Iñigo Querejeta-Azurmendi*, Panagiotis Papadopoulos, Matteo Varvello, Antonio Nappa, Jiexin Zhang, and Benjamin Livshits

ZKSENSE: A Friction-less Privacy-Preserving Human Attestation Mechanism for Mobile Devices

Abstract: Recent studies show that 20.4% of the internet traffic originates from automated agents. To identify and block such ill-intentioned traffic, mechanisms that *verify the humanness of the user* are widely deployed, with CAPTCHAs being the most popular. Traditional CAPTCHAs require extra user effort (e.g., solving mathematical puzzles), which can severely downgrade the end-user's experience, especially on mobile, and provide sporadic humanness verification of questionable accuracy. More recent solutions like Google's reCAPTCHA v3, leverage user data, thus raising significant privacy concerns. To address these issues, we present zkSENSE: the first zero-knowledge proof-based humanness attestation system for mobile devices. zkSENSE moves the human attestation to the edge: onto the user's very own device, where humanness of the user is assessed in a privacy-preserving and seamless manner. zkSENSE achieves this by classifying motion sensor outputs of the mobile device, based on a model trained by using both publicly available sensor data and data collected from a small group of volunteers. To ensure the integrity of the process, the classification result is enclosed in a zero-knowledge proof of humanness that can be safely shared with a remote server. We implement zkSENSE as an Android service to demonstrate its effectiveness and practicality. In our evaluation, we show that zkSENSE successfully verifies the humanness of a user across a variety of attacking scenarios and demonstrate 92% accuracy. On a two years old Samsung S9, zkSENSE's attestation takes around 3 seconds (when visual CAPTCHAs need 9.8 seconds) and consumes a negligible amount of battery.

Keywords: Human Attestation, Privacy Preserving Bot Detection, Frictionless verification of humanness

DOI 10.2478/popets-2021-0058
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

***Corresponding Author: Iñigo Querejeta-Azurmendi:** Universidad Carlos III Madrid / ITFI, CSIC. Part of the work performed while working at Brave Software.
Panagiotis Papadopoulos: Telefónica Research
Matteo Varvello: Bell Labs
Antonio Nappa: University of California, Berkeley
Jiexin Zhang: University of Cambridge

1. Introduction

Automated software agents that interact with content in a human-like way, are becoming more prevalent and pernicious in the recent years. Web scraping, competitive data mining, account hijacking, spam and ad fraud are attacks that such agents launch by mimicking human actions at large scale. According to a recent study [1], 20.4% of the 2019 internet traffic was fraudulent, associated with *user, albeit not human, activity.*

In the ad market specifically, such type of fraudulent traffic costs companies between $6.5-$19 billions in the U.S. alone, and it is estimated that this will grow to $50 billion by 2025 [2]. When it comes to the ever-growing mobile traffic, a recent study [3] (using data spanning 17 billion transactions) observes 189 million attacks originated specifically from mobile devices; this is an increase of 12% compared to the previous six months.

The "Completely Automated Public Turing tests to tell Computers and Humans Apart" (or just CAPTCHA) is the current state-of-the-art mechanism to assess the humanness of a user. CAPTCHAs are widely deployed across the internet to identify and block fraudulent non-human traffic. The major downsides of current CAPTCHA solutions (e.g., Securimage [4], hCaptcha [5]) include: **(i) questionable accuracy**: various past works demonstrate how CAPTCHAs can be solved within milliseconds [6–10], **(ii) added friction**: additional user actions are required (e.g., image, audio, math, or textual challenges) that significantly impoverish the user experience [11], especially on mobile devices, **(iii) discrimination**: poor implementations often block access to content [12], especially to visual-impaired users [13] **(iv) serious privacy implications**: to reduce friction, Google's reCAPTCHAv3 [14] replaces proof-of-work challenges with extensive user tracking. Google's servers attest user's humanness by collecting and validating behavioral data [15] (i.e., typing patterns, mouse clicks, stored cookies, installed plugins), thus raising significant privacy concerns [16–18].

The goal of this paper is to build a humanness attestation alternative that will put an end to the false

Benjamin Livshits: Brave Software/Imperial College

dilemma: humanness attestation at the cost of user experience or at the cost of user privacy? By leveraging the device's motion sensors we demonstrate that it is possible to build a humanness attestation mechanism on the edge that preserves the privacy of the users and runs seamlessly on the background thus requiring zero interaction from the mobile user.

The key intuition behind our approach is that whenever a (human) user interacts with a mobile device, the force applied during the touch event generates motion. This motion can be captured by the device's sensors (e.g., accelerometer and gyroscope) and used to uniquely distinguish real users from automated agents. Further, this detection can run on the user device, without requiring secure execution environment (unlike related proposals [19, 20]), but more importantly without sharing private information with any server (unlike state-of-the-art [14]), apart from a *proof* that guarantees the integrity of the attestation result. Secure execution environments are rapidly evolving and might soon become a crucial tool for human attestation mechanisms, however, we have not yet seen wide adoption by low-end devices. We realize this vision with zkSENSE, a friction-less and privacy-preserving mechanism for humanness attestation that aims to replace CAPTCHA in mobile devices. This paper makes the following contributions:

1. We design a human attestation system (zkSENSE) that is both friction-less and privacy preserving. zkSENSE leverages mobile motion sensors to verify that user actions (e.g., type/touch events) on a mobile device are triggered by an actual human. Such classification takes place by studying the output of the mobile device's motion sensors during the particular user action. To set the ground truth, we use publicly available sensor traces and we instrument an actual Android browser app to capture both user clicks and sensor traces from a small set of real users. Our approach is tested under various scenarios: (i) when device is resting (on a table), (ii) when there is artificial movement from device's vibration, or (iii) from an external swinging cradle.
2. We develop a sub-linear inner product zero-knowledge proof, which we use to build (and open-source[1]) zkSVM: a zero-knowledge based library for enclosing results of an SVM (Support-Vector Machine) classifier into zero-knowledge proofs. zkSVM leverages arithmetic properties of commitment functions and prover-effective proofs to ensure the integrity of the classification result reported to a remote server.
3. We implement an Android SDK and a demo app[2] to showcase the detection accuracy of zkSENSE. zkSENSE is *invisible to the user* and capable of verifying their humanness with accuracy higher than related proposals [19] (92%). Performance evaluation results of our prototype show that the entire attestation operation takes less than 3 seconds (compared to 9 sec [21] required by Android reCAPTCHA [22] and consumes less than 5 mAh of power.

2. Goals and Threat Model

In this section, we describe (i) the basic design principles that a humanness verification mechanism must follow, and (ii) how existing mechanisms work and compare with zkSENSE. Specifically, a successful human attestation mechanism must:

A) **Be friction-less:** Most existing mechanisms require the user to solve mathematical quizzes or image/audio challenges [4, 5], thus severely hampering the user experience. According to studies [21, 23]: (i) humans only agree on what the CAPTCHA says 71% of the time, (ii) visual CAPTCHAs take 9.8 seconds (on average) to complete, (iii) audio CAPTCHAs take 28.4 seconds (and 50% of the Audio CAPTCHA users quit). The profound degradation of the user experience forces service providers to perform user attestations only sporadically [24–26] in an attempt to save their declining conversion rates. Related research works [27–29] reduce user friction by requiring, for example, the user to tilt their phone during humanness verification. In contrast, zkSENSE is completely friction-less: humanness is attested via device micro-movements that happen during natural user actions like typing, and screen touching.

B) **Be privacy-preserving**: To mitigate the above, mechanisms like reCAPTCHA v3 [14] (i) track the user while browsing a webpage, (ii) send raw tracking data to third party attestation servers, where (iii) a "risk score" representative of humanness is computed and then (iv) shared with the webmasters. Of course, this pervasive behavioral tracking raises significant privacy concerns [16, 17]. Similarly, there are sensor-based approaches [19, 27, 28] that transmit raw mobile sensor data to remote attestation servers; something that as reported, may reveal keystrokes, gender, age, or be used

1 zkSVM source code: https://github.com/iquerejeta/zkSVM

2 Demo video: https://youtu.be/U-tZKrGb8L0

to fingerprint users [30–36]. Contrary to that, zkSENSE decouples the humanness attestation procedure from the server, and moves it to the edge. By performing the entire attestation on the user's very own device, no sensitive data but the classification result (enclosed in zero-knowledge proofs to ensure integrity) leaves the user's mobile device.

C) Be broadly accessible: The majority of CAPTCHA solutions are not accessible to all users [13, 37]. According to a survey [38], CAPTCHAs are the main source of difficulty for visually impaired users. Meanwhile, human attestation mechanisms designed for visually impaired people (like audio-based reCAPTCHA) have been exploited to bypass CAPTCHAs by providing automatic responses [39, 40]. zkSENSE's seamless integration with a user's natural interaction with a device does not introduce any additional barrier to people affected by any form of disability. We must clarify however, that zkSENSE is not tested to serve people that use voice commands instead of screen touching (and thus do not cause micro-movements during scroll, clicks, etc.). Extending our classifier to work with the buttons of the devices (i.e., lock/unlock, volume up/down) that visually impaired people use is part of our future work.

2.1 Threat Model

Formal security properties: In Appendix B we introduce formal notions of privacy and verifiability, and prove that zkSVM provides them. Informally, these notions ensure that the user data is kept private from an adversary, and that a proof convinces a verifier that the submitted result can only be originated by applying the SVM model to the *committed* vector. zkSENSE does not provide any guarantees on the origin of the input data itself. Hence, as with other widely deployed human attestation mechanisms (i.e., reCAPTCHAv3) an adversary capable of forging human activity is capable of bypassing the mechanism. Our scheme provides replay prevention of proofs, but not of the data used to generate such proofs, making it possible to perform several attestations with a single vector.

Attacker: We assume the same attacker model as in CAPTCHA systems: an attacker who is capable of automating user actions while accessing an online service. The goal of such attacker is to imitate a legitimate user and launch attacks like content scraping, ad viewing/clicking, API abuse, spam sending, DDoS performing, etc., for monetary gain. This monetary gain can be achieved either (i) indirectly: e.g., app developers pay attackers to perform Black Hat App Store Optimization attacks to increase their app ranking [41–43], or (ii) directly: an attacker registers for reward schemes in mobile apps (e.g., Brave's Ad Rewards [44]) via multiple accounts and claim rewards.

As for CAPTCHA systems, click farms are not included in this threat model, since they rely on malicious but human activity. Additionally, we assume that the attacker does not control the device's OS (rooted device). Such a powerful attacker has the capabilities to launch much more severe attacks (e.g., install spyware, steal passwords or sensitive user data, modify firmware, tamper with sensors) [45, 46].

Server-auditor: We assume a set of servers that act as *auditors*, simply receiving and verifying the humanness classification results the users report. Contrary to reCAPTCHAv3, such servers are not required to be trusted by the users. Users can report a server that denies issuing tokens to an attested user.

User: Throughout the rest of this paper, when we mention user activity, we mean screen interaction, which results (when interpreted by the mobile OS) in clicking, key typing, or scrolling. We further assume mobile devices equipped with a set of sensors that includes a gyroscope and an accelerometer.

3. Building Blocks

In this section, we briefly introduce the concepts of zero-knowledge proofs and commitment functions: the basic pillars of our privacy preserving construction.

3.1 Preliminaries

Let $\mathbb{G}$ be a cyclic group with prime order p generated by generator g. Let h be another generator of the group $\mathbb{G}$ where the discrete logarithm of h with base g is not known. Furthermore, let $g_1, \ldots, g_n, h_1, \ldots, h_n$ denote $2n$ distinct group elements where their relative discrete logarithms are unknown. We denote the integers modulo p as $\mathbb{Z}_p$ and write $a \in_R S$ to denote that a is chosen uniformly at random from a set S. Although any prime order group where the Decisional Diffie–Hellman (DDH) assumption holds may be used, in our implementation we used the ristretto255 group [47] over Curve25519 [48].

Notation: We use bold letters to denote vectors. In particular, $\boldsymbol{g} \in \mathbb{G}^n$ is defined by $(g_1, \ldots, g_n)$ with $g_i \in \mathbb{G}$, and $\boldsymbol{a} \in \mathbb{Z}_p^n$ by $(a_1, \ldots, a_n)$ with $a_i \in \mathbb{Z}_p$. We use normal exponentiation notation to denote the multi-exponentiation of two vectors, with $\boldsymbol{g}^{\boldsymbol{a}} = \prod_{i=1}^{n} g_i^{a_i}$.

Moreover, two vector multiplications (or additions) represents the entry wise multiplication (or addition).

3.2 Zero-Knowledge Proofs

In [49] the technique of (interactive) zero-knowledge Proof (ZKP) is presented to enable one party (prover) to convince another (verifier) about the validity of a certain statement. The proof must provide the following informal properties (we formalise them in Appendix A):

- Completeness: If the prover and verifier follow the protocol, the latter always validates.
- Knowledge soundness: The verifier only accepts the proof if the statement being proven holds.
- Honest-verifier zero-knowledge: The prove discloses no other information other than the fact that the statement is true.

Blum et al. [50] introduced non-interactive zero-knowledge Proofs (NIZKPs), which enable the prover to prove the validity of a statement without interacting with the verifier. NIZKPs enjoy a growing popularity and adoption in the blockchain era, used across various applications mostly for decentralization, verifiability and accountability [51–54]. NIZKPs can be extended to Signature Proofs of Knowledge (SPK) [55], in which the prover signs a message while proving the statement. We adopt the Camenisch-Stadler notation [55] to denote such proofs and write, for example:

$$SPK\{(x) : A = g^x \ \wedge \ B = A^x\}(w)$$

to denote the non-interactive signature proof of knowledge, over message w, that the prover knows the discrete log of A and B with bases g and A respectively, and that the discrete log is equal in both cases. Values in the parenthesis are private (called the witness), while all other values in the proof are public. We omit the notation of the signed message throughout the whole paper, but we assume that all SPKs sign a challenge sent by the server every time it requests human attestation. This prevents replay attacks of zero knowledge proofs. We represent proofs with Π, and define two functions: the generation function, $\Pi.\texttt{Gen}()$, and the verification function, $\Pi.\texttt{Verif}()$. The inputs are, for the case of the generation, the public values and the witness (in the example above, A, B and x). On the other hand, the verification function only takes the public values as input (in the example above, A and B).

Blockchain has created a high interest in building proofs with minimal size and verifier computation [56–60], but with the cost of increasing the prover's time. In Section 7., we show how proving execution of the SVM model based on [56] (one of the most widely used such proofs) is unbearable for mobile devices.

3.3 Homomorphic Commitment Functions

In this paper, we use additive homomorphic commitment functions, a cryptographic primitive that allows a party to commit to a given value, providing the hiding property (the value itself is hidden) and the binding property (a commitment cannot be opened with a different value than the original one). We present our work based on the Pedersen Commitment Scheme [61]. The Pedersen Commitment scheme takes as input message $m \in \mathbb{Z}_p$ and a hiding value $r \in_R \mathbb{Z}_p$ and commits to it as follows:

$$\texttt{Commit}(m, r) = g^m \cdot h^r.$$

Note the additive homomorphic property, where:

$$\texttt{Commit}(m_1, r_1) \cdot \texttt{Commit}(m_2, r_2) = g^{m_1+m_2} \cdot h^{r_1+r_2} = \texttt{Commit}(m_1 + m_2, r_1 + r_2).$$

Furthermore, in this paper we use a generalization of the Pedersen Commitment [62], that instead takes a message in $\boldsymbol{m} \in \mathbb{Z}_p^n$ and a hiding value $r \in_R \mathbb{Z}_p$ and commits to it as follows:

$$\texttt{Commit}(\boldsymbol{m}, r) = \boldsymbol{g}^{\boldsymbol{m}} h^r = \prod_{i=1}^{n} g_i^{m_i} \cdot h^r.$$

Note that such a commitment function has the same additive property as above, where addition in the message committed happens entry wise. Finally, a commitment function has an opening verification function, where, given the opening and blinding factor, determines whether these correspond to a given commitment, C. More specifically:

$$\texttt{Open}(C, \boldsymbol{m}, r, \boldsymbol{g}, h) = \begin{cases} \top & \text{if } C = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^r \\ \bot & \text{else} \end{cases}$$

We leverage zero-knowledge proofs over Pedersen commitments. First, we use a proof of knowledge of the opening of a commitment [62], Π_{Op}, and a proof of equality, Π_{Eq}. Particularly:

$$\Pi_{Op} := SPK\left\{(\boldsymbol{m}, r) : C = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^r\right\}.$$

and

$$\Pi_{Eq} := SPK\left\{(\boldsymbol{m}, r_1, r_2) : C_1 = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^{r_1} \wedge C_2 = \boldsymbol{h}^{\boldsymbol{m}} \cdot h^{r_2}\right\},$$

respectively. We also construct a ZKP that allows us to provably exchange any element of the committed vector

by a zero. To this end we provably extract the particular value we want to replace with its corresponding generator, and divide it from the initial commitment. More particularly:

$$\Pi_0 := SPK\left\{(\boldsymbol{m} \in \mathbb{Z}_p^n, r) : C = \boldsymbol{g^m} h^r \wedge E = g_j^{m_j} \wedge C/E = g_1^{m_1} \cdots g_{j-1}^{m_{j-1}} \cdot g_{j+1}^{m_{j+1}} \cdots g_n^{m_n} h^r\right\}.$$

Next, we use the proof that a committed value is the square of another committed value presented in [63]:

$$\Pi_{Sq} := SPK\left\{(m_1, m_2, r_1, r_2 \in \mathbb{Z}_p) : C_1 = g^{m_1} \cdot h^{r_1} \wedge C_2 = g^{m_2} \cdot h^{r_2} \wedge m_1 = m_2^2\right\}.$$

To prove that a number is within a range, we leverage Bulletproofs [59]:

$$\Pi_{GE} := SPK\left\{(m, r \in \mathbb{Z}_p) : C = g^m h^r \wedge m \in [0, 2^l]\right\}$$

for some positive integer l. We combine these two proofs, Π_{Sq} and Π_{GE}, to prove the correct computation of the floor of the square root of a committed value. Let $C_1 = g^{m_1} \cdot h^{r_1}$, $C_2 = g^{m_2} \cdot h^{r_2}$ and $C_1^{+1} = C_1 \cdot g = g^{m_1+1} \cdot h^{r_1}$. We want to prove that $m_1 = \lfloor \sqrt{m_2} \rfloor$. It suffices to prove the following two statements:

(i) The square of the committed value in C_1 is smaller or equal than C_2, and
(ii) The square of the committed value in C_1^{+1} is greater than C_2.

We denote this proof by:

$$\Pi_{sqrt} := SPK\left\{(m_1, m_2, r_1, r_2 \in \mathbb{Z}_p) : C_1 = g^{m_1} \cdot h^{r_1} \wedge C_2 = g^{m_2} \cdot h^{r_2} \wedge m_1 = \lfloor \sqrt{m_2} \rfloor\right\}.$$

4. Sub-linear Inner Product Zero-Knowledge Proof

In this section, we present the Sub-linear Inner Product Zero-Knowledge Proof (IP-ZKP). Our construction is based on the non zero-knowledge version presented in [59]. For their use cases, the zero-knowledge property is not required, and hence the lack of an explicit definition and study of a zero-knowledge proof. In this section we make it explicit, and in Appendix A we prove that it provides completeness, knowledge soundness and special honest-verifier zero-knowledge properties. In our construction the prover has a commitment, $A = h^\alpha \boldsymbol{g^a h^b}$, of two vectors $\boldsymbol{a}, \boldsymbol{b}$ with blinding factor α, and a second commitment $V = g^c h^\gamma$, of a value c, with blinding factor γ. The prover convinces the verifier that $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = c$ holds. The prover and verifier interact as follows (can be made non-interactive with the Fiat-Shamir Heuristic [64]):

$\mathcal{P}$: It computes blinding vectors, $\boldsymbol{s}_L, \boldsymbol{s}_R$ for each vector of the inner product and commits to it:

$$\boldsymbol{s}_L, \boldsymbol{s}_R \in_R \mathbb{Z}_p^n \tag{1}$$
$$\rho \in_R \mathbb{Z}_p \tag{2}$$
$$S = h^\rho \boldsymbol{g}^{\boldsymbol{s}_L} \boldsymbol{h}^{\boldsymbol{s}_R} \in \mathbb{G} \tag{3}$$

Now the prover defines two linear vector polynomials, $l(X), r(X) \in \mathbb{Z}_p^n[X]$, and a quadratic polynomial as follows:

$$l(X) = \boldsymbol{a} + \boldsymbol{s}_L \cdot X \in \mathbb{Z}_p^n[X] \tag{4}$$
$$r(X) = \boldsymbol{b} + \boldsymbol{s}_R \cdot X \in \mathbb{Z}_p^n[X] \tag{5}$$
$$t(X) = \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2 \in \mathbb{Z}_p^n[X] \tag{6}$$

By creating like that the polynomials, it allows for an evaluation of the polynomial at a given point without disclosing any information about the vectors $\boldsymbol{a}$ or $\boldsymbol{b}$. The prover needs to convince the verifier that the constant term of $t(X)$ equals c. To do so, the prover commits to the remaining coefficients of the polynomial

$$\tau_1, \tau_2 \in_R \mathbb{Z}_p \tag{7}$$
$$T_i = g^{t_i} h^{\tau_i} \in \mathbb{G}, i = \{1, 2\} \tag{8}$$

$\mathcal{P} \to \mathcal{V}$: S, T_1, T_2
$\mathcal{V}$: $\mathcal{C} \in_R \mathbb{Z}_p^*$
$\mathcal{V} \to \mathcal{P}$: $\mathcal{C}$

$\mathcal{P}$: It computes the response using the challenge received

$$\boldsymbol{l} = l(\mathcal{C}) = \boldsymbol{a} + \boldsymbol{s}_L \cdot \mathcal{C} \in \mathbb{Z}_p^n \tag{9}$$
$$\boldsymbol{r} = r(\mathcal{C}) = \boldsymbol{b} + \boldsymbol{s}_R \cdot \mathcal{C} \in \mathbb{Z}_p^n \tag{10}$$
$$\hat{t} = \langle \boldsymbol{l}, \boldsymbol{r} \rangle \in \mathbb{Z}_p \tag{11}$$
$$\tau_\mathcal{C} = \tau_2 \cdot \mathcal{C}^2 + \tau_1 \cdot \mathcal{C} + \gamma \tag{12}$$
$$\mu = \alpha + \rho \cdot \mathcal{C} \in \mathbb{Z}_p \tag{13}$$

$\mathcal{P} \to \mathcal{V}$: $\tau_\mathcal{C}, \mu, \hat{t}, \boldsymbol{r}, \boldsymbol{l}$.

$\mathcal{V}$: The verifier needs to check that $\boldsymbol{r}, \boldsymbol{l}$ are correct and the inner product relation holds with respect to $\hat{t}$. To this end it performs the following checks:

$$g^{\hat{t}} h^{\tau_\mathcal{C}} \stackrel{?}{=} V \cdot T_1^\mathcal{C} \cdot T_2^{\mathcal{C}^2} \tag{14}$$
$$P = A \cdot S^\mathcal{C} \in \mathbb{G} \tag{15}$$
$$P \stackrel{?}{=} h^\mu \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot \boldsymbol{h}^{\boldsymbol{r}} \tag{16}$$
$$\hat{t} \stackrel{?}{=} \langle \boldsymbol{l}, \boldsymbol{r} \rangle \tag{17}$$

If all checks validate, then this means that the statement is true with very high probability.

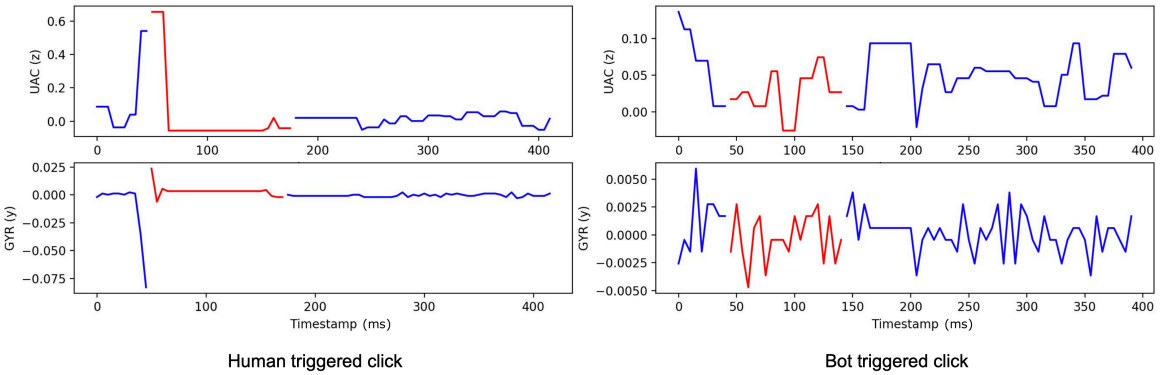

Fig. 1. Output of gyroscope and accelerometer motion sensors during human and automatically triggered click (red). The maximum linear acceleration movement is up to 8.5× greater and the angular rotational velocity is up to 4.9× greater in case of a human triggered click.

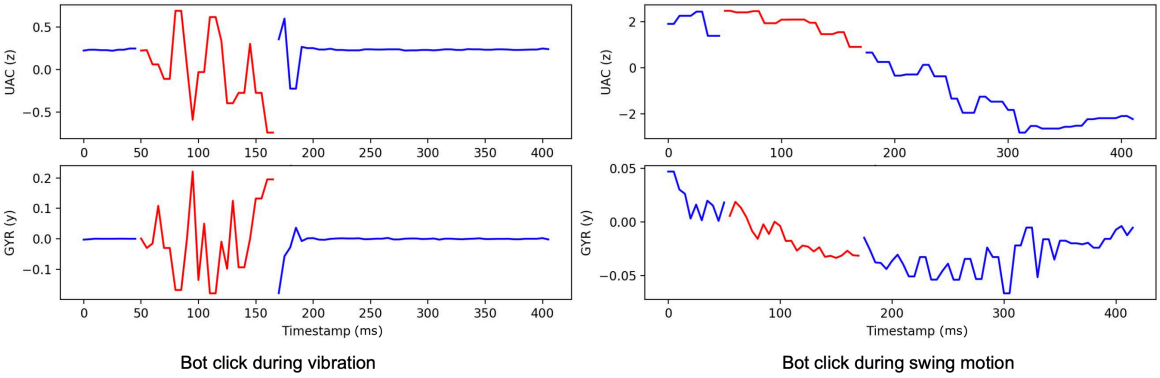

Fig. 2. Motion sensors output during automatically triggered click with artificial device movement (red): (i) during device vibration (on the left) and (ii) when device is docked on a swing (on the right).

To make this proof logarithmic, we use the same trick as in the original paper. Instead of sending the vectors $\boldsymbol{r}, \boldsymbol{l}$ to prove the inner product relation, we leverage IP-ZKP over the blinded vectors recursively. We prove the following Theorem in Appendix A:

Theorem 1. *The inner product proof presented in Protocol 1 has perfect completeness, perfect special honest verifier zero-knowledge, and knowledge soundness.*

5. ZKSENSE Overview

The key intuition behind ZKSENSE is that whenever a (human) user interacts with the mobile's display, the force applied during the touch event generates motion. This motion is captured by the embedded IMU (Inertial Measurement Unit) sensors (e.g., accelerometer and gyroscope). By contrast, when there is automated user activity (e.g., simulated touches) there is no external force exerted by fingers, and thus there is no noticeable change in the output of the above sensors.

Figure 1 shows a snapshot of the output produced by IMU sensors during click events performed both by a human (left plots) and an automated agent (right plots). During the click events (highlighted in red) the accelerometer (top plots) senses a max rate of change of 0.6 in case of human and 0.07 for an automated agent, i.e., 8.5× greater maximum linear acceleration movement. Similarly, the gyroscope (bottom plots) senses a max rate of change of 0.024 in case of human click and 0.0049 when there is automation, i.e., 4.9× greater maximum angular rotational velocity.

Figure 2 shows a snapshot of the same sensors' output in the case of automated clicks coupled with two artificial device movements: vibration and swing. With vibration (left plots), we see that the motion generated is comparable with the case of the human click depicted in Figure 1. We see that the accelerometer senses the same force with the case of the human click but for a longer time – this verifies the observations of [19]. The gyroscope though senses greater angular rotational velocity and for longer time than in the case of a human's click. With swing (right plots), the gyroscope senses similar angular rotational velocity as with the case of the human click, while the accelerometer senses greater linear acceleration movement (up to 3.8×) for a long period.

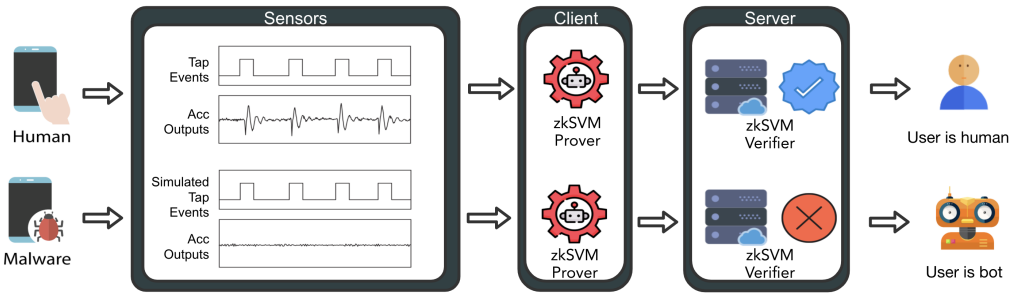

Fig. 3. High-level overview of the ZKSENSE architecture. An integrated ML-based classifier studies the patterns of sensor outputs right before, during, and shortly after a click event. To avoid leaking sensitive sensor output outside the device, the classification appears on the user side and the client has to prove the integrity of the reported result to the server.

5.1 System Overview

Building upon the above observations, ZKSENSE uses an ML-based classifier to study the pattern of sensor outputs before, during, and shortly after a click event (see Figure 3). Based on this information, the model decides about whether the action was triggered by a human or not.

Human attestation on the edge: Privacy-preserving evaluation of ML models has become increasingly important especially in the post GDPR era. One approach consists in encrypting the data on the client, and runs ML model on such encrypted data at the server. This can be achieved via Fully Homomorphic Encryption (FHE) [65–67]: clients encrypt their data with their own keys and send the ciphertext to the server to evaluate an ML model. Next, the server sends the outcome of the homomorphic computation back to the user who would provably decrypt it and send back to the server. It is easy to anticipate that the overhead to perform such multi-step operation for each client is unbearable for services with millions of clients.

In ZKSENSE, we pre-train a model on a server and we move the classifier to the edge by running it on the user side and only report the result to an attestation server responsible for auditing the humanness of the user. This way, ZKSENSE ensures that private sensor data never leaves the user's device. In Figure 3, we present the high level overview of our approach. As we can see, an attestation starts with a click (screen touch). The motion sensor outputs generated during this event are used as input to the zkSVM Prover module, which runs a trained model to classify if the action was conducted by a human or not.

5.2 Classification of Humanness

Data collection: To collect the necessary ground truth to train the various tested models, we instrumented the open source browser Brave for Android to capture click events (and their corresponding motion sensor traces) performed during browsing[3]. Then, we recruited 10 volunteers[4] who used our instrumented browser for 22 consecutive days for their daily browsing[5]. The device models used are: Google Pixel 3, Samsung Galaxy S9, S8 and S6, OnePlus 6, Realme X2 Pro, Huawei Mate 20 Lite and Honor 9. Volunteers were well-informed about the purpose of this study and gave us consent to collect and analyse the motion sensor traces generated during their screen touch events. We urged volunteers to use their phone as normal.

Data	Data amount
Volunteering Users	10 users
Duration of collection	22 days
Android Devices tested	Google Pixel 3, Realme X2 Pro, Samsung Galaxy S9/S8/S6, Honor 9, Huawei Mate 20 Lite, OnePlus 6
Human events collected	7,736 clicks
Artificial events collected	25,921 clicks

Table 1. Summary of the collected dataset.

To generate artificial user traffic, we used `adb` [68] to automate software clicks on 4 of the volunteering devices. To test different attack scenarios, during the automation, we generate software clicks with the device being in 4 different states:

3 We chose a real app to make sure that the volunteers will continue using it for a longer period contrary to a possible instrumented toy app.

4 For a production ready system a larger pool of real users will need to contribute data, but for our study we found it difficult to recruit users who would contribute their clicks for a long time.

5 During data collection the instrumented browser was running on the users' devices so we could not control their background running services. The data collected included only the raw sensor data during a user click.

Classifier	F1 (weighted)	Recall
SVM	0.92	0.95
Decision Tree (9 Layers)	0.93	0.95
Random Forest (8 Trees, 10 Layers)	0.93	0.95
KNN	0.92	0.93
Neural Network (Linear Kernel)	0.86	0.95
Neural Network (ReLU Kernel)	0.91	0.96

Table 2. Accuracy of the various tested classifiers.

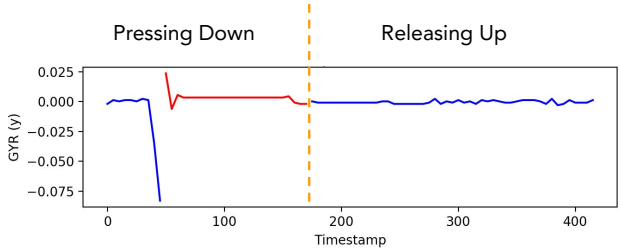

Fig. 4. The period of a click event starts 50ms before the beginning of the action and ends 250ms after the end of it.

1. while resting on a platform (desk/stand)
2. while being carried around in pocket
3. while being placed on a swing motion device
4. while device is vibrating (triggered by `adb`)

As summarized in Table 1, by the end of the data collection, we had 7,736 human-generated clicks and 25,921 artificially generated clicks.

Feature selection: During data collection, accelerometer and gyroscope sensors were sampled at 250Hz. For each click event, we not only consider the device motion during the touch, but also the device motion right before and shortly after the touch. In particular, we consider that the period starts 50ms before the finger touches the screen and finishes 250ms after it. Then, we split each period into two segments: (i) *before releasing finger* and (ii) *after releasing finger* as depicted in Figure 4. For each axis (x, y, z) in accelerometer and gyroscope, we calculate the average and standard deviation of its outputs in each segment. In addition, we calculate the consecutive difference of sensor outputs in each segment and use the average and standard deviation of these differences as features.

Classification accuracy: Using the above features, we test several ML classifiers via 10-fold cross validation. Table 2 presents the weighted F_1 score and recall of the different classifiers we tested. We choose weighted F_1 score as an evaluation index because our dataset is unbalanced. In this context, *recall* means the proportion of correctly identified artificial clicks over all artificial clicks. In other words, recall indicates the ability to capture artificial clicks.

As shown in Table 2, the four tested classifiers (i.e., SVM, decision tree, random forest, and neural network with ReLU kernel) have similar performance in terms of recall (i.e., 0.95 recall). Although, Decision Tree and Random Forest perform slightly better in terms of accuracy, zkSENSE utilizes SVM for compatibility purposes as stated in more detail in Section 5.3. Hence, our zkSENSE's accuracy in assessing the humanness of a user is 92%.

5.3 Privacy-Preserving and Provable ML

To preserve privacy, human attestation in zkSENSE is performed in the user's device and only the *result* of the attestation is shared with the server. To ensure that the server can verify the integrity of the transmitted result, the user includes a commitment of the sensors, together with a proof that the result corresponds to the model evaluated over the committed values.

We build the two ML evaluation proving components of zkSENSE: (i) zkSVM Prover and (ii) zkSVM Verifier. The zkSVM Prover checks on the client whether a user is a human based on a model we pre-trained (Section 5.2), and generates a proof to ensure its proper execution. The zkSVM Verifier, on the server's side, checks that the proof is correctly generated. If the verification is successful, the server will know that (a) the ML-based humanness detection model classifies the user as human or non-human based on the committed sensor outputs, and that (b) the used model is the genuine one, without though learning the value of sensor outputs.

SVM enclosed in zero-knowledge proofs: Section 5.2 shows that the different classifiers tested achieve similar accuracy. While decision trees, random forests, or neural networks provide slightly higher F1 accuracy than SVM (see Table 2), in zkSENSE we choose SVM as the underlying model due to its simplicity at evaluation time and its suitability with zero-knowledge proofs. Contrarily, neural networks need to perform non-linear operations, while decision trees require several range proofs, which are expensive operations to prove in zero-knowledge. As mentioned in Section 5.2, the SVM model we trained uses as features the average (μ) and standard deviation (σ) of sensor outputs, together with the average and standard deviation of the consecutive difference vector. On top of that, before applying the SVM model, the extracted features need to be normalised. The goal of normalisation is to change data values to a common scale, without distorting differences in the ranges of val-

ues. Then, trained SVM weights are assigned to each normalised feature to calculate the SVM score.

Suppose for each feature f_i, the normalisation mean, normalisation scale, and SVM weight and intercept are M_i, S_i, w_i and c respectively. Then, the SVM score s can be calculated with the equation:

$$s = \frac{1}{e^{-(c+\sum_{i=1}^{N} \frac{(f_i - M_i)}{S_i} w_i)} + 1}. \tag{18}$$

Since only the value of f_i is secret, we only need to prove the computation of $\sum_{i=1}^{N} f_i \frac{w_i}{S_i}$. Given only integer values can be processed in the underlying group arithmetic of Pedersen commitments, we instead prove $\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{S_i} 10^d \right\rfloor$ and effectively use the parameter d to preserve d-digits after the decimal points of $\frac{w_i}{S_i}$.

Building zkSVM: Without loss of generality, we assume that the number of sensor inputs is n for every sensor. The protocol is divided in three phases. First, the setup phase, $\texttt{Setup}(\lambda)$, where the server generates the model and their corresponding weights. Secondly, the proving phase, where the prover fetches the SVM related data, computes the difference vector, and proceeds to provably compute the average and standard deviation of these values. It then applies the corresponding linear combinations to the hidden features, and opens the result to send it to the verifier. Finally, the verification phase, where the verifier checks that all computations were correctly performed. Then, it checks whether the scores do correspond to a human or a bot.

Setup phase: The setup involves the server, where it generates the cryptographic material, and trains the SVM model. The details on how this model is generated falls out of the scope of zkSVM, the only requirement is that it follows the specifications described at the beggining of this section. Once these values are computed, they are published to allow provers to fetch them and generate the proofs.

Procedure 1. (Setup) On input the security parameter, λ, the zkSVM server runs $\texttt{Setup}(\lambda)$ to generate the SVM and cryptographic parameters. Precisely, it trains the SVM model and generates the normalisation mean, normalisation scale, and SVM weight and intercept, M_i, S_i, w_i and c respectively. It also defines the size of the input vectors, n, which defines how long a touch is considered and the measurement frequency. Next it selects the group, $\mathbb{G}$ with generators g, h and prime order p. It proceeds by computing two vectors, $\boldsymbol{g}, \boldsymbol{h}$, of generators that act as bases for the Pedersen Vector Commitments. Note that the corresponding discrete log of these bases must remain unknown.

Proof generation: The proof generation is divided in five protocols. First the prover computes the difference vectors and proves correctness. Next it computes the average of all vectors, followed by a computation of the standard deviation. Finally, it evaluates the normalising linear computations over the results, and sends the opening of the result to the verifier together with the proofs of correctness. The prover's secret is $\mathbf{v} \in \mathbb{Z}_p^n, r \in \mathbb{Z}_p$ such that

$$S_H = \boldsymbol{g}^{\mathbf{v}} \cdot h^r.$$

Procedure 2. (Consecutive difference) In this step the prover's goal is to compute the difference of consecutive values in the input vector, while keeping it hidden. Mainly, we want a provable value of

$$S_H^d = \boldsymbol{g}^{\mathbf{v}_d} h^{r_d},$$

with $\mathbf{v}_d = [\mathbf{v}_1 - \mathbf{v}_2, \mathbf{v}_2 - \mathbf{v}_3, \ldots, \mathbf{v}_{n-1} - \mathbf{v}_n, 0]$. The intuition here is first to get a commitment of the iterated values of the sensor vector, then leverage the homomorphic property to subtract this commitment with S_H and finally provably replace the value in position n by zero. To compute the iterated value, the prover first iterates the base generators, to get

$$\boldsymbol{g}_{iter} = [g_n, g_1, \ldots, g_{n-1}].$$

Note that this step can be performed by the verifier as the generators are public. It then commits the sensor vector with this base

$$S_H^{iter} = \boldsymbol{g}_{iter}^{\mathbf{v}} \cdot h^{r_{iter}},$$

with $r_{iter} \in_R \mathbb{Z}_p$, and generates a proof of equality,

$$\Pi_{Eq} = \Pi_{Eq}.\texttt{Gen}(\boldsymbol{g}, \boldsymbol{g}_{iter}, h, S_H, S_H^{iter}; \mathbf{v}, r, r_{iter}).$$

Note that

$$S_H^{iter} = g_1^{\mathbf{v}_2} \cdots g_{n-1}^{\mathbf{v}_n} \cdot g_n^{\mathbf{v}_1} \cdot h^{r_{iter}},$$

so now we can simply subtract the two commitments to get

$$\overline{S_H} = S_H / S_H^{iter} = g_1^{v_1 - v_2} g_2^{v_2 - v_3} \cdots g_{n-1}^{v_{n-1} - v_n} g_n^{v_n - v_1} \cdot h^{r - r_{iter}}.$$

Finally, the prover replaces the value in the exponent of g_n by a zero, to get the final commitment:

$$Diff = g_1^{v_1 - v_2} g_2^{v_2 - v_3} \cdots g_{n-1}^{v_{n-1} - v_n} g_n^0 \cdot h^{r_d}, \tag{19}$$

and generates a proof of correctness,

$$\Pi_0 = \Pi_0.\texttt{Gen}(\boldsymbol{g}, h, \overline{S_H}, Diff; \mathbf{v}_d, r - r_{iter}, r_{diff}). \tag{20}$$

It stores $\Delta = \left[Diff, S_H^{iter}, \Pi_{Eq}, \Pi_0\right]$.

Procedure 3. (Sum of vectors) The prover now computes $\tilde{\mu} = N \cdot \mu$, mainly, the sum of all values. To provably compute this, we leverage IP-ZKP between the initial commitment, S_H, and a Pedersen commitment with base $\boldsymbol{h}$ of the one vector, $\boldsymbol{h^1}$, to prove that a third commitment, $Avg = \texttt{Comm}(\tilde{\mu}, r_\mu)$, commits to the sum of the committed values in S_H,

$$\langle \mathbf{v}, \mathbf{1} \rangle = \tilde{\mu}.$$

The user proves correctness of the commitment,

$$\Pi^{\mu}_{IP} = \Pi_{IP}.\texttt{Gen}(\boldsymbol{g}, \boldsymbol{h}, g, h, S_H \cdot \boldsymbol{h^1}, Avg; \mathbf{v}, r).$$

It stores both values $\mathrm{M} = \left[Avg, \Pi^{\mu}_{IP}\right]$. It repeats the same steps as above with the commitment of the consecutive difference vector, resulting in a commitment of the average, Avg', and a proof of correctness, ${\Pi^{\mu}_{IP}}'$. It stores both values $\mathrm{M}' = \left[Avg', {\Pi^{\mu}_{IP}}'\right]$.

Procedure 4. (Standard deviation) To calculate a factor of the standard deviation, σ, we first compute the variance, σ^2. Recall that

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(v_i - \mu)^2,$$

or written differently

$$\sigma^2 = \frac{1}{N}\langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu} \rangle,$$

where $\boldsymbol{\mu}$ is a vector with μ in all its positions. For this we need the average, but only have a provable commitment of the sum, $\tilde{\mu}$. Hence, instead of computing the variance, we compute $N^3 \cdot \sigma^2$ by leveraging the inner product proof and the arithmetic properties of the commitment function. The intuition is the following: if we multiply each entry of $\mathbf{v}$ by N, we can get the following relation.

$$\begin{aligned}\langle N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}, N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}} \rangle = \\ \langle N \cdot \mathbf{v} - N \cdot \boldsymbol{\mu}, N \cdot \mathbf{v} - N \cdot \boldsymbol{\mu} \rangle = \\ \langle N \cdot (\mathbf{v} - \boldsymbol{\mu}), N \cdot (\mathbf{v} - \boldsymbol{\mu}) \rangle = \\ N^2 \langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu} \rangle = N^3 \cdot \sigma^2. \quad (21)\end{aligned}$$

However, we only have S_H, and a provable commitment of $\tilde{\mu}$ (not of $\tilde{\boldsymbol{\mu}}$). Moreover, we need a commitment of $\mathbf{v}$ and $\tilde{\boldsymbol{\mu}}$ under both bases ($\boldsymbol{g}$ and $\boldsymbol{h}$). To this end, the prover computes the following steps. First, it computes the commitment of $\tilde{\boldsymbol{\mu}}$ with both bases. To this end, the prover first computes a product of all the bases,

$$g_\Pi = \prod_{i=1}^{n} g_i \quad \text{and} \quad h_\Pi = \prod_{i=1}^{n} h_i.$$

Note that this step is again reproducible by the verifier, and hence no proof is required. Next it commits the average using these products as a base, to get

$$G^{\tilde{\mu}} = g_\Pi^{\tilde{\mu}} \cdot h^{r_G} = g_1^{\tilde{\mu}} \cdots g_n^{\tilde{\mu}} \cdot h^{r_G},$$

and

$$H^{\tilde{\mu}} = h_\Pi^{\tilde{\mu}} \cdot h^{r_H} = h_1^{\tilde{\mu}} \cdots h_n^{\tilde{\mu}} \cdot h^{r_H},$$

with $r_G, r_H \in_R \mathbb{Z}_p$. It proves equality between the opening of $Avg, G^{\tilde{\mu}}$ and $H^{\tilde{\mu}}$ using Π_{Eq}, and stores the two proofs,

$$\Pi^{G}_{Eq} = \Pi_{Eq}.\texttt{Gen}(g_\Pi, g, h, Avg, G^{\tilde{\mu}}; \mu, r_\mu, r_G),$$

and

$$\Pi^{H}_{Eq} = \Pi_{Eq}.\texttt{Gen}(h_\Pi, g, h, Avg, H^{\tilde{\mu}}; \mu, r_\mu, r_H),$$

one for each base. Finally, the prover commits to $\mathbf{v}$ with randomness r_S with $\boldsymbol{h}$ as bases,

$$H_S = \boldsymbol{h}^{\mathbf{v}} h^{r_S},$$

and proves equality of opening with respect to S_H, getting

$$\Pi^{S}_{Eq} = \Pi_{Eq}.\texttt{Gen}(\boldsymbol{g}, \boldsymbol{h}, h, S_H, H_S; \mathbf{v}, r, r_S).$$

This allows the prover to leverage relation (21) to provably compute a commitment of a factor of the variance using the proof presented in Section 4. To this end, it computes

$$\begin{aligned}A_S = S_H^N / G^{\tilde{\mu}} \cdot H_S^N / H^{\tilde{\mu}} = \\ \boldsymbol{g}^{N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}} \cdot \boldsymbol{h}^{N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}} \cdot h^{N \cdot r - r_G + N \cdot r_S - r_H}\end{aligned}$$

and the commitment of the factor of the variance,

$$Var = \texttt{Comm}(N^3 \cdot \sigma^2, r_V).$$

It generates a proof of correctness

$$\begin{aligned}\Pi^{\sigma^2}_{IP} = \Pi_{IP}.\texttt{Gen}(A_S, Var, \boldsymbol{g}, \boldsymbol{h}, g, h; \\ \mathbf{v}, \tilde{\mu}, r, r_G, r_H, r_S, r_V).\end{aligned}$$

Finally, the prover needs to compute the square root of the variance. To this end, it commits to the floor of the square root of $N^3 \cdot \sigma^2$, $Std = \texttt{Commit}(\lfloor \sqrt{N^3 \cdot \sigma^2} \rfloor, r_{\surd})$. Then, the prover leverages the square root proof introduced in Section 3.3, and generates

$$\Pi^{\sigma}_{sqrt} = \Pi_{sqrt}.\texttt{Gen}(Var, Std, g, h; \sigma^2, \sigma, r_V, r_{\surd}).$$

This results in a provable commitment of a factor of the floor of the standard deviation, *Std*. The prover stores,

$$\Lambda = \Big[G^{\tilde{\mu}}, H^{\tilde{\mu}}, H_S, \Pi^G_{Eq}, \Pi^H_{Eq}, \Pi^S_{Eq}, Var, \Pi^{\sigma^2}_{IP}, Std, \Pi^{\sigma}_{sqrt}\Big].$$

It repeats the same steps above with the consecutive difference vector (and average), resulting in

$$\Lambda' = \Big[G^{\tilde{\mu}\prime}, H^{\tilde{\mu}\prime}, H_S{}', \Pi^G_{Eq}{}', \Pi^H_{Eq}{}', \Pi^S_{Eq}{}', Var', \Pi^{\sigma^2}_{IP}{}', Std', \Pi^{\sigma}_{sqrt}{}'\Big].$$

Procedure 5. (Computing SVM score) Provably computing SVM score (Eq. 18) reduced to provably computing $\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{S_i} 10^d \right\rfloor$ where f_i are the features. However, note that we do not have the features themselves, but a factor of them. Hence, with this scheme, we need to compute instead $r = \sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$ where N_i equals N if f_i is an average, and $N^{3/2}$ if it is a standard deviation (note that we have different factors of each). Again, $\frac{w_i}{N_i \cdot S_i}$ is public. Let $\texttt{Comm}_i = \texttt{Commit}(f_i, r_i)$ be the commitment of feature f_i with blinding factor r_i. The prover computes the following:

$$Res = \prod_{i=1}^{N} \texttt{Comm}_i^{\left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor} = \prod_{i=1}^{N} \texttt{Commit}\left(f_i \cdot \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor, r_i \cdot \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor \right) = \texttt{Commit}\left(\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor, r_R \right), \quad (22)$$

where $r_R = \sum_{i=1}^{N} r_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$, is the blinding factor known to the prover. Once these operations have been performed, the prover stores the opening of the commitment, $Score = \sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$, and the blinding factor r_R.

Procedure 6. (Sending values) The prover sends to the verifier the following tuple

$$\big[S_H, \Delta, \mathrm{M}, \mathrm{M}', \Lambda, \Lambda', Score, r_R\big]$$

Proof verification: The verifier then performs all respective linear combinations commitments, and verifies the zero-knowledge proofs. If any proof fails or the evaluation of the model over *Score* fails, the verifier denies the request. Else, it accepts it. More precisely, the verifier follows the following steps:

Procedure 7. (Verifying consecutive difference) The verifier begins by iterating the base of generators:

$$\boldsymbol{g}_{iter} = [g_n, g_1, \ldots, g_{n-1}].$$

and then verifies the proof of opening equality,

$$\Pi_{Eq}.\texttt{Verif}(\boldsymbol{g}, \boldsymbol{g}_{iter}, h, S_H, S_H^{iter}) \stackrel{?}{=} \top.$$

Next it computes the subtraction commitment to get

$$\overline{S_H} = S_H / S_H^{iter}.$$

Finally, it verifies that the proof

$$\Pi_0.\texttt{Verif}(\boldsymbol{g}, h, \overline{S_H}, Diff) \stackrel{?}{=} \top.$$

Procedure 8. (Verifying sum of vectors) The verifier checks the inner product proof

$$\Pi^{\mu}_{IP}.\texttt{Verify}(\boldsymbol{g}, \boldsymbol{h}, g, h, S_H \cdot \boldsymbol{h^1}, Avg) \stackrel{?}{=} \top.$$

It repeats this step for the consecutive difference commitment and proof.

Procedure 9. (Verifying standard deviation) The verifier first computes a product of all the bases,

$$g_\Pi = \prod_{i=1}^{n} g_i \quad \text{and} \quad h_\Pi = \prod_{i=1}^{n} h_i.$$

Next it verifies the proofs of equality of commitments using these bases

$$\Pi^G_{Eq}.\texttt{Verify}(g_\Pi, g, h, Avg, G^{\tilde{\mu}}) \stackrel{?}{=} \top,$$

and

$$\Pi^H_{Eq}.\texttt{Verify}(h_\Pi, g, h, Avg, H^{\tilde{\mu}}) \stackrel{?}{=} \top.$$

Next, the verifier checks that H_S commits to the input vector

$$\Pi^S_{Eq}.\texttt{Verify}(\boldsymbol{g}, \boldsymbol{h}, h, S_H, H_S) \stackrel{?}{=} \top.$$

Now the verifier needs to generate the commitments under which the inner product proof of the variance will verify against. To this end it computes

$$L = S_H / G^{\tilde{\mu}} \text{ and } R = H_S / H^{\tilde{\mu}} \quad (23)$$

and uses them to verify the inner product proof

$$\Pi^{\sigma^2}_{IP}.\texttt{Verify}(L \cdot R, Var, \boldsymbol{g}, \boldsymbol{h}, g, h) \stackrel{?}{=} \top.$$

Finally, the verifier checks the correctness of the factor of the standard deviation commitment

$$\Pi^{\sigma}_{sqrt}.\texttt{Verify}(Var, Std, g, h) \stackrel{?}{=} \top.$$

It repeats these steps for the consecutive difference commitments.

Procedure 10. (Computing SVM score) Finally, the verifier computes the same linear combinations as the prover,

$$Res' = \prod_{i=1}^{N} \texttt{Comm}_i^{\left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor}, \tag{24}$$

and checks the validity of the received opening,

$$\texttt{Open}(Res', Score, r_R, g, h) \stackrel{?}{=} \top.$$

It uses this value to compute the final score as described in Equation (18). If any of the checks fail or the score determines the user is a bot, it returns $\perp$, otherwise it returns $\top$.

By extending the inner product proof presented in [59] to a zero-knowledge proof and leveraging the arithmetic properties of Pedersen commitments, we present zkSVM, a privacy-preserving SVM evaluation model.

6. System Implementation

To assess the feasibility and effectiveness of our approach we developed (i) an open source library of zkSVM, and (ii) a prototype Android SDK of zkSENSE.

6.1 Enclosing SVM Result in a ZKP

The zkSVM library: To prove the integrity of the classification results, we developed an open-sourced Rust library that implements the logic presented in Protocol 2, on enclosing SVM results in zero-knowledge proofs. To this end, we additionally implement the Pedersen Commitment ZKPs as described in Section 3.3. For the proofs $\Pi_{Sq}, \Pi_{Eq}, \Pi_0$ and Π_{Op}, we base our implementation in the work presented in [62]. We use the range proof presented in [59] and implemented in [69]. Finally, for Π_{IP}, we implement the zero-knowledge proof presented in Section 4. All the above proofs are implemented using the ristretto255 prime order group over Curve25519 by leveraging the curve25519-dalek [48] library. All our proofs leverage the Fiat-Shamir heuristic to make the ZKPs non-interactive, which maintains all security properties under the Random Oracle model [70]. To integrate this library with our detection engine, we use the Android NDK development kit.

General-purpose ZK-SNARK: To compare the performance of zkSVM, we implement the SVM execution using the ZoKrates general-purpose ZK-SNARK toolbox [71]. ZoKrates works as a high-level abstraction for the encoding of the computation to be proved into a ZK-SNARK. ZoKrates constructs the ZKP by using the Rust implementation of Bellman's [72] Groth16 ZK-SNARK [56]. This construction has optimal proof size and verification time. However, this comes by trading off prover's computational complexity and the requirement of a trusted setup.

6.2 ZKSENSE Prototype

We implemented a prototype SDK of zkSENSE for Android (∼1,000 lines of Java). The SDK monitors Android's accelerometer and gyroscope during a touch event and, by applying a pre-trained model, it determines if it was performed by a human or not. For demonstration purposes, we created a demo app with a user interface that shows the output of the detection model and we provide publicly a video[6] that demonstrates its functionality. In this demo, we test multiple scenarios to showcase the accuracy of our system:

1. The device is resting on a steady platform and:
   (a) A human is performing clicks.
   (b) Clicks are simulated.
2. The device is docked on a swing motion device that produce artificial movement.
3. The device is held in one hand and:
   (a) A human is performing clicks.
   (b) Clicks are simulated.

After reading the output of the accelerometer and gyroscope sensors, the zkSENSE SDK applies, on the background, our pre-trained model and classifies the origin of the touch-screen event (i.e., performed by a human or not). For the generation of the ZKPs and the model's evaluation, we leverage the library we implemented and described previously, which we call from the mobile device using the sensor data. The pre-trained model, is generated on a server of ours. Apart from generating and distributing the trained model, the server also acts as the external auditor that verifies the validity of the transmitted attestation results.

7. Performance Evaluation

In this section, we set out to explore the performance of humanness attestation in zkSENSE. More specifically, we benchmark our Android prototype with respect to the *duration* of its main operations: (i) humanness classification, (ii) Pedersen commitment computation, and (iii) zero-knowledge proof construction. Next, we evaluate general resource utilization metrics: (a) CPU, (b)

6 zkSENSE demo: https://youtu.be/U-tZKrGb8L0

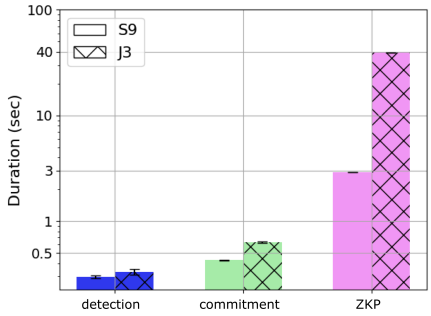

Fig. 5. Execution time per operation. On commodity hardware (S9) humanness detection and commitments are extremely fast (i.e., about 0.3), when the ZKP generation lasts about 2.9 seconds.

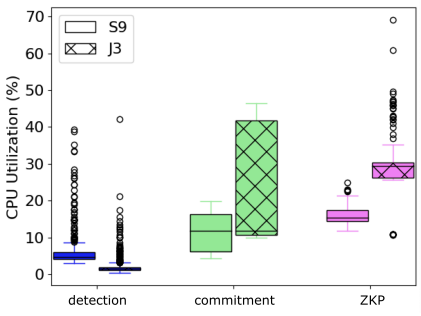

Fig. 6. CPU utilization per operation. On commodity hardware (S9), the ZKP generation is the most expensive operation, with a median CPU consumption of about 15% on commodity hardware.

Fig. 7. Energy consumption per operation on a low-end hardware. Energy consumption for detection and committing operation is negligible, when for the ZKP computation zkSENSE consumes about 5 mAh or 0.2% of the device's battery.

memory, and (c) battery consumption. Our tests cover the three key operations of a humanness attestation in zkSENSE, and a comparative baseline:

1. The *baseline*, where we run our demo application which uses zkSENSE service (see Section 6.) and several artificial clicks are generated.
2. The *detection* operation, where sensors input is collected and humanness classification realized on zkSENSE.
3. The *commitment* operation where the Pedersen commitment computation is taking place.
4. The *ZKP* operation where the proof of correct attestation is constructed.

We test and compare the different implementations described in Section 6.: (i) the general-purpose ZK-SNARK and (ii) our zkSVM. Note that for ZK-SNARK we ignore the time elapsed while computing the trusted setup, as this cannot be computed by the client. We run each stage for an hour and we ensure the same number of artificial clicks by using as an interval the duration of the longest operation (ZKP) as empirically measured on each device under test (Figure 5).

Setup: We leverage a testbed composed of two Android devices representative of a mid-end (Samsung Galaxy S9, model 2018) and a low-end (Samsung Galaxy J3, model 2016) device to inspect what is the worst performance a user can get on a cheap (around 90 USD) device. The S9 mounts an octa-core processor (a Quad-Core Mongoose M3 at 2.7GHz and a Quad-Core ARM Cortex-A55 at 1.8Ghz), while the J3 is equipped with a quad-core ARM Cortex A53 at 1.2 Ghz. The S9 also has twice as much memory (4 GB when J3 has 2 GB) and a larger battery (3,000 mAh when the battery of J3 is 2,600 mAh). The low-end device (J3) is part of Batterylab [73, 74], a distributed platform for battery measurements. It follows that fine grained battery measurements (via a Monsoon High Voltage Power Monitor [75] directly connected to the device's battery) are available for this device. Automation of the above operations is realized via adb run over WiFi to avoid noise in the power measurements caused by USB powering.

Execution time: Figure 5 shows the average duration (standard deviation as error-bars) of each zkSENSE's operation, per device, when considering zkSVM. Regardless of the device, humanness classification and commitments are extremely fast, i.e., about 0.3 and 0.6 seconds even on the less powerful J3. The ZKP generation is instead more challenging, lasting about 2.9 and 39.2 seconds on the S9 and J3, respectively.

To improve visibility, the figure omits results from the general purpose ZK-SNARK solution. In this case, we measure commitment operations comprised between 24 and 190 seconds, and ZKP generation comprised between 175 and 600 seconds, depending on the device. This suggests one order of magnitude speedup of zkSVM versus the more generic ZK-SNARK solution. Given the significant difference between zkSVM and ZK-SNARK, in the following we mostly focus on results obtained via zkSVM.

For the verification time, the general purpose ZK-SNARK (808 nanoseconds) outperforms zkSVM (177 milliseconds). This is expected as the Groth16 approach used by ZoKrates has a big prover overhead and a trusted setup in exchange of minimal communication and verification time overhead. However, in zkSENSE's scenario, these verification times can be handled by the server, and instead, zkSVM makes the prover times rea-

sonable (as shown, the entire attestation takes a bit less than 3 seconds) and removes the need for trusted setup.

CPU and memory utilization: Figure 6 shows CPU usage per operation and device. Since no significant difference was observed between baseline and detection operation, we improve the figure visibility by reporting only one boxplot representative of both operations. The figure shows minimal CPU utilization associated with humanness classification and commitment operations. The counter-intuitive higher CPU usage at the S9 is due to the fact that this is a personal device with potential background activities from other apps. Even on the less powerful J3, committing only consumes about 12% of CPU (median value across devices) with peaks up to 45% on the J3. The ZKP generation is the most expensive operation, showing a median CPU consumption of about 15% and 30%, on respectively the S9 and J3. Overall, the CPU analysis suggests minimal impact of zkSENSE's operation and feasibility even on entry-level devices like the J3.

In our tests, we also collected memory usage of zkSENSE's via `procstats`. Detailed results are omitted since zkSENSE's memory consumption is negligible, i.e., less than 20MB regardless of device and operation. In comparison, the ZK-SNARK solution requires up to 1GB of memory due to the data generated during the trusted setup. This is quite limiting in presence of low-end devices which might not have that amount of free memory, requiring swapping and thus a further increase in execution time.

Battery consumption: Figure 7 reports the battery discharge (in mAh) associated with zkSENSE's key operations (detection, commitment, ZKP creation) for the J3 – given the S9 is a personal device, we were unable to wire its battery with the power meter. As expected from the previous results, the battery overhead imposed by zkSENSE's detection and committing operation is negligible. Further, ZKP computation only consumes about 5 mAh or 0.2% of the J3's battery (2,600 mAh). Even assuming one zkSENSE's humanness verification every hour of device usage, this would amount to under 1% of battery discharge for the average user (3h15min on average, with only 20% of users using the device more than 4h30min [76]).

Bandwidth consumption: Finally, we compute what is the bandwidth consumption required by the user to send a proof. The proof consists of the commitments of the difference vector, the average and standard deviation, for each of the input vectors the user submits. We use data from two sensors, namely the gyroscope and accelerometer, and for each sensor, we use three axis data. Moreover, we split each period into two segments as explained in Section 5.2. This results in a total of 12 input vectors. For every input vector, the proof consists of 14 KB (83 compressed points and 354 scalars). In our library, we implemented the trivial construction, where we build 12 of such proofs in parallel, resulting in 167 KB. However, there are ways to reduce this overhead, for which we provide the estimates. The opening and equality proofs we used (introduced in Section 3.3) could be improved, by implementing them using the techniques presented in [60]. This would reduce the complexity from linear to logarithmic, resulting in x3 improvement, with a size of the full proof of 56 KB. Finally, we could use the batching techniques for the range proofs, as presented in [59]. This would further reduce the size of the proof to 45 KB.

7.1 ZKSENSE Vs. reCAPTCHA

As a next step, we compare the performance of zkSENSE with the state-of-the-art privacy-preserving humanness attestation mechanism (i.e., visual CAPTCHA). To do so, we developed an Android app which embeds reCAPTCHA for Android [22]. The app is minimal[7] to ensure its performance evaluation covers only the CAPTCHA aspect rather than any extra components. For the same reason, we opted for Android reCAPTCHA rather than setting up a webpage with a CAPTCHA to solve. This alternative approach would require an Android browser for testing and the performance evaluation would be tainted by the extra cost of running a full browser. We did not evaluate Privacy Pass [77] for two reasons: 1) it currently requires a full browser along with an add-on, 2) it lacks support on mobile devices. Note that we do not expect critical performance difference between Privacy Pass and Android reCAPTCHA since they use a very similar strategy. Their difference instead lies in how invasive they are, both in how frequently they require user input and from a sensor data collection standpoint.

Using the above application we setup the following experiment. We enable remote access to the S9 device via the browser using Android screen mirroring [78] coupled with noVNC [79]. Then we asked 10 volunteers to visit the device from their browsers and solve CAPTCHAs as needed by the app. The app was coded such that users can continuously request for a new

7 Source code: https://github.com/svarvel/CaptchaTest

	zkSENSE	reCAPTCHA Automatic/Manual
Overall time	3	1.4/8.9 sec
CPU utilization	15%	3/15%
Memory utilization	20 MB	20 MB
Replay protection	No	Yes
Consumed bandwidth	167 KB	16 KB

Table 3. Performance of zkSENSE vs. Android reCAPTCHA.

CAPTCHA to solve. Note that Android reCAPTCHA, as reCAPTCHA v3, leverages client side behavior to minimize friction, i.e., whether to ask or not a user to solve a visual CAPTCHA like "click on all images containing a boat". It follows that often users do not need to solve any visual CAPTCHA. We label *automatic* all the samples we collected where our volunteers did not have to solve a CAPTCHA. We instead label *manual* all the samples where human interaction was needed. To increase the chance of showing an actual CAPTCHA, we created 15 CAPTCHAs that the app rotates on.

Over one week, our volunteers have requested about 500 CAPTCHAs with a 70/30 split: 350 automatic and 150 manual. Table 3 directly compares zkSENSE with Android reCAPTCHA with respect to: execution time, CPU and memory utilization, and bandwidth consumption. The median was reported for each metric, further differentiating between automatic and manual for Android reCAPTCHA. The table shows that zkSENSE adds about 1.6 seconds to the time required by Android reCAPTCHA when no user interaction is needed. zkSENSE instead saves a whole 6 seconds to the (median) user by never requiring any interaction. Even considering the fastest user in our experiment (5.4 seconds), zkSENSE is about 2x faster – and 10x faster than the slowest user (28.5 seconds). This is possible because zkSENSE removes the need of user interaction, at the cost of a higher risk for replay attacks. With respect to CPU and memory utilization, Table 3 shows that the two mechanisms are quite similar and both very lightweight. Bandwidth-wise, reCAPTCHA outperforms zkSENSE (16 versus 160KB), which communication overhead is still minimal and bearable even by devices with very little connectivity.

Table 3 currently reports on "overall time" which includes both the computation time and the time required to report to the server. While for Google we have no control on the server endpoint – located within 10ms in our experiments – in our experiments the server runs in the same LAN with 1-2 ms delay (negligible). We currently use HTTP (POST) + TLS1.3 to return the proof. For TCP, we use an unmodified kernel running Cubic with an initial window of 10packets (1500B MTU). Given our proof has a size of 160KB, the content delivery requires a worst case of: 1 RTT (for TCP) + 1 RTT (for TLS, in case of unknown server) + 3 RTT for TCP to transfer the data – assuming slow start (doubling of cwnd), aka 15K (10 MTU sized packets) + 30K + 60K + 60K (1/2 of the last cwnd available). This sums up to about 3/4 RTTs, which we could further reduce assuming a larger initial cwnd, or using QUIC – thus bringing the duration down to a maximum of 2RTTs. Assuming a CDN runs such a service, as Google does, this would thus cost between 20ms (with optimization) up to 100ms. Assuming a very bad client connection, e.g., on mobile with RTT of 150ms, then this would cost an extra 300 to 750ms.

7.2 Summary

Our experiments show that the general purpose ZK-SNARK is not a viable solution for mobile-based ZKP computation (600 sec and 2% battery drain on a low-end device). By designing our own model (i.e., zkSVM), we reduce ZKP's execution time by 10×, achieving a duration of a bit less than 3 sec and 0.2% battery drain. This execution time is comparable with today's visual CAPTCHA solving time, 9.8 sec on average [21]), thus making zkSENSE a serious competitor to state-of-the-art mechanisms for humanness attestation.

8. Related Work

Assessing humanness: To prevent automated programs from abusing online services, the widely adopted solution is to deploy a CAPTCHA system. However, text-based CAPTCHA schemes have been proven to be insecure as machines achieved 99.8% success rate in identifying distorted text [8, 80, 81]. Audio-based CAPTCHAs have also been used to assist visually impaired people, but they are difficult to solve, with over half of users failed during their first attempt [82]. Therefore, CAPTCHA service providers started to test image-based CAPTCHA schemes, which require users to select images that match given description [83]. Nevertheless, in [7, 84] authors demonstrated that more than 70% of image-based Google and Facebook CAPTCHAs can be efficiently solved using deep learning.

In [85], authors designed a multi-level data fusion algorithm, which combines scores from individual clicks to generate a robust human evidence. These CAPTCHA systems require users to perform additional tasks worsening user experience, especially on mobile devices [86]. ReCAPTCHA v2 uses a risk analysis engine to avoid interrupting users unnecessarily [87]. This engine col-

lects and analyses relevant data during click events. ReCAPTCHA v3 no longer requires users to click but instead it studies user interactions within a webpage and gives a score that represents the likelihood that a user is a human [88]. Although these CAPTCHA schemes are invisible to users, a plethora of sensitive data, including cookies, browser plugins, and JavaScript objects, is collected [89] that could be used to fingerprint the user.

With the proliferation of smartphones, various approaches leverage the variety of available sensors. Most of them require users to perform additional motion tasks. In [29], authors showed that waving gestures could be used to attest the intention of users. In [27], authors designed a bot detection system that asks users to tilt their device according to the description to prove they are human. In [28], authors presented a movement-based CAPTCHA scheme that requires users to perform certain gestures (e.g., hammering and fishing) using their device. In [90], authors exploited touch screen data during screen unlocking to authenticate users. In [91], authors suggested a brightness-based bot prevention mechanism that generates a sequence of circles with different brightness when typing a PIN; users will input misleading lie digits in circles with low brightness. In [92], authors proposed a behavioural-based authentication scheme, which uses timing and device motion information during password typing.

The work that is most closely related to ours is the Invisible CAPPCHA [19]. Similar to zkSENSE, authors leveraged the different device acceleration appearing on a finger touch and a software touch to make a decision about whether a user is a bot. However, Invisible CAPPCHA is not fully implemented as it requires a secure execution environment and its accuracy is low when device is stable on a table. In addition, it only considers simple tap and vibration events; its accuracy on more complicated touch events (e.g., drag, long press, and double tap) is unclear In comparison, zkSENSE considers any types of touch events and works regardless of the device movement. To improve the accuracy, zkSENSE uses more data sources in addition to accelerometer and introduces context into the detection.

Privacy-preserving and provable ML: A potential approach to offer privacy-preserving machine learning is to evaluate the model locally, without sending data to a server. However, if, unlike zkSENSE, such approach is taken without proving correct evaluation of the model, then verification may be lost [93–95]. In cases such as bot detection the user's interest might be of faking the evaluation model, and this may be vulnerable to user attacks. To the best of our knowledge, there are only 2 papers aiming to provide provable machine learning local evaluation without a trusted execution environment. The first one [96] tries to solve a similar problem, where personalization of a user device is done by evaluating a model locally on the user's machine. This work uses Bayesian classification, for which they need from 100-300 feature words. The generation of correct model evaluation for such range of feature words ranges from 30 to 80 seconds. Moreover, this study uses standard techniques for constructing zero-knowledge proofs, which give a big overhead to the verifier. For our particular use case (where the verifier may need to handle several requests simultaneously), such an overhead for the verifier is not acceptable. The second one [97] proposes a solution where after the evaluation of Random Forest and Hidden Markov models, the user generates a zero-knowledge proof of correct evaluation. However, this paper misses an evaluation study or availability of the code, which makes a study of the scalability of their approach inaccessible.

9. Conclusion

Service providers need a reliable way to attest whether a client is human or not and thus prevent user-side automation from abusing their services. Current solutions require (i) either additional user actions (e.g., sporadically solve mathematical quizzes or pattern recognition) like CAPTCHAs or (ii) user behavioral data to be sent to the server, thus raising significant privacy concerns.

In this paper, we propose zkSENSE: a novel continuous human attestation scheme, which is both (i) privacy-preserving and (ii) friction-less for the user. By leveraging the motion sensor outputs during touch screen events, zkSENSE performs human attestation at the edge, on the user's very own device, thus avoiding to transmit any sensitive information to a remote server. By enclosing the classification result in zero-knowledge proofs, zkSENSE guarantees the integrity of the attestation procedure. We tested our system under different attack scenarios: (i) when device is resting (on a table), (ii) when there is artificial movement from device's vibration, (iii) when the device is docked on a artificial movement generating swinging cradle, and it was able to detect if an action was triggered by a human with 92% accuracy. Performance evaluation of our Android prototype demonstrates that each attestation takes around 3 seconds, with minimal impact on both CPU and battery consumption.

Acknowledgments

We would like to thank our shepherd, Fan Zhang, and the anonymous reviewers for their suggestions to improve the paper. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Matthew Hughes. Bots drove nearly 40% of internet traffic last year - and the naughty ones are getting smarter. https://thenextweb.com/security/2019/04/17/bots-drove-nearly-40-of-internet-traffic-last-year-and-the-naughty-ones-are-getting-smarter/, 2019.

[2] Shailin Dhar Mikko Kotila, Ruben Cuevas Rumin. Compendium of ad fraud knowledge for media investors. https://www.wfanet.org/app/uploads/2017/04/WFA_Compendium_Of_Ad_Fraud_Knowledge.pdf, 2017.

[3] ThreatMetrix. H2 2018 cybercrime report. https://www.threatmetrix.com/info/h2-2018-cybercrime-report/, 2018.

[4] Drew Phillips. What is securimage? https://www.phpcaptcha.org/, 2015.

[5] Intuition Machines, Inc. hcaptcha: Earn money with a captcha. https://www.hcaptcha.com, 2019.

[6] Roberto Iriondo. Breaking captcha using machine learning in 0.05 seconds. https://medium.com/towards-artificial-intelligence/breaking-captcha-using-machine-learning-in-0-05-seconds-9feefb997694, 2018.

[7] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. I am robot:(deep) learning to break semantic image captchas. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 388–403. IEEE, 2016.

[8] Jeff Yan and Ahmad Salah El Ahmad. A Low-cost Attack on a Microsoft CAPTCHA. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554. ACM, 2008.

[9] Jarrod Overson. Bypassing captchas with headless chrome. https://medium.com/@jsoverson/bypassing-captchas-with-headless-chrome-93f294518337, 2018.

[10] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. uncaptcha: a low-resource defeat of recaptcha's audio challenge. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[11] Ruti Gafni and Idan Nagar. Captcha–security affecting user experience. *Issues in Informing Science and Information Technology*, 13:063–077, 2016.

[12] Josh Dzieza. Why captchas have gotten so difficult. https://www.theverge.com/2019/2/1/18205610/google-captcha-ai-robot-human-difficult-artificial-intelligence, 2019.

[13] Scott Hollier, Janina Sajka, Jason White, and Michael Cooper. Inaccessibility of captcha: Alternatives to visual turing tests on the web. https://www.w3.org/TR/turingtest/, 2019.

[14] Wei Liu. Introducing recaptcha v3: the new way to stop bots. https://webmasters.googleblog.com/2018/10/introducing-recaptcha-v3-new-way-to.html, 2018.

[15] FreePrivacyPolicy. Privacy policy for recaptcha. https://www.freeprivacypolicy.com/blog/recaptcha-privacy-policy/, 2019.

[16] Thomas Claburn. Google's recaptcha favors – you guessed it – google: Duh, only a bot would refuse to sign into the chocolate factory. https://www.theregister.co.uk/2019/06/28/google_recaptcha_favoring_google/, 2019.

[17] Katharine Schwab. Google's new recaptcha has a dark side. https://www.fastcompany.com/90369697/googles-new-recaptcha-has-a-dark-side, 2019.

[18] Ismail Akrout, Amal Feriani, and Mohamed Akrout. Hacking Google reCAPTCHA v3 using Reinforcement Learning. 2019.

[19] Meriem Guerar, Alessio Merlo, Mauro Migliardi, and Francesco Palmieri. Invisible CAPPCHA: A usable mechanism to distinguish between malware and humans on the mobile IoT. *Computers and Security*, 78:255–266, 2018.

[20] Muhammad Asim Jamshed, Wonho Kim, and KyoungSoo Park. Suppressing bot traffic with accurate human attestation. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, pages 43–48, 2010.

[21] Tim Allen. Having a captcha is killing your conversion rate. https://moz.com/blog/having-a-captcha-is-killing-your-conversion-rate, 2013.

[22] Google. Safetynet recaptcha api. https://developer.android.com/training/safetynet/recaptcha, 2021.

[23] Elie Bursztein, Steven Bethard, Celine Fabry, John C Mitchell, and Dan Jurafsky. How good are humans at solving captchas? a large scale evaluation. In *2010 IEEE symposium on security and privacy*, 2010.

[24] Ivan Enríquez. Why is captcha killing your conversion rate? https://blog.arengu.com/why-captcha-is-killing-your-conversion-rate/, 2019.

[25] Richard Kahn. How the use of captcha can hurt user experience. https://www.anura.io/blog/how-the-use-of-captcha-can-hurt-user-experience, 2020.

[26] Interaction Design Foundation. Killing the captcha for better ux. https://www.interaction-design.org/literature/article/killing-the-captcha-for-better-ux, 2016.

[27] Meriem Guerar, Mauro Migliardi, Alessio Merlo, Mohamed Benmohammed, and Belhadri Messabih. A completely automatic public physical test to tell computers and humans apart: A way to enhance authentication schemes in mobile devices. In *2015 International Conference on High Performance Computing & Simulation (HPCS)*, 2015.

[28] Thomas Hupperich, Katharina Krombholz, and Thorsten Holz. Sensor Captchas: On the Usability of Instrumenting Hardware Sensors to Prove Liveliness. In *International Conference on Trust and Trustworthy Computing*, 2016.

[29] Babins Shrestha, Nitesh Saxena, and Justin Harrison. Wave-to-Access: Protecting Sensitive Mobile Device Services via a Hand Waving Gesture. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security*, 2013.

[30] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. In *NDSS*, 2016.

[31] Jorge-L. Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. Transition-Aware Human Activity Recognition Using Smartphones. *Neurocomputing*, 2016.

[32] Rubén San-Segundo, Henrik Blunck, José Moreno-Pimentel, Allan Stisen, and Manuel Gil-Martín. Robust Human Activity Recognition using smartwatches and smartphones. *Engineering Applications of Artificial Intelligence*, 2018.

[33] Mohammad Malekzadeh, Richard G Clegg, Andrea Cavallaro, and Hamed Haddadi. Protecting Sensory Data Against Sensitive Inferences. In *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*, W-P2DS'18.

[34] Erhan Davarci, Betul Soysal, Imran Erguler, Sabri Orhun Aydin, Onur Dincer, and Emin Anarim. Age group detection using smartphone motion sensors. In *2017 25th European Signal Processing Conference (EUSIPCO)*, 2017.

[35] Jiexin Zhang, Alastair R Beresford, and Ian Sheret. SensorID: Sensor Calibration Fingerprinting for Smartphones. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (SP)*. IEEE, 5 2019.

[36] Elias P. Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P. Markatos. The long-standing privacy debate: Mobile websites vs mobile apps. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, 2017.

[37] World Wide Web Consortium (W3C). Captcha alternatives and thoughts. https://www.w3.org/WAI/GL/wiki/Captcha_Alternatives_and_thoughts, 2019.

[38] Web Accessibility In Mind (WebAIM). Screen reader user survey #7 results. https://webaim.org/projects/screenreadersurvey7/, 2017.

[39] Armin Sebastian. Buster: Captcha solver for humans. https://github.com/dessant/buster, 2019.

[40] Jennifer Tam, Jiri Simsa, Sean Hyde, and Luis V Ahn. Breaking audio captchas. In *Advances in Neural Information Processing Systems*, pages 1625–1632, 2009.

[41] Yuanxi Ou. What is shuabang, and should i be using it to promote my game? https://www.mobvista.com/en/blog/shuabang-using-promote-game/, 2018.

[42] Cristina Stefanova. Black hat aso for mobile apps & games: What is it and how it works (and why you shouldn't do it). https://thetool.io/2018/black-hat-aso, 2018.

[43] Gabriel Machuret. Blackhat aso news 2016: Shuabang – a notorious blackhat app store optimization provider in china. https://asoprofessional.com/blackhat-aso-news-2016-shuabang-a-notorious-blackhat-app-store-optimization-provider-in-china/, 2020.

[44] Brave Software, Inc. Get rewarded for browsing and support your favorite content creators. https://brave.com/brave-rewards/, 2019.

[45] Anton Kivva. The banker that can steal anything. https://securelist.com/the-banker-that-can-steal-anything/76101/, 2016.

[46] Mike Murray. Pegasus for android: the other side of the story emerges. https://blog.lookout.com/pegasus-android, 2017.

[47] Henry de Valence, Jack Grigg, George Tankersley, Filippo Valsorda, and Isis Lovecruft. The ristretto255 Group. Internet-Draft draft-hdevalence-cfrg-ristretto-01, Internet Engineering Task Force.

[48] Isis Agora Lovecruft and Henry de Valence. curve25519-dalek. https://crates.io/crates/curve25519-dalek, 2020.

[49] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, 1985.

[50] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, 1988.

[51] Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Goncalo Pestana, and Benjamin Livshits. Vpn0: A privacy-preserving decentralized virtual private network. In *Decentralising the Internet with IPFS and Filecoin*, DI2F'21, 2021.

[52] Eli Ben-sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. 2014.

[53] Nick Grosz. How icash protects delegate votes and identities in its proof of trust protocol. https://medium.com/@nickgrosz/how-icash-protects-votes-and-voter-identity-in-its-proof-of-trust-protocol-7a06c38e4296, 2018.

[54] Gonçalo Pestana, Iñigo Querejeta-Azurmendi, Panagiotis Papadopoulos, and Benjamin Livshits. Themis: Decentralized and trustless ad platform with reporting integrity. *arXiv preprint arXiv:2007.05556*, 2020.

[55] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *CRYPTO*, 1997.

[56] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology*, EUROCRYPT'16, 2016.

[57] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Report 2019/099, 2019. https://eprint.iacr.org/2019/099.

[58] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[59] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[60] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. Cryptology ePrint Archive, Report 2016/263, 2016. https://eprint.iacr.org/2016/263.

[61] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, 1991.

[62] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, 2009.

[63] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, 1999.

[64] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, 1987.

[65] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ICML'16, 2016.

[66] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. In *Lecture notes in computer science*, volume 7839, 2012.

[67] Joppe Bos, Kristin Lauter, and Michael Naehrig. Private Predictive Analysis on Encrypted Medical Data. Technical Report MSR-TR-2013-81, 9 2013.

[68] Android Developers. Android debug bridge (adb). https://developer.android.com/studio/command-line/adb, 2020.

[69] Henry de Valence, Cathie Yun, and Oleg Andreev. Bulletproofs. https://crates.io/crates/bulletproofs, 2020.

[70] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. CCS '93, 1993.

[71] ZoKrates community. Zokrates: A toolbox for zksnarks on ethereum. https://github.com/Zokrates/ZoKrates, 2019.

[72] str4d. Bellman: Zero-knowledge cryptography in rust. https://github.com/zkcrypto/bellman, 2016.

[73] Matteo Varvello, Kleomenis Katevas, Mihai Plesa, Hamed Haddadi, and Benjamin Livshits. BatteryLab: a distributed power monitoring platform for mobile devices. In *HotNets '19*, 2019.

[74] BatteryLab. A Distributed Platform for Battery Measurements. https://batterylab.dev, 2019.

[75] Monsoon Solutions Inc. High voltage power monitor. https://www.msoon.com/online-store/High-Voltage-Power-Monitor-Part-Number-AAA10F-p90002590, 2019.

[76] Jory Mackay. Screen time stats 2019: Here's how much you use your phone during the workday. https://blog.rescuetime.com/screen-time-stats-2018/, 2019.

[77] Alex Davidson. The privacy pass protocol. https://tools.ietf.org/html/draft-privacy-pass-00, 2019.

[78] Solly Ross Joel Martin, Samuel Mannehed and Pierre Ossman. Novnc: Html vnc client library and application. https://github.com/Genymobile/scrcpy, 2021.

[79] Novnc - the open source vnc client. https://github.com/novnc/noVNC, 2021.

[80] A A Chandavale, A M Sapkal, and R M Jalnekar. Algorithm to Break Visual CAPTCHA. In *2009 Second International Conference on Emerging Trends in Engineering Technology*, pages 258–262, 12 2009.

[81] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[82] Aimilia Tasidou, Pavlos S Efraimidis, Yannis Soupionis, Lilian Mitrou, and Vasilios Katos. User-centric, Privacy-Preserving Adaptation for VoIP CAPTCHA Challenges. 2012.

[83] Google. Are you a robot? Introducing "No CAPTCHA reCAPTCHA". https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html, 2014.

[84] Yuan Zhou, Zesun Yang, Chenxu Wang, and Matthew Boutell. Breaking Google reCaptcha V2. *J. Comput. Sci. Coll.*, 34(1):126–136, 10 2018.

[85] Chamila Walgampaya, Mehmed Kantardzic, and Roman Yampolskiy. Real time click fraud prevention using multi-level data fusion. In *Proceedings of the World Congress on Engineering and Computer Science*, 2010.

[86] Gerardo Reynaga and Sonia Chiasson. The usability of CAPTCHAs on smartphones. In *2013 International Conference on Security and Cryptography (SECRYPT)*, 2013.

[87] Google. Choosing the type of reCAPTCHA. https://developers.google.com/recaptcha/docs/versions, 2019.

[88] Google Developers. reCAPTCHA v3. https://developers.google.com/recaptcha/docs/v3, 2018.

[89] Lara O'Reilly. Google's new CAPTCHA security login raises 'legitimate privacy concerns'. https://www.businessinsider.com/google-no-captcha-adtruth-privacy-research-2015-2?r=US&IR=T, 2015.

[90] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch Me Once and I Know It's You!: Implicit Authentication Based on Touch Screen Patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, 2012.

[91] Meriem Guerar, Mauro Migliardi, Alessio Merlo, Mohamed Benmohammed, Francesco Palmieri, and Aniello Castiglione. Using screen brightness to improve security in mobile social network access. *IEEE Transactions on Dependable and Secure Computing*, 15(4):621–632, 2016.

[92] Attaullah Buriro, Sandeep Gupta, and Bruno Crispo. Evaluation of motion-based touch-typing biometrics for online banking. In *2017 International Conference of the Biometrics Special Interest Group (BIOSIG)*, pages 1–5. IEEE, 2017.

[93] Theja Tulabandhula, Shailesh Vaya, and Aritra Dhar. Privacy-preserving Targeted Advertising. *CoRR*, abs/1710.0, 2017.

[94] Mikhail Bilenko and Matthew Richardson. Predictive Client-side Profiles for Personalized Advertising. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 413–421, New York, NY, USA, 2011. ACM.

[95] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, 2011.

[96] Drew Davidson, Matt Fredrikson, and Benjamin Livshits. Morepriv: Mobile os support for application personalization and privacy. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, 2014.

[97] George Danezis, Markulf Kohlweiss, Benjamin Livshits, and Alfredo Rial. Private Client-side Profiling with Random Forests and Hidden Markov Models. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, 2012.

Appendix

A Sub-linear Inner Product Zero-Knowledge Proof

In this section, we prove that the zero-knowledge proof presented in section 3.3 provides perfect completeness, special honest verifier zero-knowledge and knowledge soundness. Firstly, we formally define these properties, which we reproduce as defined in [60].

A zero-knowledge proof system is defined by three probabilistic polynomial time algorithms, $(\mathcal{K}, \mathcal{P}, \mathcal{V})$, the generator, prover and verifier. The generator takes as input a security parameter written in unary form, 1^λ, and builds the common input of a proof, $pp \leftarrow \mathcal{K}(1^\lambda)$. In our paper, we use only common inputs that do not need to be honestly generated, meaning that the output of $\mathcal{K}$ can be publicly verified. The $\mathcal{P}$ and $\mathcal{V}$ algorithms take as input (pp, u, w) and (pp, u) respectively. We denote the interaction between prover and verifier, and the latter's output (0 if valid or 1 otherwise) by $\langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u) \rangle$. We consider relations $\mathcal{R}$ that consist of a three element tuple (pp, u, w), which we refer as the common input, instance and witness respectively. We define the set of all valid instances as $\mathcal{L}_\mathcal{R} = \{(pp, u) | \exists w : (pp, u, w) \in \mathcal{R}\}$. The protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is called a zero-knowledge proof system if it has perfect completeness, knowledge soundness and special honest-verifier zero-knowledge as described below. First, we introduce the notion of negligible function.

Definition 1 (Negligible function)**.** *A non-negative function $f : \mathbb{N} \to \mathbb{R}$ is called negligible if for every $\gamma \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that for all $k \geq k_0, f(k) \leq 1/k\gamma$.*

We now proceed with a formal definition of the properties a proof system needs to have to be considered a zero-knowledge proof.

Definition 2 (Perfect Completeness)**.** *A proof system is perfectly complete if for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(); (u, w) \leftarrow \mathcal{A}(pp) : \\ (pp, u, w) \notin \mathcal{R} \vee \langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u) \rangle = 1 \end{array}\right] = 1$$

Paraphrasing, this means that whenever prover and verifier proceed with the protocol, the verifier will always validate the proof.

Definition 3 (Knowledge soundness)**.** *A proof system has (strong black-box) computational knowledge soundness if for all DPT $\mathcal{P}^*$ there exists a PPT extractor $\mathcal{E}$ such that for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\lambda); (u, s) \leftarrow \mathcal{A}(pp); \\ w \leftarrow \mathcal{E}^{\langle \mathcal{P}^*(s) || \mathcal{V}(pp,u) \rangle}(pp, u) : \\ b = 1 \wedge (pp, u, w) \notin \mathcal{R} \end{array}\right]$$

is negligible with respect to λ.

Here the oracle $\langle \mathcal{P}^*(s) || \mathcal{V}(pp, u) \rangle$ runs a full protocol execution from the state s, and if the proof is successful it returns a transcript of the prover's communication. The extractor $\mathcal{E}$ can ask the oracle to rewind the proof to any point and execute the proof again from this point on with fresh challenges from the verifier. We define $b \in 0, 1$ to be the verifier's output in the first oracle execution, i.e., whether it accepts or not, and we think of s as the state of the prover. If the definition holds also for unbounded $\mathcal{P}^*$ and $\mathcal{A}$ we say the proof has statistical knowledge soundness. The definition can then be paraphrased as saying that if the prover in state s makes a convincing proof, then we can extract a witness.

Definition 4 (Special Honest-Verifier Zero-Knowledge)**.** *A proof system is computationally special honest-verifier zero-knowledge (SHVZK) if there exists a PPT simulator S such that for all state-full interactive PPT adversaries $\mathcal{A}$ that output (u, w) such that $(pp, u, w) \in \mathcal{R}$ and randomness ϕ for the verifier*

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\lambda); (u, w, \phi) \leftarrow \mathcal{A}(pp); \\ \mathtt{view}_\mathcal{V} \leftarrow \langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u, \phi) \rangle : \\ \mathcal{A}(\mathtt{view}_\mathcal{V}) = 1 \end{array}\right] -$$

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\lambda); (u, w, \phi) \leftarrow \mathcal{A}(pp); \\ \mathtt{view}_\mathcal{V} \leftarrow S(pp, u, \phi) : \\ \mathcal{A}(\mathtt{view}_\mathcal{V}) = 1 \end{array}\right]$$

is negligible with respect to λ. We say the proof is statistically SHVZK if the definition holds against unbounded adversaries, and perfect SHVZK if the probabilities are equal.

This definition can be paraphrased as saying that for every valid protocol run, a simulator can generate simulated random view which is indistinguishable from the original run. Having formalised the properties, we proceed with the proof of Theorem 1:

Proof. We follow the lines of the proof presented in the work of Bünz et al. [59] to complete our proof. Completeness is trivial. To prove special honest verifier zero-knowledge we construct a simulator that generates valid

statements which are indistinguishable from random. To this end, the simulator acts as follows:

$$\begin{aligned}\mathcal{C} &\in_R \mathbb{Z}_p^* \\ \boldsymbol{l}, \boldsymbol{r} &\in_R \mathbb{Z}_p^n \\ \hat{t}, \tau_{\mathcal{C}}, \mu &\in_R \mathbb{Z}_p \\ T_2 &\in_R \mathbb{G} \\ T_1 &= \left(g^{\hat{t}} h^{\tau_{\mathcal{C}}} \cdot V^{-1} \cdot T_2^{-\mathcal{C}^2}\right)^{1/\mathcal{C}} \\ S &= \left(h^{\mu} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot \boldsymbol{h}^{\boldsymbol{r}} \cdot A^{-1}\right)^{1/\mathcal{C}}\end{aligned}$$

We can see that the simulated transcript,

$$(S, T_1, T_2; \mathcal{C}; \boldsymbol{l}, \boldsymbol{r}, \hat{t}, \tau_{\mathcal{C}}, \mu),$$

all elements except for S and T_1 are random, and the latter two are computationally indistinguishable from random due to the DDH assumption. Also, note that S and T_1 are generated following the verification equations, hence this simulated conversation is valid.

Next, we construct an extractor to prove knowledge soundness. The extractor runs the prover with three different values of the challenge $\mathcal{C}$, ending with the following valid proof transcript:

$$\begin{aligned}&(S, T_1, T_2; \mathcal{C}'; \boldsymbol{l}', \boldsymbol{r}', \hat{t}', \tau_{\mathcal{C}}', \mu') \\ &(S, T_1, T_2; \mathcal{C}''; \boldsymbol{l}'', \boldsymbol{r}'', \hat{t}'', \tau_{\mathcal{C}}'', \mu'') \\ &(S, T_1, T_2; \mathcal{C}'''; \boldsymbol{l}''', \boldsymbol{r}''', \hat{t}''', \tau_{\mathcal{C}}''', \mu''').\end{aligned}$$

Additionally, the extractor invokes the extractor of the inner product argument. This extractor is proved to exist in the original paper of Bunz et al. [59]. For each proof transcript, the extractor first runs the inner product extractor, to get a witness $\boldsymbol{l}, \boldsymbol{r}$ to the inner product argument such that $P = h^{\mu}\boldsymbol{g}^{\boldsymbol{l}}\boldsymbol{h}^{\boldsymbol{r}} \wedge \langle \boldsymbol{l}, \boldsymbol{r} \rangle = \hat{t}$. With this witness, and using two valid transcripts, one can compute linear combinations of (16), we can compute $\alpha, \rho, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{s}_L, \boldsymbol{s}_R$ such that $A = h^{\alpha}\boldsymbol{g}^{\boldsymbol{a}}\boldsymbol{h}^{\boldsymbol{b}}$ and $S = h^{\rho}\boldsymbol{g}^{\boldsymbol{s}_L}\boldsymbol{h}^{\boldsymbol{s}_R}$. Such an extraction of these values proceeds as follows:

$$A \cdot S^{\mathcal{C}'} = h^{\mu'}\boldsymbol{g}^{\boldsymbol{l}'}\boldsymbol{h}^{\boldsymbol{r}'} \qquad A \cdot S^{\mathcal{C}''} = h^{\mu''}\boldsymbol{g}^{\boldsymbol{l}''}\boldsymbol{h}^{\boldsymbol{r}''}.$$

Combining both relations we have

$$S = \left(h^{\mu' - \mu''}\boldsymbol{g}^{\boldsymbol{l}' - \boldsymbol{l}''}\boldsymbol{h}^{\boldsymbol{r}' - \boldsymbol{r}''}\right)^{\frac{1}{\mathcal{C}' - \mathcal{C}''}}.$$

The extraction of A follows.

Using these representations of A and S, as well as $\boldsymbol{l}^i$ and $\boldsymbol{r}^i$ with $i \in \{', '', '''\}$, we have that

$$\begin{aligned}\boldsymbol{l}^i &= \boldsymbol{a} + \boldsymbol{s}_L \mathcal{C}^i \\ \boldsymbol{r}^i &= \boldsymbol{b} + \boldsymbol{s}_R \mathcal{C}^i.\end{aligned}$$

If these do not hold for all challenges, then the prover has found two distinct representations of the same group element, yielding a non-trivial discrete logarithm relation.

Next, we extract the values T_i with $i \in \{1, 2\}$ from equation (14) as follows:

$$\begin{aligned}g^{\hat{t}'} h^{\tau_{\mathcal{C}}'} &= V \cdot T_1^{\mathcal{C}'} \cdot T_2^{\mathcal{C}'^2} \\ g^{\hat{t}''} h^{\tau_{\mathcal{C}}''} &= V \cdot T_1^{\mathcal{C}''} \cdot T_2^{\mathcal{C}''^2} \\ g^{\hat{t}'''} h^{\tau_{\mathcal{C}}'''} &= V \cdot T_1^{\mathcal{C}'''} \cdot T_2^{\mathcal{C}'''^2}\end{aligned}$$

which we can combine to get the following representation of T_2:

$$T_2 = \left(\left(g^L h^R\right)^{\frac{1}{\mathcal{C}' - \mathcal{C}'''}}\right)^{\frac{\mathcal{C}' + \mathcal{C}'''}{\mathcal{C}' + \mathcal{C}''}}$$

with $L = \frac{\hat{t}' - \hat{t}''}{\mathcal{C}' - \mathcal{C}''} - \frac{\hat{t}' - \hat{t}'''}{\mathcal{C}' - \mathcal{C}'''}$ and $R = \frac{\tau_{\mathcal{C}}' - \tau_{\mathcal{C}}''}{\mathcal{C}' - \mathcal{C}''} - \frac{\tau_{\mathcal{C}}' - \tau_{\mathcal{C}}'''}{\mathcal{C}' - \mathcal{C}'''}$. Extractions of T_1 and V follow.

If for any transcript, we have that

$$\hat{t}^i \neq c + t_1 \mathcal{C}^i + t_2 \mathcal{C}^{i2}$$

with $i \in \{', '', '''\}$, then the extractor has again found a non trivial discrete logarithm relation. Let $P(X) = \langle l(X), r(X) \rangle$. Due to the validity of the transcripts, we have that $P(X)$ equals $t(X) = c + t_1 X + t_2 X^2$ at least in the different challenges. In other words, the polynomial $P(X) - t(X)$ has at least three roots, and is of degree 2, hence it must be the zero polynomial. Therefore, we have that $t(X) = P(X)$. This implies that the zero coefficient of $t(X)$, namely c, equals $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ and we have a valid witness for the statement. ■

B Formal Analysis of zkSVM

In this section we formally define the properties we expect out of zkSVM, privacy and verifiability. We use game based proofs to show that zkSVM indeed provides these properties. We formally define them here, and include the proofs subsequently. To model the experiments of privacy and verifiability, we define the following five functions:

- `Setup`(λ): Which is defined exactly as in Procedure 1. We omit the notation of the cryptographic material, and consider it implicit. We represent the set of parameters of the SVM model (normalisation mean, normalisation scale, SVM weight and SVM intercept) by W.
- `GenProof`($\mathbf{v}$): Generates the zkSVM proof by running Procedure 2, 3, 4, and 5. Mainly, it runs all steps of the proof except for the submission step.

We simplify the representation of the resulting tuple by $[S_H, \Theta, Score, r_R]$, where Θ consists of all intermediate proofs and commitments.

- `SubmitReq`$([S_H, \Theta, Score, r_R])$: Submits the output of `GenProof`$(\mathbf{v})$ by sending it to the verifier (Procedure 6).
- `VerifReq`$([S_H, \Theta, Score, r_R])$: Runs all procedures defined in the Verification phase of zkSVM, mainly Procedures 7, 8, 9 and 10.
- `EvalSVM`$(\mathbf{v})$: Generates the result, *Score*, corresponding to $\mathbf{v}$, as defined in the zkSVM proof, but excluding the cryptographic mechanisms.

B1 Privacy

The goal of zkSVM is that no information is leaked from the input vector other than the result of the SVM model. To model this, in Figure 8 we define an experiment, $\mathtt{Exp}^{\mathtt{priv},b}_{\mathcal{A},\mathcal{C}}$, between an adversary, $\mathcal{A}$, and a challenger, $\mathcal{C}$. The latter chooses a bit $b \in \{0,1\}$, uniformly at random, which is given as input to the experiment. The adversary controls the zkSVM verifier, and its goal is to distinguish the submissions of two different input vectors. The adversary is given access to an oracle, $\mathcal{O}$`Submit`(), which takes as input two vectors of size n, runs `GenProof`() over them, and submits a result. Note that it is the adversary who chooses the vectors over which the zkSVM is executed and may modify the weights of the SVM model outputted by `Setup`(). Therefore, this experiment models the malicious choice of the SVM parameters, as well as any possible choice of input vector. Depending on the bit, b, the oracle submits the result of one vector or the other, by running `SubmitReq`(). However, to avoid a trivial win by the adversary, the submitted SVM score is always computed over the first vector. Hence, the experiment simulates the proof of the second vector by running `SimResult`(). The adversary may call this oracle as many times as it wishes. By the end of the experiment, the adversary outputs a bit, $b' \in \{0,1\}$. The adversary wins if $b' = b$ with non-negligible probability with respect to the security parameter, λ.

Theorem 2. *There exists a* `SimResult` *algorithm, such that no PPT adversary can win the* zkSVM *privacy experiment with colluding verifier with probability non-negligibly better than 1/2 with respect to* λ.

Proof. To prove that zkSVM provides privacy we proceed by a series of games. We start with the adversary playing the privacy experiment with $b = 0$, and after a sequence of game step transitions, the adversary finishes playing the ballot privacy experiment with $b = 1$. We argue that each of these steps are indistinguishable, and therefore the results follows. The proof proceeds along the following sequence of games:

$\mathtt{Exp}^{\mathtt{priv},b}_{\mathcal{A},\mathcal{C}}(\lambda)$:
  $\mathsf{W} \leftarrow$ `Setup`(λ)
  $b' \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Submit}}(W)$
  Output b'

$\mathcal{O}$`Submit`(v_1, v_2):
  Let $[S^1_H, \Theta^1, Score^1, r^1_R] \leftarrow$ `GenProof`(v_1, W)
  Let $[S^2_H, \Theta^2, Score^2, r^2_R] \leftarrow$ `GenProof`(v_2, W)
  Let $\Theta^2 \leftarrow$ `SimResult`$(S^2_H, Score^1, r^1_R)$
  return `SubmitReq`$([S^b_H, \Theta^b, Score^1, r^1_R])$

Fig. 8. In the privacy experiment $\mathtt{Exp}^{\mathtt{priv},b}_{\mathcal{A},\mathcal{C}}$, the adversary $\mathcal{A}$ has access to the oracle $\mathcal{O}$`Submit` and controls zkSVM verifier.

Game $\mathbf{G}_0$ Let game $\mathbf{G}_0$ be the $\mathtt{Exp}^{\mathtt{priv},0}_{\mathcal{A},\mathcal{C}}(\lambda)$ game (see Figure 8).

Game $\mathbf{G}_1$ Game $\mathbf{G}_1$ is as in $\mathbf{G}_0$, but now $\mathcal{O}$`Submit` always computes a simulation of Θ regardless of the bit. Mainly, $\mathcal{O}$`Submit` proceeds as follows:

$\mathcal{O}$`Submit`(v_1, v_2):
  Let $[S^1_H, \Theta^1, Score^1, r^1_R] \leftarrow$ `GenProof`(v_1, W)
  Let $[S^2_H, \Theta^2, Score^2, r^2_R] \leftarrow$ `GenProof`(v_2, W)
  Let $\Theta^1 \leftarrow$ `SimResult`$(S^1_H, Score^1, r^1_R)$
  Let $\Theta^2 \leftarrow$ `SimResult`$(S^2_H, Score^1, r^1_R)$
  return `SubmitReq`$([S^1_H, \Theta^1, Score^1, r^1_R])$

The function `SimResult` proceeds by simulating all zero knowledge proofs contained in Θ. Because all these proofs are Zero Knowledge Proofs, and hence have the Special Honest-Verifier Zero-Knowledge property, there exists a simulation algorithm such that $\mathcal{A}$ cannot distinguish between a real and a simulated proof. Note that at this point of the experiment, the commitment S_H and all commitments in Θ correspond to those of v_1—only the zero knowledge proofs in Θ are simulated.

Game $\mathbf{G}_2$ Game $\mathbf{G}_2$ is as in $\mathbf{G}_1$, but now, instead of returning

$$\mathtt{SubmitReq}([S^1_H, \Theta^1, Score^1, r^1_R]),$$

the oracle $\mathcal{O}$`Submit` returns

$$\mathtt{SubmitReq}([S^2_H, \Theta^2, Score^1, r^1_R]).$$

In $\mathbf{G}_2$ the view of the adversary is identical to the one of $\mathtt{Exp}^{\mathtt{priv},1}_{\mathcal{A},\mathcal{C}}(\lambda)$. Only thing that remains is to prove that $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable

$\texttt{Exp}^{\texttt{verif}}_{\mathcal{A},\mathcal{C}}(\lambda)$:
W$\leftarrow$ $\texttt{Setup}(\lambda)$
$[\mathbf{v}, S_H, \Theta, Score, r_R] \leftarrow \mathcal{A}(W)$
If $\texttt{EvalSVM}(\mathbf{v}) = Score$ return 0
If $\texttt{VerifReq}([S_H, \Theta, Score, r_R]) = \perp$
then return 0, else return 1.

Fig. 9. In the verifiability experiment $\texttt{Exp}^{\texttt{verif}}_{b,\mathcal{A}}$, the adversary $\mathcal{A}$ needs to submit a result not corresponding to the committed vector.

Given the Special Honest-Verifier Zero-Knowledge property of the proofs, we know that the simulated view is random. Given that both simulations are equally distributed, it is infeasible to distinguish between Θ^1 and Θ^2. Similarly, given the perfectly hiding property of Pedersen commitments, no adversary can distinguish between S^1_H and S^2_H. Clearly the resulting view is independent of b. And privacy follows. ■

B2 Verifiability

The other goal of zkSVM is that an adversary cannot convince a verifier that the result is not linked to the committed vector as defined by the protocol. To model this, in Figure 9 we define an experiment, $\texttt{Exp}^{\texttt{verif}}_{\mathcal{A},\mathcal{C}}$, between an adversary, $\mathcal{A}$, and a challenger, $\mathcal{C}$. Informally, verifiability ensures that a result that is not the outcome of the model evaluation over the committed vector cannot have a valid proof. During the experiment the adversary has access to the weights of the model, and generates an input vector, a result, and its corresponding proof material. The adversary wins the experiment if the result does not correspond the the SVM execution of the committed vector, and the verifier validates. We formally describe the experiment in Figure 9.

Theorem 3. *No PPT adversary can win the* zkSVM *verifiability experiment with non-negligible probability with respect to* λ.

Proof. At the end of the proof generation procedure, the prover (in our case the adversary) outputs a commitment, S_H, a tuple of intermediate cryptographic material, Θ, and a score, $Score$, together with the randomness associated to the commitment of the score, r_R. The result follows from the soundness property of ZKPs and the binding property of Pedersen commitments. Let us extend the cryptographic material associated with the vector. We have that $\Theta = [\Delta, \mathrm{M}, \mathrm{M}', \Lambda, \Lambda']$, with

$$\begin{aligned}
\Delta &= \left[Diff, S^{iter}_H, \Pi_{Eq}, \Pi_0\right], \\
\mathrm{M} &= \left[Avg, \Pi^{\mu}_{IP}\right], \\
\mathrm{M}' &= \left[Avg', \Pi^{\mu}_{IP}{}'\right], \\
\Lambda &= \left[G^{\tilde{\mu}}, H^{\tilde{\mu}}, H_S, \Pi^G_{Eq}, \Pi^H_{Eq}, \Pi^S_{Eq}, \right. \\
&\qquad \left. Var, \Pi^{\sigma^2}_{IP}, Std, \Pi^{\sigma}_{sqrt}\right], \\
\Lambda' &= \left[G^{\tilde{\mu}}{}', H^{\tilde{\mu}}{}', H_S{}', \Pi^G_{Eq}{}', \Pi^H_{Eq}{}', \right. \\
&\qquad \left. \Pi^S_{Eq}{}', Var', \Pi^{\sigma^2}_{IP}{}', Std', \Pi^{\sigma}_{sqrt}{}'\right].
\end{aligned}$$

The proof verifies all proofs. In particular

- Procedure 7 first generates the iterated generators, and checks that indeed S_H and S^{iter}_H commit to the same opening, by verifying Π_{Eq}. Then, it checks that $Diff$ commits to the difference of S_H and S^{iter}_H, in all entries but the last, which contains a zero, by verifying Π_0.
- Procedure 8 checks that Avg commits to the sum of the elements committed in S_H by verifying Π^{μ}_{IP}. It does the same for the difference vector.
- Procedure 9 first checks that $G^{\tilde{\mu}}$ and $H^{\tilde{\mu}}$ commit to the sum committed in Avg by verifying Π^G_{Eq} and Π^H_{Eq} respectively. Then, it checks that H_S commits to the same values as S_H by verifying Π^S_{Eq}. With these verified commitments, the verifier can check that Var commits to $N^2\langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu}\rangle$, in other words, that it commits to a factor of the variance. It does so by running the algebraic operations of Equation (23), and verifying $\Pi^{\sigma^2}_{IP}$. Finally, to check that Std commits to the standard deviation of the vector committed in S_H, it simply needs to check that it contains the square root of Var. It does so by verifying Π^{σ}_{sqrt}. The same is performed for the difference vector.

Given that all these proofs are Zero Knowledge Proofs, which provide the knowledge soundness property, the commitments of the SVM features, Avg, Avg', Std, Std', do not contain the expected features of the vector committed in S_H with negligible probability. It only remains to prove that the result indeed corresponds to the SVM function executed with these features as inputs.

- Procedure 10 first leverages the homomorphic properties of the commitment scheme. In this way, the verifier obtains a commitment of the linear combination of the values committed in Avg, Avg', Std, Std'. This results in the commitment of the result, as described in Equation (22). This operation is con-

ducted solely by the verifier, avoiding any possible attacks by the adversary. Finally, the verifier checks that the submitted score, and the corresponding opening key, indeed correspond to the locally computed commitment. Only if this is true, the verifier validates.

Given that the commitment scheme used in zkSVM is computationally binding, a PPT adversary has no more than negligible probability of submitting an opening that does not correspond to the committed value.

Given that `EvalSVM` and the proof of zkSVM compute the exact same operations over the input vector, the result that the adversary has no more than negligible probability of wining the verifiability experiment follows. ■