$\text{Store}((LE, \sigma''_A))$ (50)

Legend: Audi: **Audi**tor. Auth: **Auth**ority. B_t: **b**eacon pulse with timestamp t. C: external **c**hallenge. h: **h**ash of the software. Inst: **Inst**rument. LE: tuple variable indicating a **l**og **e**ntry. LS: tuple variable indicating a **l**ocal **s**tate; π: **p**roof transcript. Π: (probabilistic) function to produce a C-bound non-interactive ZK**P**oK of pre-image S of the hash h. r_t: **r**andom output value (`randOut`) from pulse B_t. r'_t: local **r**andom value from Audi. sid: **s**ession **id**. σ_i, σ_j: **s**ignatures by Inst i and Audi j, respectively. σ_A, σ_V: **s**ignatures by **A**uth and **V**endor at the time of software approval. σ'_A: Auth' s **s**ignature of current auditing request. σ''_A: Auth' s **s**ignature of final log entry data. S: [expected] **s**oftware in Inst i. t: **t**imestamp in pulse B_t. t_0: **t**imestamp of software approval signatures σ_A, σ_V t_1: Auth's **t**imestamp of auditing request t_2: Auth's **t**imestamp of received auditing proof t': local **t**ime at Audi. t'': local **t**ime at Inst or at Auth. (x_i, X_i): private/public keys of Inst. (y_j, Y_j): private/public keys of Audi.

Figure 2. Baseline ZKASP protocol