UNIVERSITY OF CALIFORNIA,
IRVINE

Privacy in Emerging Technologies

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Ercan Ozturk

Dissertation Committee:
Professor Gene Tsudik, Chair
Professor Qi Alfred Chen
Dr. Andrew Paverd

2021

DEDICATION

To my grandparents, Arife and Bekir, and to my dad, Hayretdin Ozturk...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

A PhD is an endeavor with many facets, as I have learned, and it would not have been possible for me to complete this journey without the help and support of many.

First of all, I would like to thank my advisor, Gene Tsudik, from the bottom of my heart for giving me the chance to embark on this journey. Without his guidance, encouragement and support throughout my PhD, I would not be able to conduct rigorous research and navigate the many challenges one faces on this adventure.

I would also like to thank my defense committee, Gene Tsudik, Qi Alfred Chen and Andrew Paverd for their insightful comments and being an inspiration to me with the awe inspiring research they conduct.

I also owe my sincerest gratitude to all my teachers along my education; especially to Ali Aydin Selcuk for introducing me to the field of Security and to Ismail Kemal Sunel for sparking my interest in learning and exploring.

I would also like to thank my coauthors – with whom I have shared many deadlines, and labmates at the SPROUT group at UC Irvine: Tyler Kaczmarek, Tatiana Bradley, Christopher Wood, Norrathep Rattanavipanon, Xavier Carpent, Ivan De Oliveira Nunes, Pier Paolo Tricomi, Andrew Paverd, Seoyeon Hwang, Sashidhar Jakkamsetti, Ai Enkoji, Esmerald Aliaj, Ben Terner, Renascence Tarafder Prapty and Youngil Kim; specifically Yoshimichi Nakatsuka and Andrew Searles for their constructive comments on a previous version of this dissertation and its presentation. Rock on!

I am extremely grateful for my family and friends. They have always been there for me through thick and thin – even if our timezones did not always match.

It is only fair that I end this chapter with a quotation from Edgar Allan Poe which perplexed me when I first heard years back starting my PhD: “Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.”.

This dissertation contains the results of years of research conducted at UC Irvine. During that time, I was supported by a combination of teaching and research assistanships.

VITA

Ercan Ozturk

EDUCATION

Doctor of Philosophy in Computer Science	**2021**
University of California, Irvine	*Irvine, CA, USA*
Master of Science in Computer Science	**2021**
University of California, Irvine	*Irvine, CA, USA*
Bachelor of Science in Computer Engineering	**2016**
TOBB University of Economics and Technology	*Ankara, Turkey*
Bachelor of Science in Math (Minor)	**2016**
TOBB University of Economics and Technology	*Ankara, Turkey*

RESEARCH EXPERIENCE

Graduate Research Assistant	**2018–2021**
University of California, Irvine	*Irvine, CA, USA*
Researcher Intern	**Summer, 2019**
Microsoft Research	*Cambridge, UK*

TEACHING EXPERIENCE

TA for Computer and Network Security (CS134)	**Spring, 2019**
TA for Computer Networks (CS132)	**Fall, 2018**
TA for Introduction to Programming (ICS31)	**Spring, 2018**
Reader for Computer and Network Security (CS134)	**Winter, 2018**
TA for Programming in Java as a Second Language (ICS45J)	**Fall, 2017**
Reader for Introduction to C++ (ICS45C)	**Winter and Spring, 2017**
Reader for Computer and Network Security (CS134)	**Fall, 2016**
University of California, Irvine	*Irvine, CA, USA*

PROFESSIONAL EXPERIENCE

Software Engineer Intern	**Summer, 2020**
Facebook	*Menlo Park, CA, USA*
Software Engineer Intern	**Summer, 2018**
Google	*Mountain View, CA, USA*
Summer Intern	**Summer, 2017**
Yahoo	*Sunnyvale, CA, USA*

REFEREED CONFERENCE PUBLICATIONS

CACTI: Captcha Avoidance via Client-side TEE Integration **2021**
30th USENIX Security Symposium (USENIX Security 21)

Balancing Security and Privacy in Genomic Range Queries **2019**
Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society

Thermanator: Thermal Residue-Based Post Factum Attacks on Keyboard Data Entry **2019**
Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security

Assentication: User De-authentication and Lunchtime Attack Mitigation with Seated Posture Biometric **2018**
International Conference on Applied Cryptography and Network Security. Springer, Cham

PAPERS IN SUBMISSION OR UNDER REVIEW

VICEROY: GDPR-/CCPA-compliant Enforcement of Verifiable Accountless Consumer Requests **2022**
Network and Distributed System Security Symposium (NDSS)

ROSEN: RObust and SElective Non-repudiation (for TLS) **2021**
ACM SIGSAC Conference on Cloud Computing Security Workshop

Thermal (and Hybrid Thermal/Audio) Side-Channel Attacks onKeyboard Input **2021**
Computers & Security

ABSTRACT OF THE DISSERTATION

Privacy in Emerging Technologies

By

Ercan Ozturk

Doctor of Philosophy in Computer Science

University of California, Irvine, 2021

Professor Gene Tsudik, Chair

The importance of privacy has been growing steadily for over 25 years. Increasingly popular areas (such as IoT, cryptocurrencies and genomics) have attracted and fueled new types of privacy-focused attacks and exploits. This dissertation focuses on the lifecycle of secrets (or data) – from initial entry to use, to present attacks and defenses that utilize emerging technologies.

We start by presenting a side-channel attack that targets password entry. This attack uses thermal residues (that results from human fingertips touching the keyboard) to recover recently entered passwords on external keyboards. Then, we present a privacy-preserving CAPTCHA alternative that mimics the rate-limiting nature of CAPTCHAs. To skip CAPTCHAs, clients generate rate-proofs when the rate at which they have performed an action (e.g., visit a website, sign up for an email account) is below a server-supplied threshold. Rate-proofs, generated by client-side Trusted Execution Environments (TEEs), assure servers that clients are not acting in an abusive manner. We also propose a scalable data ownership framework in which clients with no accounts on a website can prove ownership of data collected from them. Although data ownership proofs are possible using traditional authentication methods (e.g., passwords), there is no accepted way of achieving this for acountless clients. This framework completes the missing piece of verifiable consumer requests which are used to exercise data rights (access/modify/delete) granted by recent data protection regulations such as GDPR and CCPA. A client-side TEE can be used to store a secret that can initiate these requests. The use of TEEs, as shown in these two work, allows us to secure and privacy-protect secrets/data after entry.

Lastly, we present a cryptograpy-based solution for range queries in the genomics domain. This ensures the authenticity and integrity of the genome of the individual while minimizing the exposed data to testers. It uses a variety of techniques ranging from zero-knowledge range proofs and digital signatures to continual linking of elements inspired by literature on range queries on databases. We use the genomics domain to show how privacy can be achieved if there are no TEEs.

Chapter 1

Introduction

The growing popularity of big data [6, 61] amplified by swift adoption of various "smart" devices that collect information on a myriad of activities has led to the parallel growth of privacy concerns. Today, more than 60% of Americans think that data collection is incessant and more than 80% consider that the risks outweigh the benefits [1]. Although recent legislations, such as GDPR [20] and CCPA [13], regulate how personal data are handled, it is very important for current and emerging technologies to support privacy by default to prevent (or at least reduce) data exposure.

To this end, there is a need to reconcile privacy with functionality such that data exposure *is* minimized *while* offering desired functionality *without* encumbering users. Achieving an acceptable balance of privacy, functionality and usability is challenging due to numerous functional requirements and constraints for various domains. The goal of this dissertation is to construct privacy-agile approaches for representative areas with varying degrees of technology penetration.

As noted by Diffie and Hellman [102], privacy and authentication are closely related. Users authenticate, on average, 45 times in a day and 74.4% of these authentication events are a form of "something one knows" (e.g., passwords, PINs) [142]. Ubiquitous use of passwords for authentication make them prime targets for attackers. Although simple attacks, such as shoulder surfing, are easy to mount, they are also easy to detect and prevent. Whereas, lower level side-channel attacks utilize information leakage from flawed implementations of authentication mechanisms. Combined with insider threats and emerging technologies that enable novel attacks, study of side-channel attacks require utmost scrutiny to be up-to-date with emerging dangers.

However, thwarting side-channel attacks alone is insufficient to protect a secret. There is a need for a secure channel between the external password entry medium (e.g., a keyboard) and the authentication software. Otherwise, more privileged software, such as the operating system, may interfere. One mitigation approach requires secure hardware – a common type of which is Trusted Execution Environments (TEEs) [160].

Popular TEEs, such as Intel SGX [33], focus on isolated execution where no other software can observe the runtime state of a program. This allows TEEs to improve the privacy of existing systems. With the increasing availability of TEEs on client devices, many applications can be security- and privacy-enhanced.

However, for certain fields, TEEs are not available or not commonly used. In such domains more traditional techniques, such as cryptography, are the norm for protecting data. Crypto[1]-based approaches can be used to provide authenticity and integrity of data along with minimal exposure. Study of such techniques allow us to cover more representative areas.

1.0.1 Contributions

The goal of this dissertation is to utilize emerging or previously niche technologies to improve the privacy of various applications. This work makes the following contributions:

[1]By "crypto", we always refer to cryptography, not to cryptocurrencies.

- We explore how an emerging side-channel with a post-factum adversary model can be used to recover passwords. This attack utilizes thermal residues to target passwords entered using external keyboards. The goal is to recover passwords that can later be used to access unauthorized data, thus violate the privacy of users. We also propose mitigation strategies.

- Privacy of Internet users are threatened by extensive data collection. With the increasing availability of TEEs, current Web services can be replaced with privacy-preserving versions that impose less burden on the users. For instance, currently CAPTCHAs serve as rate-limiters due to easy access to massive CAPTCHA solving services and advances in machine learning. Furthermore, CAPTCHAs recently started a transition from hard-for-machines-to-solve problems to more data-collection based solutions. We design a privacy-preserving mechanism to substitute/complement CAPTCHAs with rate-limiters implemented in TEEs.

- Recent data protection regulations, such as GDPR and CCPA, grant various rights to clients regarding the data collected about them. Exercising these rights however requires authentication/verification of clients which poses a challenge for clients with no accounts. By utilizing TEEs and minimal secret storage, we design and implement a scalable, easy-to-use and privacy preserving mechanism for generating verifiable consumer requests. Our approach can also be used for clients with accounts to further improve security.

- For some domains, including genomics, cryptography is still the main method to provide security and privacy. Utilizing cryptography, we propose a range query protocol that balances security (authentication and integrity) and privacy of genomic material.

Chapter 2

Thermal Residue-Based Post Factum Attacks on Keyboard Data Entry

2.1 Introduction

As famously noted by Diffie and Hellman [102], authentication and privacy are closely related. To ensure privacy, access must be restricted to users with appropriate authorizations. But first, users need to prove that "they are who they say they are". This step constitutes authentication.

There are three types of authentication: *something one knows*, *something one has* and *something one is*. The most widely used form of authentication is via passwords; an example of *something one knows*. Although they are the most common, passwords suffer from insecure selection and reuse [170]. This problem is exacerbated by insider threats [27]. Simple examples, such as shoulder surfing, can be very effective [21]; however, easy prevention is possible by being more careful of one's surroundings.

Side-channels on the other hand are more on the side of a "hidden" threat. They threaten privacy at the time of data entry – or shortly after entry as we show in this chapter. A side-channel is often unnoticed and overlooked information leakage. For passwords, side-channels may arise from many sources, such as the sounds keys make when pressed [67], or the timing information between key-presses [114]. Often such side-channels require the adversary to be present at the time of password entry. However, post factum attacks are intriguing since they reduce the possibility of detection. To this end, this dissertation proposes *the first thermal residue side-channel attack against passwords entered on external keyboards*. We show that this attack is a real threat – not just a *Mission Impossible*-style gimmick, since thermal imaging devices that were previously considered niche are now cheaper and more advanced.

To-date, there has been no systematic investigation of thermal profiles of keyboards, and thus no efforts have been made to secure them. This serves as our main motivation for constructing a means for password harvesting from keyboard thermal emanations. Specifically, we introduce `Thermanator`, a new post factum insider attack based on heat transfer caused by a user typing a password on a typical external keyboard.

We conduct and describe a user study that collected thermal residues from 30 users entering 10 unique passwords (both weak and strong) on 4 popular commodity keyboards. Results show that entire sets of key-presses can be recovered by non-expert users as late as 30 **seconds** after initial password entry, while partial sets can be recovered as late as 1 **minute** after entry. Furthermore, we find that typists who press keys individually (also known as Hunt-and-Peck typists) are particularly vulnerable. We also discuss some Thermanator mitigation strategies. The take-away of our work is three-fold: (1) using keyboards to enter passwords is even less secure than previously recognized, (2) post factum (either planned or impromptu) thermal imaging attacks are realistic, and (3) we should either stop using keyboards for password entry, or abandon passwords altogether.

2.2 Motivation & Contributions

Any time two objects with unequal temperatures come in contact with each other, an exchange of heat occurs. This is unavoidable. Being warm-blooded, human beings naturally prefer environments that are colder than their internal temperature. Because of this heat disparity, it is inevitable that we leave thermal residue on numerous objects that we routinely touch, especially, with bare fingers. Furthermore, it takes time for these heated objects to cool off and lose heat energy imparted by human contact. It is both not surprising and worrisome that this includes our interactions with keyboards that are used for entering sensitive private information, such as passwords.

Based on this observation, we consider a mostly unexplored attack space where heat transfer and subsequent thermal residue can be exploited by a clever adversary to steal passwords from a keyboard some time after it was used for password entry. The main distinctive benefit of this attack type is that adversary's real time presence is not required. Instead, a successful attack can occur with after-the-fact adversarial presence: as our results show, many seconds later.

While there has been some prior work on using thermal emanations to crack PINs, mobile phone screen-locks and opening combinations of vaults/safes [175, 106, 107, 109], this work represents the first comprehensive investigation of human-based thermal residues and emanations of external computer keyboards.

In this chapter, we propose and evaluate a particular human-based side-channel attack class, called Thermanator. This attack class is based on exploiting thermal residues left behind by a user (victim) who enters a password using a typical external keyboard. Shortly after password entry, the victim either steps away inadvertently, or is drawn away (perhaps as a result of being prompted by the adversary) from the personal workplace. Then, the adversary captures thermal images of the victim keyboard. We examine the efficacy of Thermanator

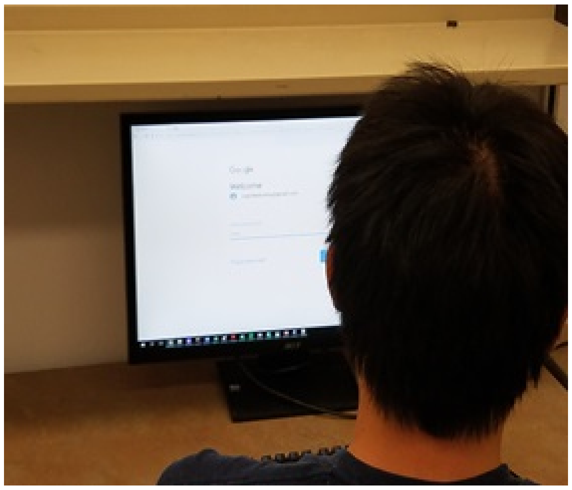
Attacks for a moderately sophisticated adversary equipped with a mid-range thermal imaging camera. The goal of the attack is to learn information about the victim password.

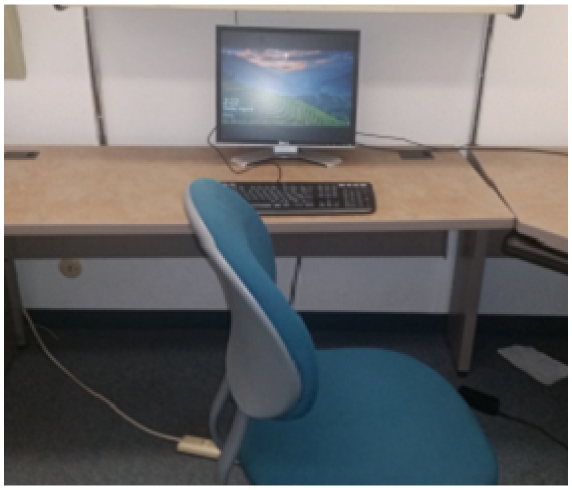
To confirm viability of Thermanator Attacks, we conducted a rigorous two-stage user study. The first stage collected password entry data from 31 subjects using 4 common keyboards. In the second stage, 8 non-expert subjects acted as adversaries and attempted to derive the set of pressed keys from the thermal imaging data collected in the first stage. Our results show that even novice adversaries can use thermal residues to reliably determine the entire set of key-presses **up to 30 seconds** after password entry. Furthermore, they can determine a partial set of key-presses as long as a full minute after password entry. We provide a thorough discussion of the implications of this study, and mitigation techniques against Thermanator Attacks.

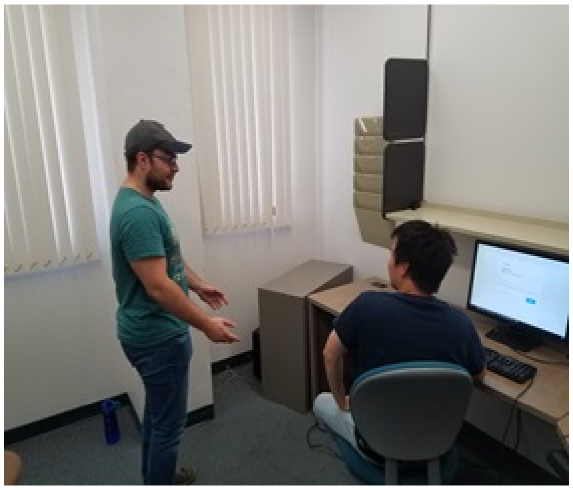
Furthermore, in the course of exploring Thermanator Attacks, we introduce a new post factum adversarial model. We comprehensively compare this model with those of other insider attacks that target user behavior and physical properties, such as Lunch-Time, Shoulder-Surfing, and Acoustic Emanations attacks. In doing so, we focus on attack characteristics, such as: goals, timeline and equipment required by the adversary. This comparison can be found in Section 2.8.

2.3 Adversarial Model & Attacks

This section describes the adversarial model for Thermanator Attacks.


(a) STEP 1: Victim Enters Password


(b) STEP 2: Victim Leaves (*Opportunistic*)


(b) STEP 2: Victim Drawn Away (*Orchestrated*)

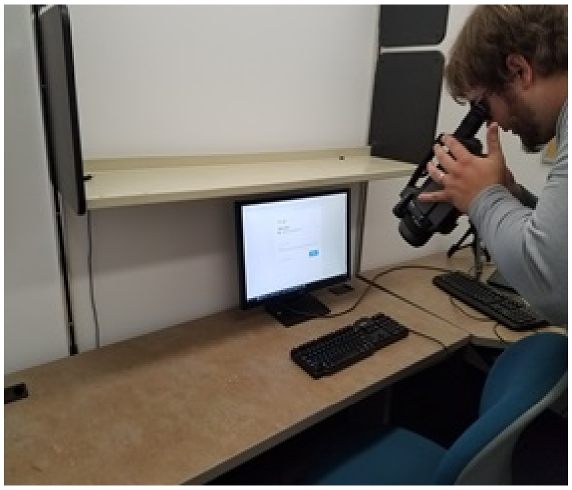

(c) STEP 3: Thermal Residues Captured

Figure 2.1: An Example Thermanator Attack

2.3.1 Physical Premise

As mentioned in Section 2.4, Fourier's Law states that contact between any two objects with unequal temperatures results in transfer of heat energy from the hotter to the cooler object. It is reasonable to assume that the typical office environment has the ambient temperature within the OSHA-recommended range of $293.15 - 298.15K$ ($=20 - 25°C$) [150]. In that setting, the average human hand is expected to conductively transfer an observable amount of heat to the ambient-temperature keyboard. Consequently, a bare-fingered human typist can not avoid leaving thermal residue on a keyboard. This physical interaction can be abused by the adversary in order to harvest the thermal residue of a victim who recently used a keyboard to enter potentially sensitive information, e.g., a password. This forms the premise for Thermanator Attacks.

2.3.2 Thermanator

Thermanator is a distinct type of insider attack, where a typical attack scenario proceeds as follows (see also Figure 2.1):

STEP 1: The victim uses a keyboard to enter a genuine password, as part of the log-in (or session unlock) procedure.

STEP 2: Shortly thereafter, the victim either: (1) willingly steps away, or (2) gets drawn away, from the workplace.

STEP 3: Using thermal imaging (e.g., photos taken by a commodity FLIR camera) the adversary harvests thermal residues from the keyboard.

STEP 4: At a later time, the adversary uses the "heat map" of the images to determine recently pressed keys. This can be done manually (i.e., via visual inspection) or automatically (i.e., via specialized software).

REPEAT: The adversary can choose to repeat STEPS [1-4] over multiple sessions.

The two options in **STEP 2** correspond to two attack sub-types: *opportunistic* and *orchestrated*. In the former, the adversary patiently waits for the situation described in **STEP 2** case (1) to occur. Once the victim leaves (on their own volition) shortly after password entry, the adversary swoops in and collects thermal residues. This strategy is similar to Lunch-Time Attacks. In an *orchestrated* attack, instead of waiting for the victim to leave, the adversary uses an accomplice to draw the victim away shortly after password entry, as in **STEP 2** case (2).

2.4 Background

In this section we provide some background material on physical interactions that describe thermal phenomena. We start with a glossary of terms, then describe the form factor and material composition of modern 104-key "Windows" keyboards and finish with certain Physics concepts used in the rest of the chapter.

2.4.1 Basic Thermal Terminology

- Joule (J) – Unit of energy Corresponding to 1 Newton-Meter ($N \cdot m$)
- Kelvin (K) – Base unit of temperature in Physics. The temperature T in Kelvin (K) minus 273.15 yields the corresponding temperature in degrees Celsius ($°C$).
- Watt (W) – Unit of power corresponding to 1 Joule per second: ($\frac{J}{s}$)
- Conduction – Transfer of Thermal Energy caused by two objects in physical contact that are at different Temperatures.

- Convection – Transfer of Thermal Energy caused by submerging an object in a fluid.
- Heat Transfer Coefficient - Property of a fluid that determines rate of convective heat flow. Expressed in Watts per square meter Kelvin: $\frac{W}{m^2K}$
- Specific Heat – Amount of Thermal Energy in Joules that it takes to increase temperature of 1kg of material by $1K$. Expressed in Joules over kilograms degrees Kelvin: $\frac{J}{kgK}$.
- Thermal Conductivity – Rate at which Thermal Energy passes through a material. Expressed in Watts per meters Kelvin: $\frac{W}{mK}$
- Thermal Energy – Latent energy stored in an object due to heat flowing into it.
- Thermal Source – Object or material that can internally generate Thermal Energy such that it can stay at constant temperature during a thermal interaction, e.g., a heat pump.

2.4.2 Heating via Thermal Conduction

Thermal Conduction is transfer of heat between any two touching objects of different temperatures. It is expressed as the movement of heat energy from the warmer to the cooler object.

> *Heat transfer between two objects can be modeled by the equation:* $q = \frac{\mathcal{K}A(T_1-T_2)t}{d}$, *where* $\mathcal{K}$ *is thermal conductivity*[1] *of the object being heated,* A *is area of contact,* T_1 *is initial temperature of the hotter object,* T_2 *is initial temperature of the cooler object,* t *is time, and* d *is the thickness of the object being heated.*

The relationship between an object's heat energy and its temperature is governed by the object's mass and specific heat, as dictated by the formula: $q = cm\Delta T$, where q is total heat energy, c is object's specific heat, m is object's mass and ΔT is change in temperature.

[1]$\mathcal{K}$ should not be confused with K – degrees Kelvin.

We consider the human body to be a thermal source, and we assume that any change in the fingertip temperature during the (very short) fingertip-keycap contact period is negligible, due to internal heat regulation [108]. Furthermore, we assume that:

- Average human skin temperature is $307.15K$ ($= 34°C$) [85].
- Keyboard temperature is the same of that as that of the air, which, for a typical office, is OSHA[2]-recommended $294.15K$ ($= 21°C$) [150].
- Keycap area is 0.00024025 m^2, keycap thickness is 0.0015 meter and keycap mass is $.4716g$ (See: Section 2.4.4).
- Average duration of a key-press is 0.28s [162].

Therefore, for variables mentioned above, we have:

$$\boxed{\mathcal{K}\mathbf{=0.25,\quad A=0.00024025,\quad T1=34,\quad T2=21,\quad t=0.28,\quad and\quad d=0.0015}}$$

Plugging these values into Fourier's Law, we get:

$$q = \frac{(0.25)(0.00024025)(34 - 21)(.28)}{0.0015}$$

which yields total energy transfer: $q = 0.1458$J. We then use total energy q in the specific heat equation to determine total temperature change: $0.1458 = (1000)(0.0004716)\Delta T$. This gives us a total temperature change of $\Delta T = 0.3092$. Therefore, we conclude that the average human fingertip touching a keycap at the average room temperature results in the keycap heating up by $0.3092K$.

[2]OSHA = Occupational Safety and Hazards Administration, a United States federal agency.

2.4.3 Cooling via Thermal Convection

After a keycap heats up as a result of conduction caused by a press by a warm(er) human finger, it begins to cool off due to convective heat transfer with the air in the room. Convection is defined as the transfer of heat resulting from the internal current of a fluid, which moves hot (and less dense) particles upward, and cold (and denser) particles – downward. This interaction is governed by Newton's Law of Cooling. Its particulars are impacted by the shape and position of the heated object. In our case, there is a plane surface[3] facing towards the cooling fluid (i.e., a keycap directly exposed to ambient air) which is described by the formula:

$$T(t) = T_s + (T_0 - T_s)e^{-\kappa t}$$

where $T(t)$ is temperature at time t, T_s is temperature of ambient air, T_0 is initial object temperature, and κ is the cooling constant of still (non-turbulent) air over a $0.00024025m^2$ plane.

This comes with the additional intuitive notion that a surface convectively cools quicker when the temperature difference between the heated object and the fluid is higher. Similarly, it cools slower when the temperature difference is smaller. Finally, Newton's Law of Cooling is asymptotic, and cannot be used to find the time at which the object reaches the exact temperature of the ambient fluid. Thus, instead of finding the time when the temperatures are equal, we determine the time when the temperature difference falls below an acceptable threshold, which we set at $0.04K$. Plugging this into Newton's Law of Cooling results in:

$$t = -\frac{ln(\frac{0.3092}{0.04})}{0.037}$$

which yields $t = 55.7$ for total time for a pressed key to cool down to the point where it is indistinguishable from the room temperature.

[3]The actual keycap surface can be slightly concave.

2.4.4 Modern Keyboards

Most commodity external keyboard models are of the 104-key "Windows" variety, shown in Figure 2.2a. On such keyboards, the distance between centers of adjacent keys is about 19.05mm, and a typical keycap shape is an $\approx [15.5mm\ x\ 15.5mm\ x\ 1.5mm]$ rectangular prism, with an average travel distance of $3.55mm$ [149]; see Figure 2.2b. All such keyboards are constructed out of Polybutylene Terephthalate (PBT) with density of $1.31g/cm^3$, resulting in an average keycap mass of $.4716g$ [111]. PBT generally has the following characteristics: specific heat = $1,000\frac{J}{kgK}$ and thermal conductivity = $0.274\frac{W}{mK}$ [111].

(a) Typical "Windows"-style Keyboard.

(b) Typical Keycap Profile.

2.4.5 Thermal Cameras

In the past few years, many niche computational and sensing devices have moved from Hollywood-style fantasy into reality. This includes thermal imagers or cameras. In order to clarify their availability to individuals (or agencies) at different levels of sophistication, we provide the following brief comparison of several types of readily-available FLIR: **F**orward-**L**ooking **I**nfra-**R**ed devices. (See: Figure 2.3 for product images and `https://www.flir.com/products` for full product specifications.) In the rest of the chapter, we use the following terms interchangeably: FLIR device, thermal imager and thermal camera.

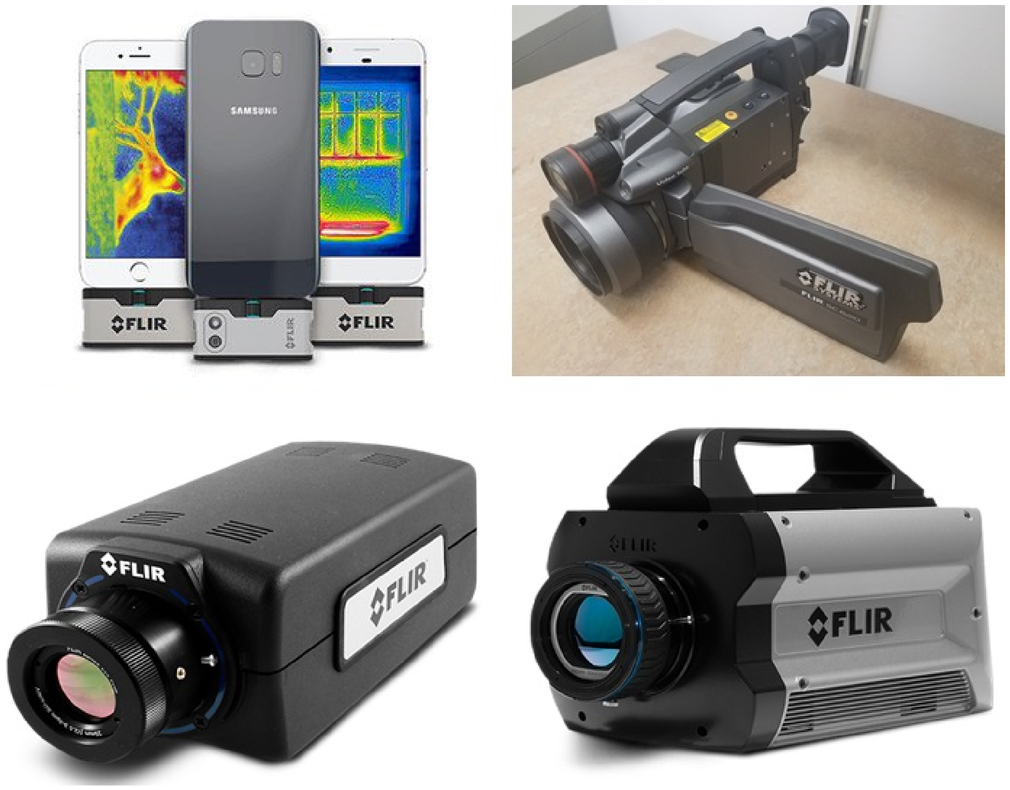


Figure 2.3: FLIR Devices / Thermal Imagers: FLIR ONE(top left), SC620 (top right), A6700sc (bottom left) and X8500sc (bottom right).

FLIR One – Price: About US$300. Thermal Sensitivity: 0.15K. Thermal Accuracy: ±1.5K or 1.5% of reading. Resolution:50x80. Image Capture: Manual, 1 image at a time. Video Capture: None

SC620 – Price: About US$1500 (used). Thermal Sensitivity: 0.04K Thermal Accuracy: ±2K or 2% of reading. Resolution: 640x480. Image Capture: Automatic, programmable to capture images by timer, or when specific criteria are met, at maximum rate of 1 image per second. Video Capture: None.

A6700sc – Price: About US$25,000. Thermal Sensitivity: 0.018K Thermal Accuracy: ±2K or 2% of reading. Resolution: $640x512$. Image Capture: Automatic, programmable to capture images by timer or when specific criteria are met, at up to 100fps. Video Capture: High speed, up to 100fps.

X8500sc – Price: About US$100,000. Thermal Sensitivity: 0.02K: Thermal Accuracy: $\pm$2K or 2% of reading. Resolution: $1280x1024$ Image Capture: Automatic, programmable to capture images by timer or when specific criteria are met, at up to 180fps. Video Capture: High speed, up to 180fps.

Obviously, a sufficiently motivated organization or a nation-state could easily obtain thermal imagers of the highest quality and price. However, we assume that the anticipated adversary is of a mid-range sophistication level, i.e., capable of acquiring a device exemplified by SC620. However, we note the adversary armed with a FLIR One (which is on the low-end of the spectrum for thermal imagers, and can be connected to any commodity smartphone without substantially altering the overall form factor) can collect thermal residues up to 20 seconds after entry. Whereas, the adversary with a A6700sc or X8500sc can do the same 139 seconds, and 136 seconds after entry, respectively. Also, since thermal residues decay at a logarithmic rate, future advances in thermal camera sensitivity will result in a exponential increase of collection time.

2.5 Methodology

In this section we describe of the experimental apparatus, procedures, and subject recruitment methods.

2.5.1 Apparatus

The experimental setup was designed to simulate a typical office setting. It was located in a dedicated office in a research building of a large university. Since experiments were conducted during the academic year, there was always some (though not excessive) amount of typical busy office-like ambient noise. Figure 2.5a shows the setup from the subject's perspective. Equipment used in the experiments consisted of the following readily available (off-the-shelf) components:

I. FLIR Systems SC620 Thermal Imaging Camera[4] This camera was perched on a tripod 24″ above the keyboard.
II. Four popular and inexpensive commodity computer keyboards: (a) Dell SK-8115, (b) HP SK-2023 (c) Logitech Y-UM76A, and (d) AZiO Prism KB507. The first (Dell) is shown in 2.2a above, and the other three – in Figure 2.4.

The particular thermal camera that was used in our experiments was chosen to be realistic for a moderately sophisticated and determined adversary. We assume this type of adversary to be an individual, i.e., not an intelligence agency, a nation-state, or a powerful criminal organization. FLIR SC620 Thermal Imager costs approximately US$1,500 used. (This model is about 6-7 years old.) It automatically records images at the resolution of $640x480$ pixels, with $1Hz$ frequency. Its thermal sensitivity is $0.04K$.

The four keyboards were chosen to cover the typical range of manufacturers represented in an average workplace. Dell, HP and Logitech keyboards are popular default keyboards included in new computer orders from major PC, desktop, and workstation manufacturers. Each costs ≈ US$20. Meanwhile, Azio Prism is a popular low-cost and independently manufactured keyboard that can be easily obtained on-line e.g., from Amazon; it costs ≈ US$25.

[4]see: `http://www.FLIR.com` for a full specification.

(a) HP SK-2023

(b) Logitech Y-UM76A.

(c) AZiO Prism KB507 (backlit).

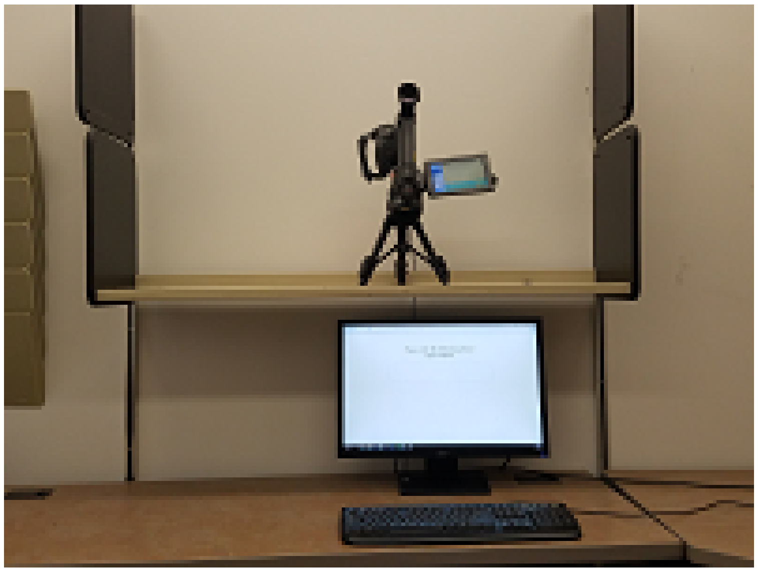
Figure 2.4: Keyboards

2.5.2 Procedures

`Thermanator` was evaluated using a two-stage user study. The first stage was conducted to collect thermal emanation data, and the second – to evaluate efficacy of Thermanator Attacks. A given subject only participated in a single stage.

Stage One: Password Entry

Recall that Thermanator's goal is to capture thermal residues of subjects **after** keyboard password entry. This is accomplished by having FLIR SC620 take a sequence of images (60


(a) SC620 Apparatus Setup

(b) Example of Thermal Emanations being Recorded.

total), one per second, for a total of one minute after initial password entry. The first stage is shown in Figure 2.6. This collection of 60 images does **not** represent the requirements for a single attack. In reality, the adversary would arrive as quickly as possible (after the victim leaves the workspace) and take a single thermal image. For strictly experimental purposes, a full minute of thermal data was captured to more accurately model adversaries arriving after some time has elapsed.

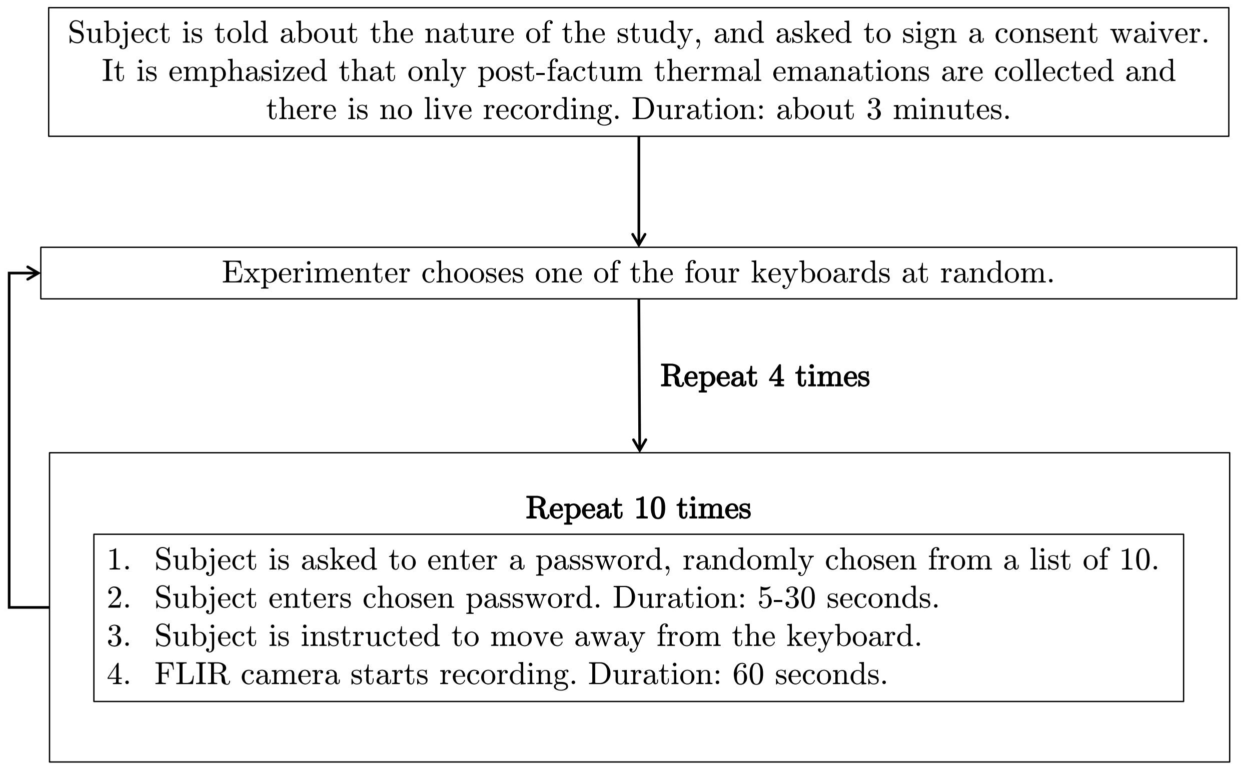

Subject is told about the nature of the study, and asked to sign a consent waiver. It is emphasized that only post-factum thermal emanations are collected and there is no live recording. Duration: about 3 minutes.
Experimenter chooses one of the four keyboards at random.
Repeat 4 times
Repeat 10 times
1. Subject is asked to enter a password, randomly chosen from a list of 10.
2. Subject enters chosen password. Duration: 5-30 seconds.
3. Subject is instructed to move away from the keyboard.
4. FLIR camera starts recording. Duration: 60 seconds.

Figure 2.6: Experiment Stage One: Flowchart

Each subject entered 10 passwords on 4 keyboards and each entry was followed by one minute of keyboard recording (60 successive images) by the FLIR. Each subject entered a total of 40 passwords and every entry took, on average, between 10 and 20 seconds. The total duration of the experiment for a Stage 1 subject ranged between 50 and 60 minutes, based on the individual's typing speed and style. Both keyboards and passwords were presented to each subject in random order, in an attempt to negate any side-effects due to subject training or familiarity with the task.

We selected 10 passwords that included both "insecure" and "secure" categories. The former passwords were culled from the top 100 passwords by popularity that adhere to common password requirements, such as Gmail [5]. Whereas, "secure" passwords were created by randomly generating 8-, 10-, and 12-character strings of lower/uppercase letters as well as

[5]see: `https://support.google.com` for details

numbers and symbols that adhere to Gmail restrictions. Our selection criteria resulted in the following 10 candidate passwords:

- **[Insecure]:** "password", "12345678", "football", "iloveyou", "12341234", "passw0rd", and "jordan23",
- **[Secure]:** "jxM#1CT[", "3xZFkMMv|Y", and "6pl;0>6t(OvF".

Stage Two: Data Inspection

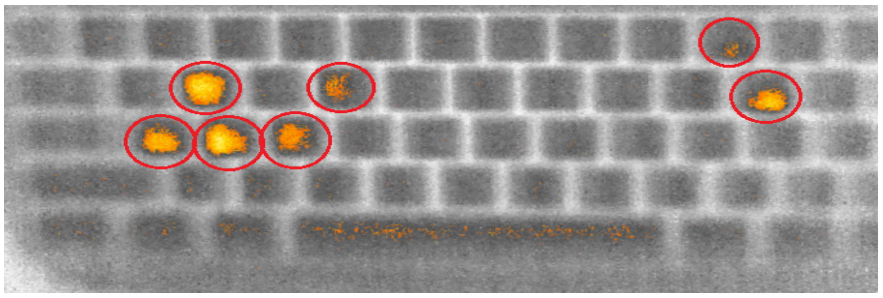


Figure 2.7: Thermal image of "passw0rd" 20 seconds after entry.

The second stage of the experiment has subjects act as adversaries conducting Thermanator Attacks. Subjects were shown images obtained from the first stage of the experiment, e.g., Figure 2.7, and were instructed to identify the "lit" regions. Each subject was shown 150 recordings of password entries in random order. On average, a subject could process a single recording in 45 – 60 seconds. Total time for each Stage 2 subject varied in the range of 100 – 130 minutes.

2.5.3 Subject Recruitment Procedure

Subjects were recruited from the (student body of a large public University using a unified Human Subjects Pool designated for undergraduate volunteers seeking to participate in studies such as ours. Subjects were compensated with course credit. Because of this, overwhelming majority of subjects were of college age: $18--25$. The subject gender breakdown was: 16 male and 15 female.

All experiments were authorized by the Institutional Review Board (IRB) of the authors' employer, well ahead of the commencement of the study. The level of review was: Exempt, Category II. No sensitive data was collected during the experiments and minimal identifying information was retained. In particular, no subject names, phone numbers or other personally identifying information (PII) was collected. All data is stored pseudonymously.

2.6 Results

We now describe the results of Stage 2 analysis of thermal images obtained in Stage 1. We divide it into two categories:

- Hunt-and-Peck Typists — 'those who **do not** rest their fingertips on, or hover their fingers just over, the home-row of keys (i.e. "ASDF" on the left hand, and "JKL;" on the right hand.).
- Touch Typists – those whose fingertips routinely hover over, or lightly touch, the home-row.

The distribution of our Stage 1 subjects to these categories were: 18 hunt-and-peck typists and 12 touch typists.

As it turns out, our study results indicate that the category of the typist is the most influential factor for the quality thermal imaging data. For each category, we separately analyze "secure" and "insecure" passwords types. Since we did not observe a significant statistical difference between results of different keyboards, results include all keyboards.

For full context, aggregate results (identification rates) from the entire subject population are shown in Figures 2.8, 2.9 and 2.10; they correspond to stage 2 subjects' analysis of "insecure" and "secure" passwords, respectively. For clarity's sake, "insecure" passwords are split into two subcategories: alphabetical and alphanumeric. The former contains "insecure" passwords that consist only of English-language letters, while the latter contains "insecure" passwords that also include numbers. In each graph, "D = 0" refers to average latest time when stage 2 subjects could correctly identify every keystroke of the entered password, while "D = 1" denotes average latest time when subjects could identify all-but-one keystroke; "D = 2" denotes the average latest time when subjects could identify all-but-two keystrokes, and so on. The distance "D" is calculated as

$$D = |(K \cup P) \setminus (K \cap P)| \tag{2.1}$$

where P is the set of pressed keys identified by Stage 2 subjects and K is the set of keys in the actual password. Note that keys missed and misidentified as pressed are both considered in this distance calculation.

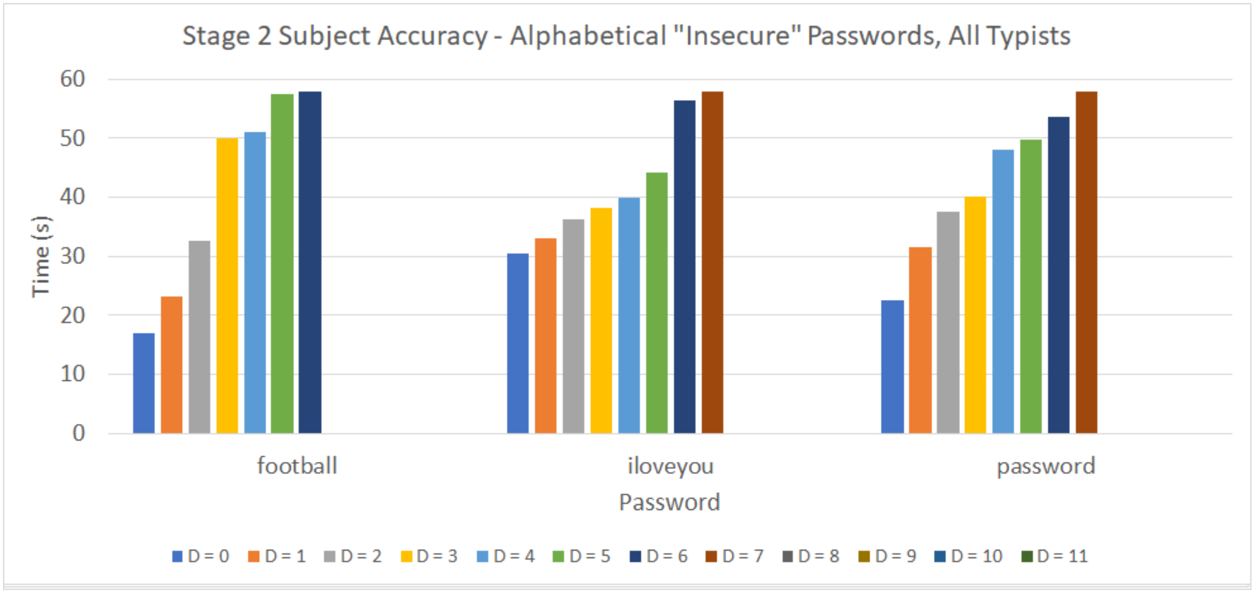


Figure 2.8: Stage 2 Subject Performance: Alphabetical "Insecure" Passwords, all Typists

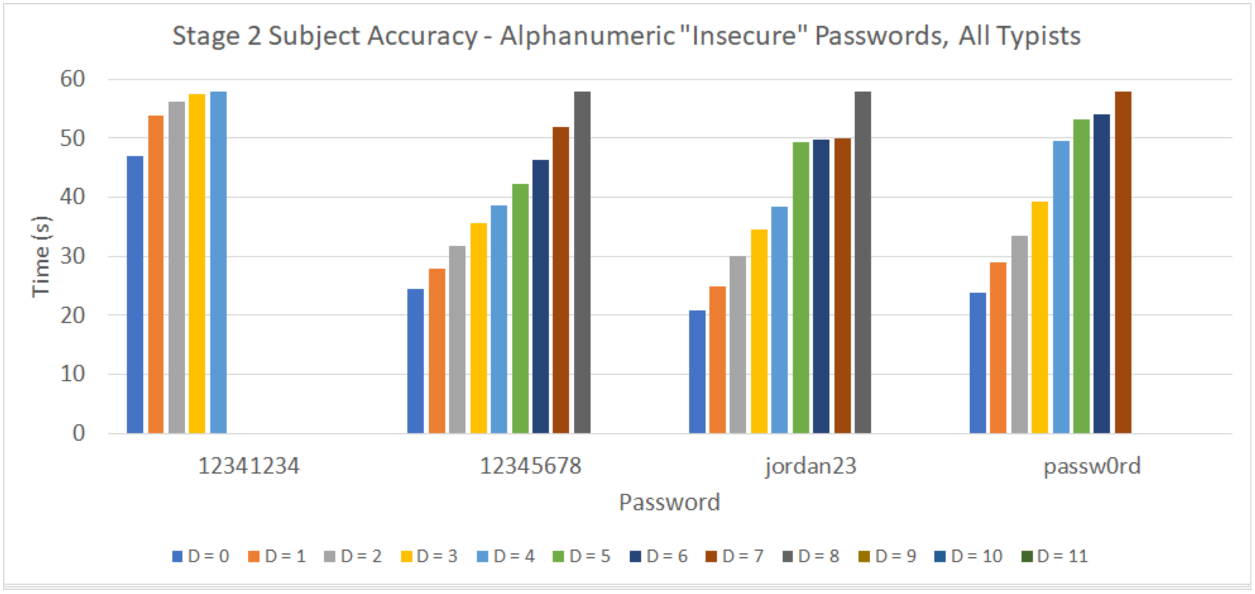


Figure 2.9: Stage 2 Subject Performance: Alphanumeric “Insecure” Passwords, all Typists

2.6.1 Hunt-and-Peck Typists

Our analysis of Hunt-and-Peck typists was straightforward. Because these typists do not rest their fingertips on (or hover right above) the keyboard home-row, it is readily apparent that each bright spot on the thermal image corresponds to a key-press. However, as discussed below, we encountered some challenges with “secure” passwords.

Insecure Passwords

As Figure 2.11 and 2.12 show, analysis of Hunt-and-Peck typists entering “insecure” passwords is straightforward. In fact, in the best-case of “12341234” subjects could correctly recall every keystroke, on average, 45.25 seconds after entry. Even the weakest result, “football” was fully recoverable 25.5 seconds later, on average. This is in line with conventional thought. Hunt-and-Peck typists typically only use their forefingers to type. Because of this, they make contact with a larger finger over a large surface area. Also, since Hunt-and-Peck typists are generally less skilled, they take longer for each keystroke, resulting in longer con-

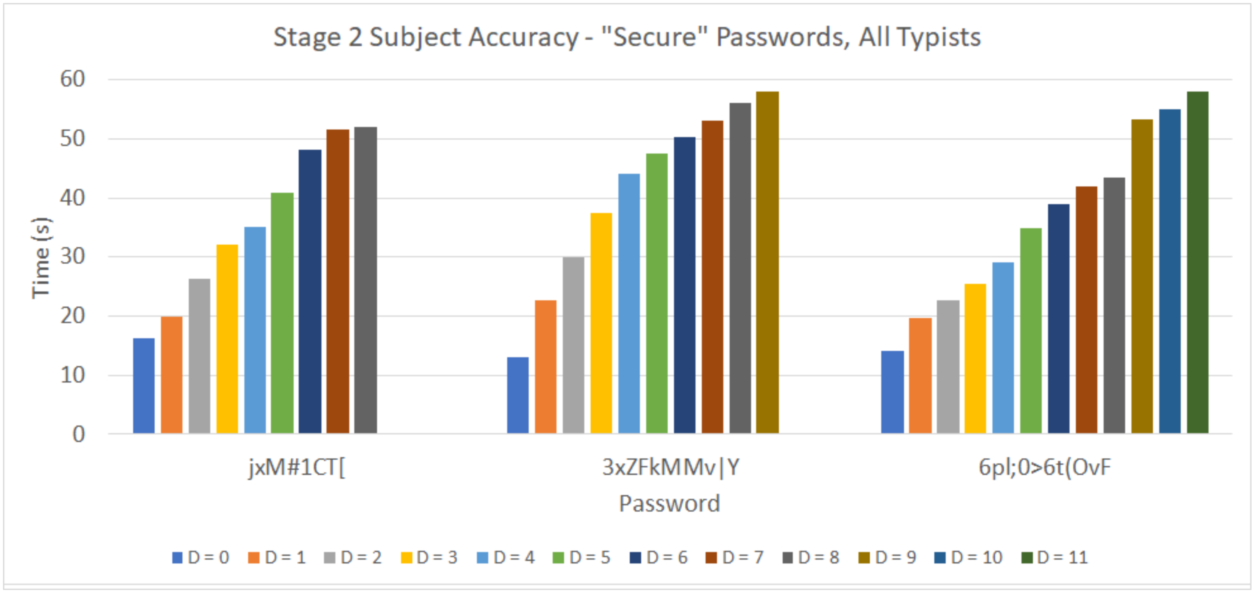


Figure 2.10: Stage 2 Subject Performance: “Secure” Passwords, all Typists

tact time. These two factors combined yield high-quality thermal residue for Thermanator Attacks.

Secure Passwords

“Secure” passwords are more challenging to analyze. As shown in Figure 2.13 full recall was possible, on average, up to 31 seconds after recording started, in the best case, and 19.5 seconds, in the worst case. Performance of stage 2 subjects was uniform in terms of password length: the shortest password was the easiest to analyze correctly. Anecdotally, this is not surprising. It was quite common for Hunt-and-Peck typists to look back and forth between the characters of a relatively complex “secure” passwords, and their keyboards. This resulted in longer completion times, which left longer time for keycaps to cool off before recording began.

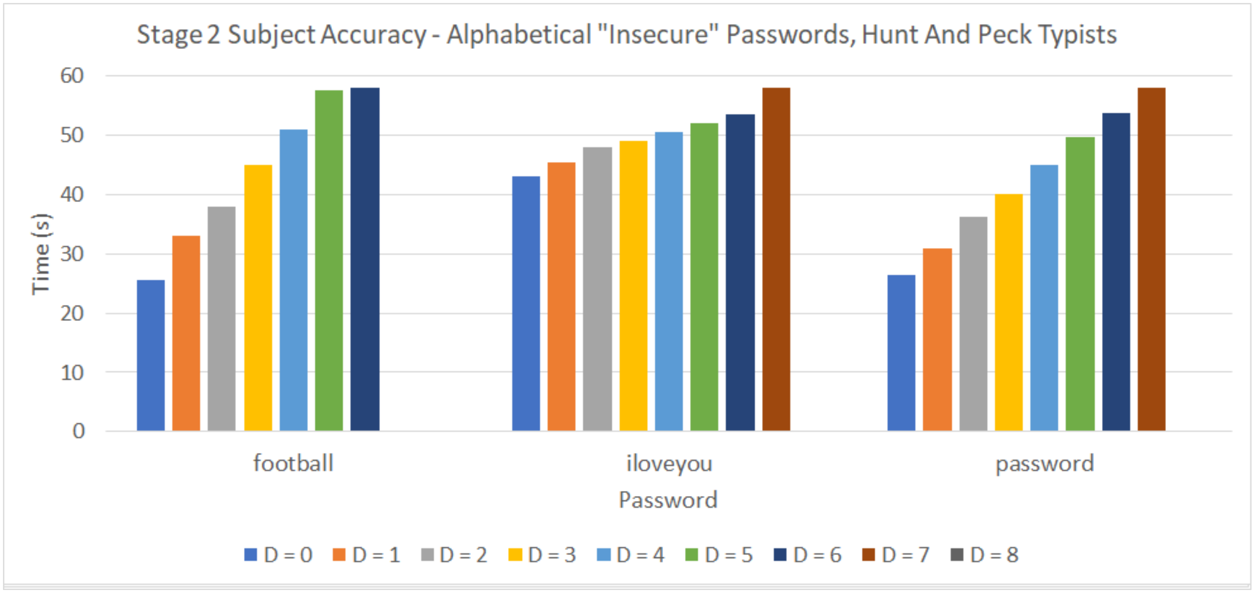


Figure 2.11: Stage 2 Subject Performance: Alphabetical "Insecure" Passwords, Hunt-and-Peck Typists

2.6.2 Touch Typists

Analyzing data from Touch typists was a challenge for stage 2 subjects. Since a typical Touch typist's fingers are constantly in contact with (or in very close proximity of) the home-row of the keyboard, there are two incidental sources of thermal noise. First, there is thermal residue on the 2 groups of 4 home-row keys: "asdf" and "jkl;" which results from the typist's fingertips. However, whenever typist's fingers rest on the keyboard for a long time, additional observed effects occur outside (though near) the home-row, on the following keys:

```
"qwertgvcxz", "][poiuhnm,./"
```

Even though this secondary thermal residue was not as drastic as that on the home-row, it had a more pronounced effect on stage 2 subjects. In many cases, a subject was uncertain whether a key was lit on the thermal image because it was actually pressed, or because it was simply close to the home-row. This uncertainty in turn led to mis-classification of some keys as unpressed. Also, mis-classification of home-row keys as pressed keys was not counted

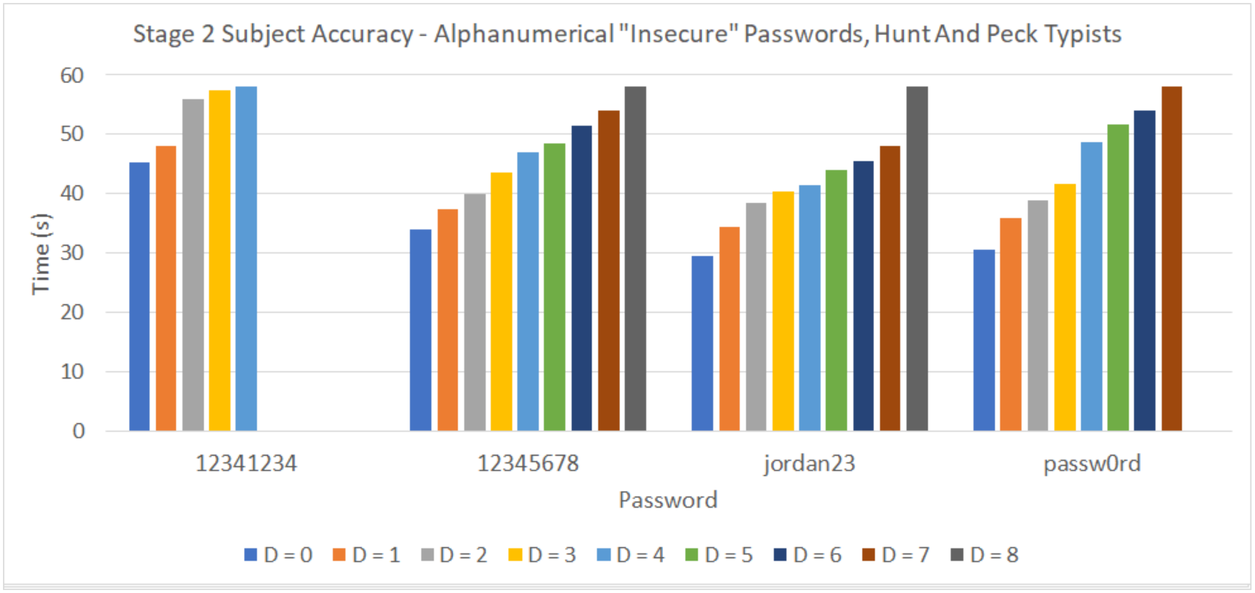


Figure 2.12: Stage 2 Subject Performance: Alphanumerical “Insecure” Passwords, Hunt-and-Peck Typists

in the distance. We justify this choice in Section 2.7.

Insecure Passwords

While more difficult than analysis of “insecure” password for Hunt-and-Peck typists, stage 2 subjects have moderate success analyzing Touch typists entering “insecure” passwords. As Figures 2.14 and 2.15 show, the best average time for full recall was for password: “12341234” at 47.6 seconds, and the worst was for “jordan23”, at 17.8 seconds. This follows the notion that stage 2 subjects were hesitant to classify home-row-adjacent key-presses, e.g., “o”, “r” and “n” in “jordan23”. Furthermore, this supports the notion that a simple, repeated password such as “12341234” leaves ideal thermal residue. Since each key is repeated, it is analogous to each key being pressed once for twice as long. This results in twice as much thermal energy being transferred from the fingertip to the keycap.

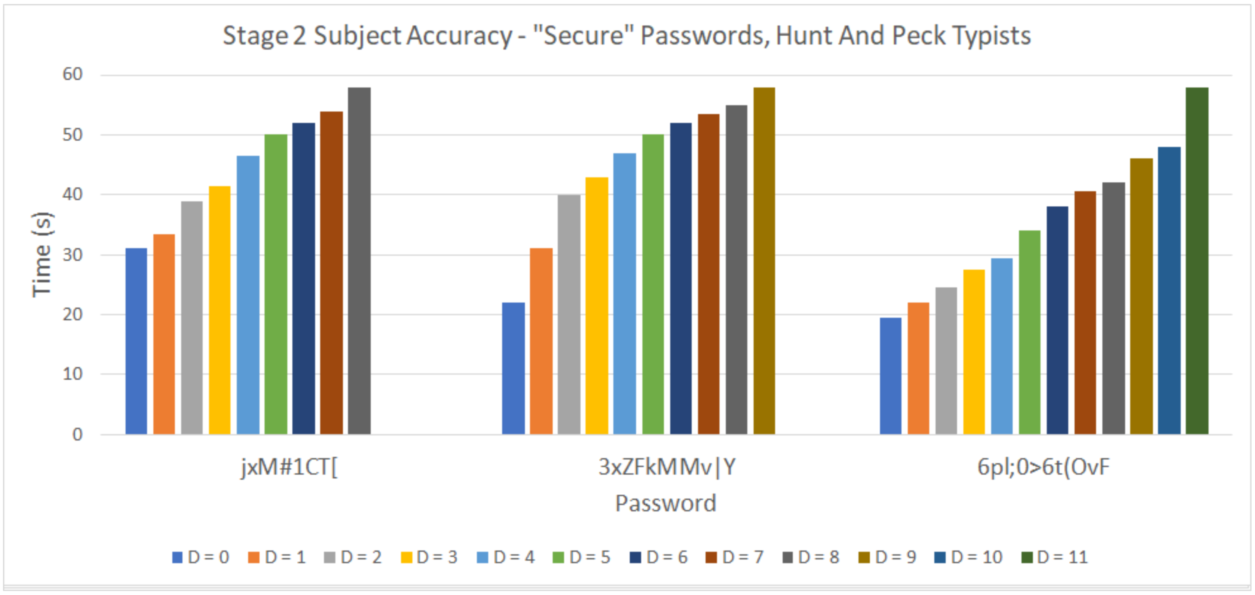


Figure 2.13: Stage 2 Subject Performance: “Secure” Passwords, Hunt-and-Peck Typists

Secure Passwords

Touch typists entering “secure” passwords were the most difficult for the stage 2 subjects to analyze. As shown in Figure 2.16, full recall was only possible, on average, within the first 14.33 – –18.5 seconds. Surprisingly, the password with the smallest window for full recall was “jxM#1CT[”. We believe that many stage 2 subjects were hesitant to classify home-row-adjacent keys in this password as keystrokes (as opposed to thermal noise). This might explain why the window for full recall is so small. As with all other cases, the time window between full recall at $d = 0$ and a single mis-identification $d = 1$ was much greater than any other window between $d = n$ and $d = n + 1$, which is consistent with Newton’s Law of Cooling.

2.6.3 Outlier: Acrylic Nails

There was a single Stage 1 subject that had long acrylic fingernails. Instead of typing with fingertips, this person tapped the keys with nail-tips. Since these do not have nearly as

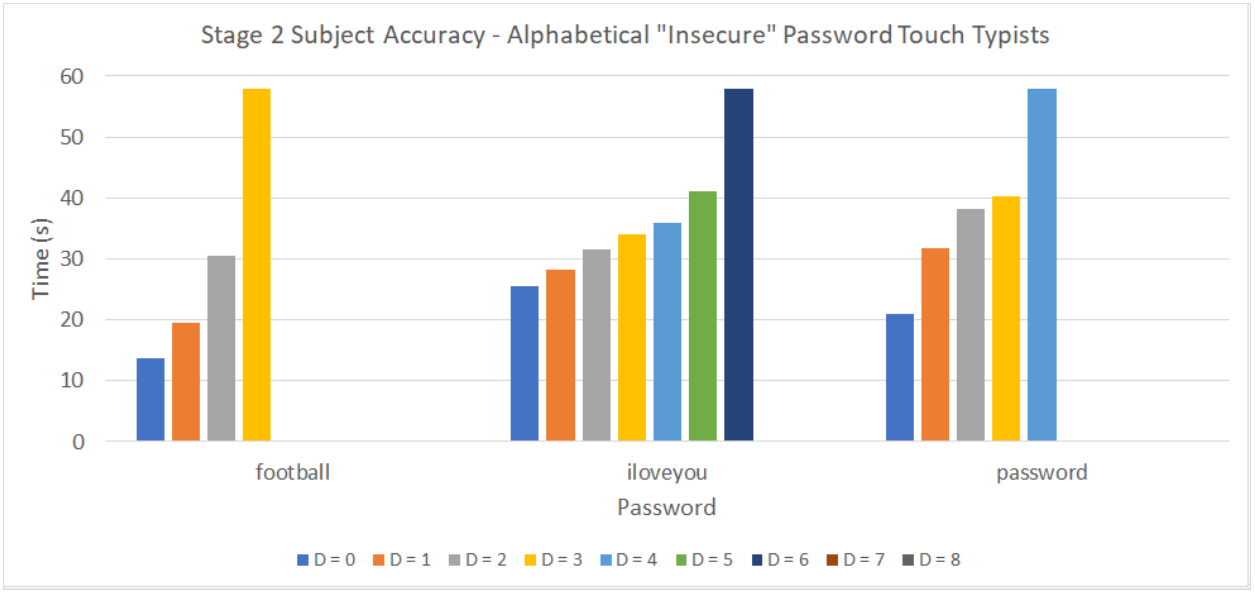


Figure 2.14: Stage 2 Subject Performance: Alphabetical "Insecure" Passwords, Touch Typists

much surface area as fingertips, and false nails do not have any blood vessels to regulate their temperature, this subject left almost no thermal residue. In fact, not a single key-press could be correctly identified in any of the 40 password entry trials. Consequently, this subject is not included in either Touch or Hunt-and-Peck typist populations. However, as a side curiosity, we note that, although it may be a rare occurrence, any user with long acrylic fingernails is virtually immune to Thermanator Attacks.

2.7 Discussion

In this section, we break down our observations from Section 2.6 between two password classes, and among two categories of typists.

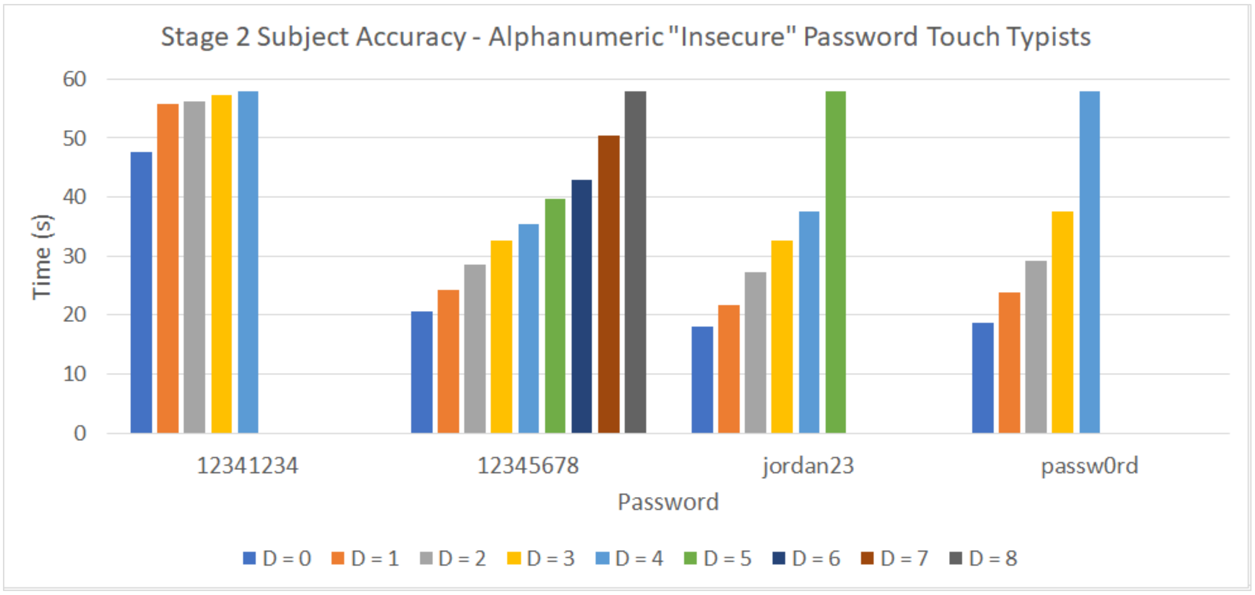


Figure 2.15: Stage 2 Subject Performance: Alphanumeric "Insecure" Passwords, Touch Typists

2.7.1 Results with Common Passwords

Stage 2 subjects were particularly adept at identifying passwords that are English words or phrases. Even though we could not reliably detect the exact sequence of pressed keys, ordering can be found indirectly by mapping the set of pressed keys to words (essentially, solving an anagram puzzle). Furthermore, a list of distances between detected keys (characters) and possible words, can be used to reconstruct full passwords from incomplete thermal residues.. Finally, the same list of distances can help determine when a key is pressed multiple times. These combinations highlight the threat posed by Thermanator Attacks to already insecure passwords.

2.7.2 Results with Random Passwords

However, strong results from Stage 2 subjects' identification of English-language words does not extend to secure, randomly-selected passwords. First, inability to reliably determine the order of pressed keys can not be mitigated by leveraging the underlying linguistic structure.

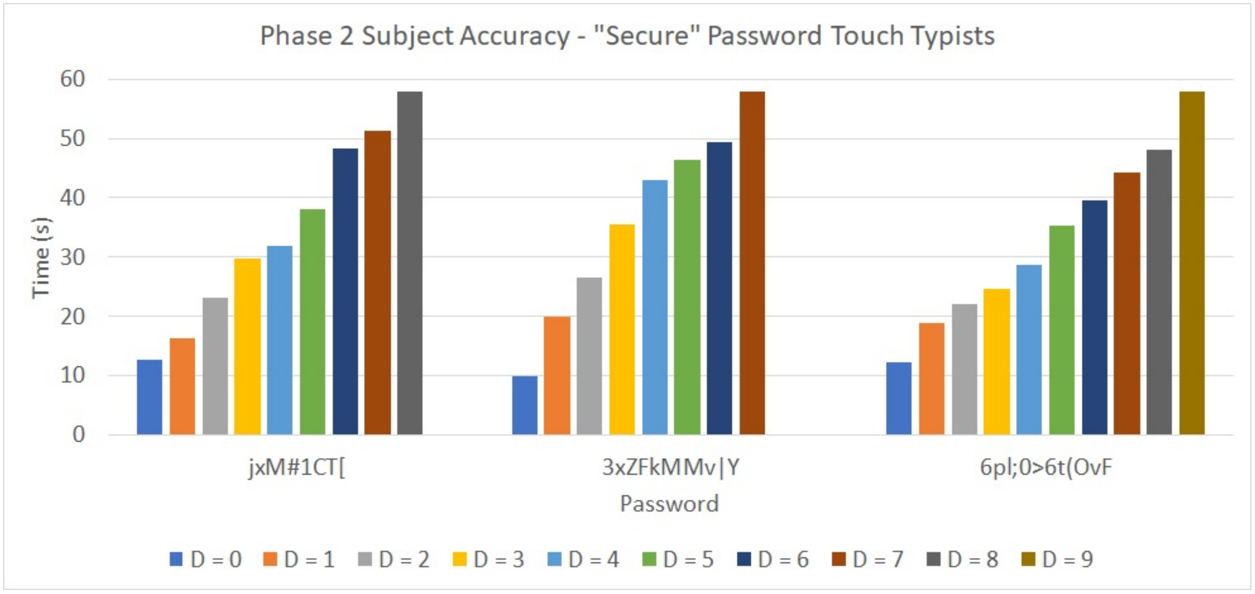


Figure 2.16: Stage 2 Subject Performance: “Secure” Passwords, Touch Typists

Moreover, it is unclear whether a given set of emanations represents the whole password, or if some information was lost. Finally, it is impossible to tell if a key was pressed multiple times. However, even with these shortcomings, our subjects managed to greatly reduce the password search space from 72^n to $72^{n-m} * m!$ where n is the total number of characters in the password, and m is the number of identified key-presses. This represents a reduction in search space by a factor of 10^{10} for an 8-character password where the individual keys have been identified. Techniques to further reduce the space of candidate passwords are discussed in the following section.

2.7.3 Results with Hunt-and-Peck Typists

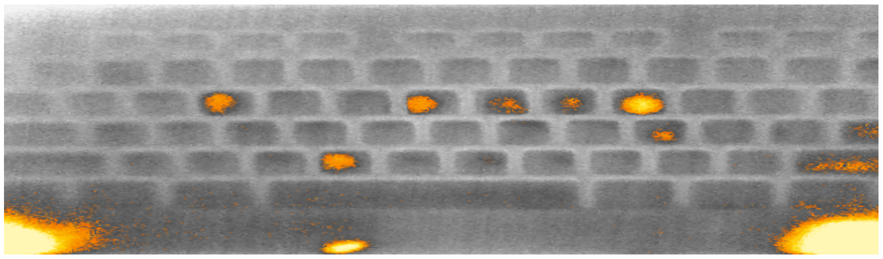


Figure 2.17: Password “iloveyou” entered by a Hunt-and-Peck typist.

As described in Section 2.6.1, Hunt-and-Peck typists are particularly vulnerable to Thermanator Attacks. This is not surprising, given that these less-skilled typists tend to type more slowly, and primarily use their index fingers, which have greater fingertip surface area than ring or pinky fingers [112]. This results in greater heat transfer, due to longer contact duration with a larger contact area. Also, as seen from Figure 2.17, Hunt-and-Peck typists do not touch any keys that are not part of the password. Therefore, every observed key-press is part of the password.

2.7.4 Results with Touch Typists

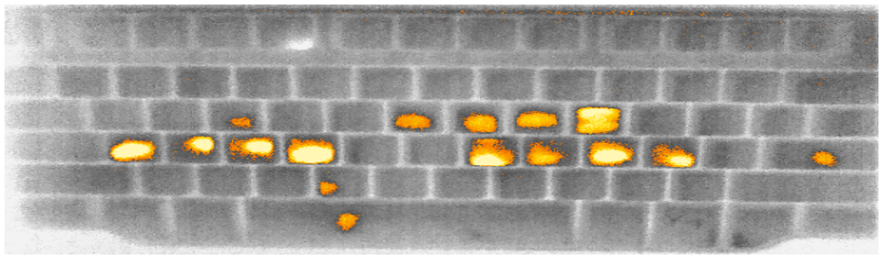


Figure 2.18: Password "iloveyou" Entered by a Touch Typist.

For Touch typists, two factors confuse their thermal residues and make passwords harder to harvest. One is their habit to rest their hands on the home-row, which introduces potential false positives. as Figure 2.18 shows. This is exacerbated by the possibility that any home-row key might actually be part of the password. Because of this, stage 2 subjects were not penalized for classifying the home-row keys as pressed; they were instructed to identify all keys that looked to them as having been pressed.

Another issue is that Touch typists tend to use all fingers of both hands while typing. This causes two advantages over their Hunt-and-Peck counterparts. First, they touch individual keys for a shorter time, thus transferring less heat to the key-cap. Second, they type much more quickly and also use their ring and pinky fingers. Fingertips of these smaller fingers tend to have 1/2 of the surface area of larger index or middle fingers. Thus, they transfer half of the total heat energy due to conduction during a key-press [112]. Such factors make Touch typists much more resistant to Thermanator Attacks, particularly, at the level of our moderately sophisticated adversarial model.

2.7.5 Ordering of Key-Presses

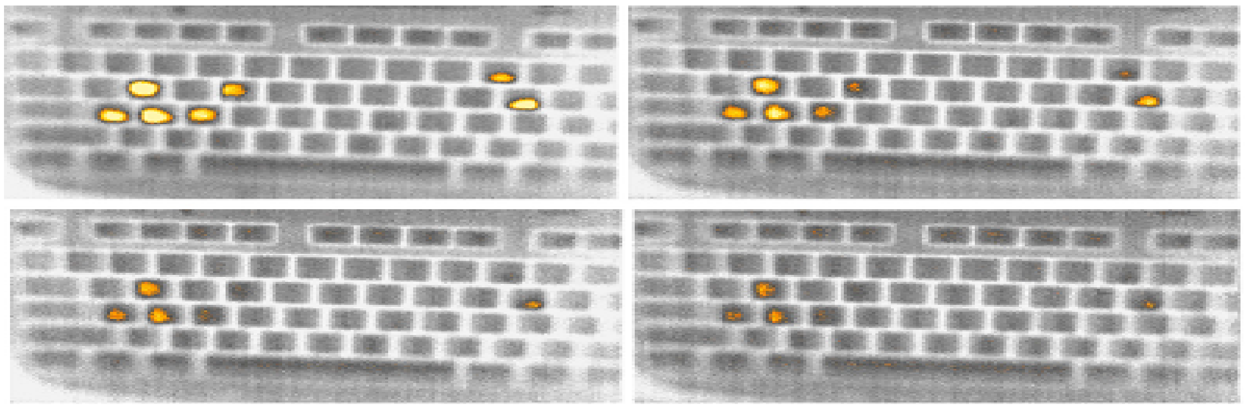


Figure 2.19: Password "passw0rd" thermal residue after 0(top left), 15 (top right), 30 (bottom left), and 45 (bottom right) seconds after entry

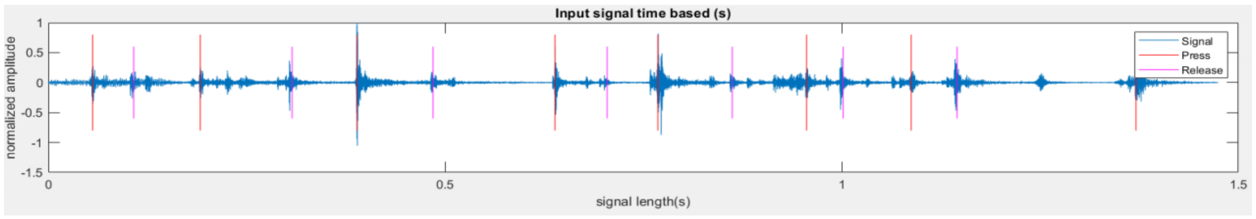


Figure 2.20: Acoustic Emanations of Password "jordan23"

Unfortunately, inspection of thermal images by stage 2 subjects did not yield any reliable key-press ordering information. Newton's Law of Cooling might seem to indicate that any reduction in heat energy would occur uniformly across all pressed keys, resulting in exposure of ordering. However, this is not true in practice. One reason is due to by *keystroke inconsistency* in the dynamics of Touch typists. Factors, such as the travel distance between keys and the particular finger used to press a key, result in small differences in the duration, and total surface area of, contact. Since each key-press is distinct, intensity of a given thermal residue does not correspond to its relative position in the target password. This holds even for Hunt-and-Peck typists, who tend to use only their index fingers. As evidenced by Figure 2.19, Hunt-and-Peck typist does not necessarily press keys with uniform force or for a uniform duration. These inconsistencies make reliable ordering of key-presses infeasible in our analysis framework. However, as mentioned above, for insecure (language-based) passwords, dictionary tools can be used to infer the most likely key-press order.

Anagram Solvers There are many anagram-finder tools such as the Anagrammer[6] or the Anagram Solver[7] which the adversary can use to find possible natural-language password candidates from observed keys. For alphanumeric passwords, solvers can be used with intuitive substitutions (e.g. "3" as a substitute for "e" or "0" for "o".) This can be done to find both individual natural-language words such as "football" as well as more complicated multiple-word phrases such as "iloveyou", with relative ease. Therefore, even though ordering is not apparent from the thermal image, the adversary can easily derive it for "insecure" passwords.

2.7.6 Mitigation Strategies

There are several simple strategies to mitigate or reduce the threat of Thermanator Attacks, without modifying any existing hardware. The most intuitive solution is to introduce *Chaff typing* right after a password is entered. This can be as simple as asking the users to swipe their hands along the keyboard after password entry, or requiring them to introduce noise by typing arbitrary "chaff". This would serve to obscure the password by introducing useless thermal residues, and thus make the password key-presses much more difficult to retrieve. Another way is to avoid keyboard entry altogether and use the mouse to select (click on) password characters displayed on the on-screen keyboard. A variation is to have drop-down

[6]See: `https://www.wordplays.com/anagrammer/`
[7]See: `https://www.thewordfinder.com/anagram-solver/`

menu for each position of the password and the user selects each character individually. A more burdensome alternative is to use the keyboard arrow keys to adjust a random character string (displayed on the screen) to the actual password. All such methods are well-known and are quite viable. However, they are more vulnerable to Shoulder-Surfing Attacks, due to the ease of watching a victim's larger, visible screen instead of their smaller, partially occluded keyboard. Finally, a user who is willing to go to extreme lengths to avoid leaving thermal residues could wear insulating gloves or rubber thimblettes over their fingers during password entry. This would greatly reduce thermal residues, and make Thermanator ineffective, since thermal conductivity of the insulating material would be much less than that of human skin.

If hardware changes are possible, other mitigation techniques might be appropriate. For example, a touch-screen would allow password entry without the use of a keyboard. However, this would be more (than keyboard entry) vulnerable to Shoulder-Surfing Attacks. Also, the use of touch-screens opens the door for attacks that exploit smudge patterns left behind by fingers [105]. Alternatively, common plastic keyboards could be replaced with metallic ones. Metals have much higher thermal conductivity than plastics. Thus, any localized thermal residues very quickly dissipate throughout the keyboard. A similar strategy was adopted to protect ATMs from thermal attacks [109].

2.8 Comparison with Similar Attacks

We now compare Thermanator with several similar human factors-based insider attacks. We focus on several aspects: adversary's *Goal*, any *Required Equipment*, the *Timeliness* requirements, whether a *Careless Victim* is needed, and finally, if *Prior Profiling* of the victim is required. Summary of the comparison is shown in Table 2.1.

Table 2.1: Feature Comparison of Common Human-Based Attack Types.

Attack Type	**Attack Goal**	**Adversary Timeliness**	**Careless Victim?**	**Equipment Needed**	**Prior Profiling Required?**
Lunch-Time	Hijack Log-in Session	15 min (default)	YES	None	NO
Shoulder-Surfing	Password	Real-Time	YES	Pair of Eyes or Video Camera	NO
Acoustic Emanations	Password	Real-Time	NO	Audio Recorder	YES
Keyboard Vibrations	Password	Real-Time	NO	Accelerometer	YES
Thermanator	Password	Up to 1 min	NO	Thermal Camera	NO

2.8.1 Lunch-Time

Lunch-Time Attacks are performed by the insider adversary who relies on a careless victim that neglects to terminate their secure log-in session [163].

- *Objective*: to gain access to a single secure (authenticated) session.
- *Required Equipment*: none, the adversary only needs to physically access the computer once the victim leaves.
- *Timeliness:* determined by the de-authentication technique(s) used by the victim. For example, the default inactivity timeout for Windows machines is a generous 15 minutes.
- *Careless Victim*: required for this attack to work. At the minimum, the victim needs to leave their workstation unattended without logging out or locking the screen.
- *Profiling*: no prior victim profiling is needed. The adversary can be opportunistic; it gains access to an authenticated session with out any additional or prior knowledge required.

2.8.2 Shoulder-Surfing

Shoulder-Surfing Attacks are performed by the insider adversary who looks over the shoulder of a careless victim while the password is entered. It can also be performed with the aid of a (perhaps hidden) camera pointed at the victim's keyboard, in which case adversarial presence is not required.

- *Objective*: to learn the victim's password.
- *Required Equipment*: none, though a video camera can be useful.
- *Timeliness* real-time, as the adversary must watch victim password entry as it occurs.
- *Careless Victim*: required, since the adversary has to stand over the victim's terminal to watch them type in their password. Careless victim is not required in case of a pre-placed viceoa camera.
- *Profiling*: no prior victim profiling is needed and the adversary can be opportunistic: it learns the victim's password with no additional or prior knowledge.

2.8.3 Acoustic Emanations

Acoustic Emanations Attacks are performed by the insider adversary who instruments the victim's environment with an audio recording device and exploits acoustic dynamics [115]

- *Objective*: learn the victim's password.
- *Required Equipment*: an audio recording device, placed nearby.
- *Timeliness*: real-time, since the adversary must record the keyboard sounds instantaneously.
- *Careless Victim*: not required; the recording device can be hidden from view.
- *Profiling*: prior victim profiling is needed; the adversary must build an acoustic profile of the victim to accurately interpret keystroke sounds.

2.8.4 Keyboard Vibrations

Vibration Attacks are performed by the insider adversary using an accelerometer to record vibrations created by a victim typing into a keyboard, in order to reconstruct what was typed [110].

- *Objective*: learn the victim's password.
- *Required Equipment*: an accelerometer, placed nearby (closer than in Acoustic Emanations Attacks).
- *Timeliness*: real-time, since the adversary must record the victim's vibrations instantaneously.
- *Careless Victim*: not required; the recording device can be hidden from view.
- *Profiling*: prior victim profiling is needed; the adversary must build a vibration profile in order to accurately interpret keystroke vibration patterns.

2.8.5 Thermanator

Thermanator Attacks are performed by an insider adversary who records thermal residues users after recent password entry.

- *Objective*: learn the victim's password.
- *Required Equipment*: thermal camera.
- *Timeliness*: up to 1 minute, the adversary must record the keyboard before thermal residues dissipate.
- *Careless Victim*: not required; recording/imaging takes place after the victim leaves.
- *Profiling*: prior victim profiling not needed. The adversary does not need any prior knowledge of the victim to analyze thermal images (though it obviously helps, especially with insecure passwords).

2.9 Related Work

Thermal residue side-channel has been shown to be an avenue for obtaining secrets (e.g., key-codes, PINs) with [175]. [109] investigated the influence of material composition (metal vs. plastic) and camera distance (14 vs. 28 inches) on PIN recovery, using a US$17,950 thermal camera, on commercial PoS-style PIN pads. [113] explored the effectiveness of a low-cost thermal camera ($\approx US\$330$, attachable to a smartphone) to recover 4-digit PINs entered into rubber keypads. Lastly, [173] discussed the viability of thermal imaging attacks on various PIN-entry devices including a keyboard, digital door lock, cash machine and payment terminal. Analysis showed that the attack was a credible threat. The attack on keyboards was to recover a 4-digit PIN entry and did not consider passwords.

[106] investigated using a thermal camera to infer screen-lock patterns of smartphones. In a similar effort, [107] conducted more extensive experiments to assess efficacy of thermal imaging attacks against screen-lock patterns. It was shown that PINs were vulnerable to such an approach, while swipe-patterns were not.

2.10 Conclusion

As formerly niche sensing devices become less and less expensive, new side-channel attacks move from "Mission: Impossible" towards reality. This strongly motivates exploration of novel human-factors attacks, such as those based on Thermanator. Work described in this chapter sheds some light on understanding the thermodynamic relationship between human fingers and external computer keyboards. In particular, it exposes the vulnerability of standard password-based systems to adversarial collection of thermal emanations.

Based on the study results, we believe that Thermanator Attacks represent a new credible threat for password-based systems, and that human-induced thermal side-channels deserve further study. This is especially true considering constantly decreasing costs and increasing availability of high-quality thermal imagers. It is realistic to expect that – in several years' time – thermal imagers that can be attached to smartphones, e.g., FLIR One, will offer the quality equivalent to SC620 that was used in our study. This would allow surreptitious collection of thermal images without bulky, unusual or suspicious-looking equipment. Cameras in the price range of our SC620 would offer the image quality of A6700sc, with time-windows for collecting thermal residues that last for several minutes.

Chapter 3

Captcha Avoidance via Client-side TEE Integration

3.1 Introduction

In traditional systems, software with more privilege (e.g., hypervisor, operating system) can interfere with data destined for a program on the system. One solution to this problem is to utilize Trusted Execution Environments (TEEs) [160]. A TEE isolates the execution of a program from other (system) software and additionally may offer secure I/O (Input/Output). For example, TrustZone allows locking GPIO (General Purpose Input/Output) ports to be used only by the secure/trusted code [2]. Intel SGX [33], on the other hand, does not support such a feature. Yet, due to its availability on PCs [51], it is a suitable target platform for developing TEE-based applications.

One common denominator of TEEs is that they provide isolated code execution. Utilizing this feature, we change our focus of attention to TEE-based Web solutions that revolve around privacy. We show how TEEs (more specifically Intel SGX) can be used to implement a privacy-preserving CAPTCHA alternative that focuses on rate-limiting. This is in response to the transition of emerging CAPTCHAs from being *hard-to-solve AI problems* [168] to *behavioral-based* [47] ones which collect user data.

In this chapter, we present CACTI: **C**APTCHA **A**voidance via **C**lient-side **T**EE **I**ntegration. Using client-side TEEs, CACTI allows legitimate clients to generate unforgeable *rate-proofs* demonstrating how frequently they have performed specific actions. These rate-proofs can be sent to web servers in lieu of solving CAPTCHAs. CACTI provides strong client privacy guarantees, since the information is only sent to the visited website and authenticated using a group signature scheme. Our evaluations show that overall latency of generating and verifying a CACTI rate-proof is less than 0.25 sec, while CACTI's bandwidth overhead is over 98% lower than that of current CAPTCHA systems.

3.2 Motivation & Contributions

In the past two decades, as Web use became almost universal and abuse of Web services grew dramatically, there has been an increasing trend (and real need) to use security tools that help prevent abuse by automated means, i.e., so-called **bots**. The most popular mechanism is CAPTCHAs: Completely Automated Public Turing test to tell Computers and Humans Apart [169]. A CAPTCHA is essentially a puzzle, such as an object classification task (Figure 3.1a) or distorted text recognition (see Figure 3.1b), that aims to confound (or at least slow down) a bot, while being easily[1] solvable by a human user. CAPTCHAs are often used to protect sensitive actions, such as creating a new account or submitting a web form.

Although primarily intended to distinguish humans from bots, it has been shown that CAPTCHAs are not very effective at this task [146]. Many CAPTCHAs can be solved by algorithms (e.g., image recognition software) or outsourced to human-driven *CAPTCHA-*

[1]Exactly what it means to be "easily" solvable is subject to some debate.

farms[2] to be solved on behalf of bots. Nevertheless, CAPTCHAs are still widely used to increase the adversary's costs (in terms of time and/or money) and reduce the *rate* at which bots can perform sensitive actions. For example, computer vision algorithms are computationally expensive, and outsourcing to CAPTCHA-farms costs money and takes time.

From the users' perspective, CAPTCHAs are generally unloved (if not outright hated), since they represent a barrier and an annoyance (a.k.a. Denial-of-Service) for legitimate users. Another major issue is that most CAPTCHAs are visual in nature, requiring sufficient ambient light and screen resolution, as well as good eyesight. Much less popular audio CAPTCHAs are notoriously poor, and require a quiet setting, decent-quality audio output facilities, as well as good hearing.

More recently, the reCAPTCHA approach has become popular. It aims to reduce user burden by having users click a checkbox (Figure 3.1c), while performing behavioral analysis of the user's browser interactions. Acknowledging that even this creates friction for users, the latest version ("invisible reCAPTCHA") does not require any user interaction. However, the reCAPTCHA approach is potentially detrimental to **user privacy** because it requires maintaining long-term state, e.g., in the form of Google-owned cookies. Cloudflare recently decided to move away from reCAPTCHA due to privacy concerns and changes in Google's business model [39].

Notably, all current CAPTCHA-like techniques are server-side, i.e., they do not rely on any security features of, or make any trust assumptions about, the client platform. The purely server-side nature of CAPTCHAs was reasonable when client-side hardware security features were not widely available. However, this is rapidly changing with the increasing popularity of Trusted Execution Environments (TEEs) on a variety of computing platforms, e.g., TPM and Intel SGX for desktops/laptops and ARM TrustZone for smartphones and even smaller devices. Thus, it is now realistic to consider abuse prevention methods that include client-

[2] A CAPTCHA farm is usually sweatshop-like operation, where employees solve CAPTCHAs for a living.

side components. For example, if a TEE has a trusted path to some form of user interface, such as a mouse, keyboard, or touchscreen, this *trusted User Interface (UI)* could securely confirm user presence. Although this feature is still unavailable on most platforms, it is emerging through features like Android's Protected Confirmation [94]. This approach's main advantages are minimized user burden (e.g., just a mouse click) and increased security, since it would be impossible for software to forge this action. Admittedly however, this approach can be defeated by adversarial hardware e.g., a programmable USB peripheral that pretends to be a mouse or keyboard.

However, since the majority of consumer devices do not currently have a trusted UI, it would be highly desirable to reduce the need for CAPTCHAs using only existing TEE functionality. As discussed above, the main goal of modern CAPTCHAs is to increase adversarial costs and reduce the *rate* at which they can perform sensitive actions. Therefore, if legitimate users had a way to prove that their rate of performing sensitive actions is below some threshold, a website could decide to allow these users to proceed without solving a CAPTCHA. If a user can not provide such a proof, the website could simply fall back to using CAPTCHAs. Though this would not fully prevent bots, it would not give them any advantage compared to the current arrangement of using CAPTCHAs.

Motivated by the above discussion, this chapter presents CACTI, a flexible mechanism for allowing legitimate users to prove to websites that they are not acting in an abusive manner. By leveraging widespread and increasing availability of client-side TEEs, CACTI allows users to produce *rate-proofs*, which can be presented to websites in lieu of solving CAPTCHAs. A rate-proof is a simple assertion that:

1. The rate at which a user has performed some action is below a certain threshold, and
2. The user's time-based counter for this action has been incremented.

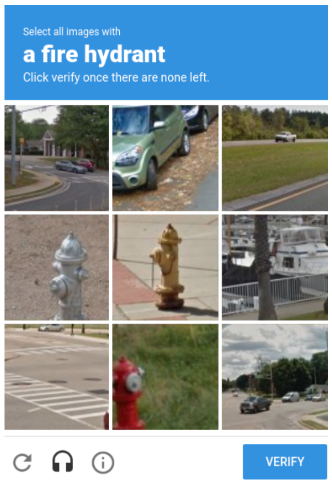
When serving a webpage, the server selects a *threshold* value and sends it to the client. If the client can produce a rate-proof for the given threshold, the server allows the action to proceed without showing a CAPTCHA. Otherwise, the server presents a CAPTCHA, as before. In essence, CACTI can be seen as a type of "express checkout" for legitimate users.

One of the guiding principles and goals of CACTI is user privacy – it reveals only the minimum amount of information and sends this directly to the visited website. Another principle is that the mechanism should not mandate any specific security policy for websites. Websites can define their own security policies e.g., by specifying thresholds for rate-proofs. Finally, CACTI should be configurable to operate without any user interaction, in order to make it accessible to all users, including those with sight or hearing disabilities.

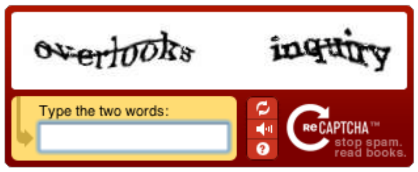
Although chiefly motivated by the shortcomings of CAPTCHAs, we believe that the general approach of client-side (TEE-based) rate-proofs, can also be used in other common web scenarios. For example, news websites could allow users to read a limited number of articles for free per month, without relying on client side cookies (which can be cleared) or forcing users to log-in (which is detrimental to privacy). Online petition websites could check that users have not signed multiple times, without requiring users to provide their email addresses, which is once again, detrimental to privacy. We therefore believe that our TEE-based rate-proof concept is a versatile and useful web security primitive.

Anticipated contributions of this work are:

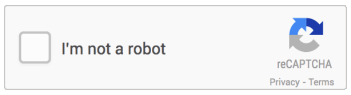
1. We introduce the concept of a *rate-proof*, a versatile web security primitive that allows legitimate users to securely prove that their rate of performing sensitive actions falls below a server-defined threshold.
2. We use the rate-proof as the basis for a concrete client-server protocol that allows legitimate users to present rate-proofs in lieu of solving CAPTCHAs.
3. We provide a proof-of-concept implementation of CACTI, over Intel SGX, realized as a Google Chrome browser extension.



(a) Image-based object recognition re-CAPTCHA [47]



(b) Image-based text recognition re-CAPTCHA [47]



(c) Behavior-based reCAPTCHA [47]

Figure 3.1: Examples of CAPTCHAs

4. We present a comprehensive evaluation of security, latency, and deployability of CACTI.

3.3 Background

3.3.1 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a primitive that protects confidentiality and integrity of security-sensitive code and data from untrusted code. A typical TEE provides the following features:

Isolated execution. The principal function of a TEE is to provide an execution environment that is isolated from all other software on the platform, including privileged system software, such as the OS, hypervisor, or BIOS. Specifically, data inside the TEE can only be accessed by the code running inside the TEE. The code inside the TEE provides well-defined entry points (e.g., call gates), which are enforced by the TEE.

Remote attestation. Remote attestation provides a remote party with strong assurances about the TEE and the code running therein. Specifically, the TEE (i.e., the *prover*) creates a cryptographic assertion that: (1) demonstrates that it is a genuine TEE, and (2) unambiguously describes the code running in the TEE. The remote party (i.e., the *verifier*) can use this to decide whether to trust the TEE, and then to bootstrap a secure communication channel with the TEE.

Data sealing. Data sealing allows the code running inside the TEE to encrypt data such that it can be securely stored outside the TEE. This is typically implemented by providing the TEE with a symmetric *sealing key*, which can be used to encrypt/decrypt the data. In current TEEs, sealing keys are platform-specific, meaning that data can only be unsealed on the same platform on which it was sealed.

Hardware monotonic counters. A well known attack against sealed data is *rollback*, where the attacker replaces the sealed data with an older version.Mitigating this requires at least some amount of rollback-protected storage, typically realized as a hardware monotonic counter. When sealing, the counter can be incremented and the latest value is included in the sealed data. When unsealing, the TEE checks that the included value matches the current hardware counter value. Since hardware counters themselves require rollback-protected storage, TEEs typically only have a small number of counters.

One prominent TEE example is *Intel Software Guard Extensions (SGX)* [65, 127, 144]. SGX is a hardware-enforced TEE available on Intel CPUs from the Skylake microarchitecture onwards. SGX allows applications to create isolated environments, called *enclaves*, running in the application's virtual address space. A special region in physical memory is reserved for enclaves, called the Enclave Page Cache (EPC). The EPC can hold up to 128MB of code and data, shared between all running enclaves. When enclave data leaves the CPU boundary, it is transparently encrypted and integrity-protected by CPU's Memory Encryption Engine (MEE) to defend against physical bus snooping/tampering attacks. Since enclaves run in the application's virtual address space, enclave code can access all the memory of its host application, even that outside the enclave. Enclave code can only be called via predefined function calls, called `ECALL`s.

Every enclave has an enclave identity (`MRENCLAVE`), which is a cryptographic hash of the code that has been loaded into the enclave during initialization, and various other configuration details. Each enclave binary must be signed by the developer, and the hash of the developer's public key is stored as the enclave's signer identity (`MRSIGNER`).

SGX provides two types of attestation: local and remote. Local attestation allows two enclaves running on the same platform to confirm each other's identity and communicate securely, even though this communication goes via the untrusted OS. SGX uses local attestation to build remote attestation. Specifically, an application enclave performs local attestation with an Intel-provided *quoting enclave*, which holds a group private key provisioned by Intel. The quoting enclave verifies the local attestation and creates a signed *quote*, which includes the application enclave's and signer's identities, as well as user-defined data provided by the application enclave. This quote is sent to the remote verifier, which, in turn, uses the Intel Attestation Service (IAS) to verify it. Since the attestation uses a group signature scheme, the verifier cannot determine whether two quotes were generated by the same platform.

In SGX, data can be sealed in one of two modes, based on: (1) the enclave's identity, such that only the same type of enclave can unseal it, or (2) the signer identity, such that any enclave signed by the same developer (running on the same platform) can unseal it. SGX provides hardware monotonic counters and allows each enclave to use up to 256 counters at a time.

3.3.2 Group Signatures

A group signature scheme aims to prevent the verifier from determining the group member which generated the signature. Each group member is assigned a group private key under a single group public key. In case a group member needs to be revoked, a special entity called *group manager* can open the signature. A group signature scheme is composed of five algorithms [70]:

- **Setup:** Given a security parameter, an efficient algorithm outputs a group public key and a master secret for the group manager.
- **Join:** A user interacts with the group manager to receive a group private key and a membership certificate.
- **Sign:** Using the group public key, group private key, membership certificate, and a message m, a group member generates a group signature of m.
- **Verify:** Using the group public key, an entity verifies a group signature.
- **Open:** Given a message, a putative signature on the message, the group public key and the master secret, the group manager determines the identity of the signer.

A secure group signature scheme satisfies the following properties [70]:

- **Correctness:** Signatures generated with any member's group private key must be verifiable by the group public key.
- **Unforgeability:** Only an entity that holds a group private key can generate signatures.
- **Anonymity:** Given a group signature, it must be computationally hard for anyone (except the group manager) to identify the signer.
- **Unlinkability:** Given two signatures, it must be computationally hard to determine whether these were signed by the same group member.
- **Exculpability:** Neither a group member nor the group manager can generate signa-

tures on behalf of other group members.

- **Traceability:** The group manager can determine the identity of a group member that generated a particular signature.
- **Coalition-resistance:** Group members cannot collude to create a signature that cannot be linked to one of the group members by the group manager.

Enhanced Privacy ID (EPID) [82] is a group signature scheme used by remote attestation of Intel SGX enclaves. It satisfies the above properties whilst providing additional privacy-preserving revocation mechanisms to revoke compromised or misbehaving group members. Specifically, EPID's *signature-based revocation* protocol does not "Open" signatures but rather uses a signature produced by the revoked member to notify other entities that this particular member has been revoked.

3.4 System & Threat Models

The ecosystem that we consider includes three types of principals/players: (1) servers, (2) clients, and (3) TEEs. There are multitudes of these three principal types. The number of clients is the same as that of TEEs, and each client houses exactly one TEE. Even though a TEE is assumed to be physically within a client, we consider it to be separate security entity. Note that a human user can, of course, operate or own multiple clients, although there is clearly a limit and more clients implies higher costs for the user.

We assume that all TEEs are trusted: honest, benign and insubvertible. We consider all side-channel and physical attacks against TEEs to be out of scope of this work and assume that all algorithms and cryptographic primitives implemented within TEEs are impervious to such attacks. We also consider cuckoo attacks, whereby a malicious client utilizes multiple (possibly malware infected) machines with genuine TEEs, to be out of scope, since clients and their TEEs are not considered to be strongly bound. We refer to [176] and [103] as far as means for countering such attacks. We assume that servers have a means to authenticate and attest TEEs, possibly with the help of the TEE manufacturer.

All clients and servers are untrusted, i.e., they may act maliciously. The goal of a malicious client is to avoid CAPTCHAs, while a malicious server either aims to inconvenience a client (via DoS) or violate client's privacy. For example, a malicious server can try to learn the client's identity or link multiple visits by the same client. Also, multiple servers may collude in an attempt to track clients.

Our threat model yields the following requirements for the anticipated system:

- **Unforgeability:** Clients cannot forge or modify CACTI rate-proofs.
- **Client privacy:** A server (or a group thereof) cannot link rate-proofs to the clients that generated them.

We also pose the following non-security goals:

- **Latency:** User-perceived latency should be minimized.
- **Data transfer:** The amount of data transfer between client and server should be minimized.
- **Deployability:** The system should be deployable on current off-the-shelf client and server hardware.

3.5 CACTI Design & Challenges

This section discusses the overall design of CACTI and justifies our design choices.

3.5.1 Conceptual Design

Rate-proofs. The central concept underpinning our design is the *rate-proof.* Conceptually, the idea is as follows: Assuming that a client has an idealized TEE, it stores one or more named sorted lists of timestamps in its rollback-protected secure memory. To create a rate-proof for a specific list, the TEE is given the name of the list, a *threshold* (Th), and a new timestamp (t). The threshold is expressed as a starting time (t_s) and a count (k). This can be interpreted as: *"no more than k timestamps since t_s"*. The TEE checks that the specified list contains k or fewer timestamps with values greater than or equal to t_s. If so, it checks if the new timestamp t is greater than the latest timestamp in the list. If both checks succeed, the TEE pre-pends t to the list and produces a signed statement confirming that the named list is below the specified threshold and the new timestamp has been added. If either check fails, no changes are made to the list and no proof is produced. Note that the rate-proof does not disclose the number of timestamps in the list.

Furthermore, each list can also be associated with a public key. In this case, requests for rate-proofs must be accompanied by a signature over the request which is computed with a corresponding private key. This allows the system to enforce a *same-origin* policy for specific lists – proofs over such lists can only be requested by the same entity that created them. Note that this does not provide any binding to the *identity* of the entity holding the private key, as doing so would necessitate the TEE to check identities against a global public key infrastructure (PKI) and we prefer for CACTI not to require it.

Rate-proofs differ from rate *limits* because the user is allowed to perform the action any number of times. However, once the rate exceeds the specified threshold, the user will no longer be able to produce rate-proofs. The client can always decide to *not* use its TEE; this covers clients who do not have TEEs or those whose rates exceeded the threshold in addition to those who choose not to use CACTI at specific times. On the other hand, if the server does not yet support CACTI, the client does not store any timestamps, or perform any additional computation.

CAPTCHA-avoidance. In today's CAPTCHA-protected services, the typical interaction between the client (C) and server (S) proceeds as follows:

1. C requests access to a service on S.
2. S returns a CAPTCHA for C to solve.
3. C submits the solution to S.
4. If the solution is verified, S allows C access to the service.

Although current approaches, e.g., reCAPTCHA, might include additional steps (e.g., communicating with third-party services), these can be abstracted into the above pattern.

CACTI keeps the same interaction sequence, while substituting steps 2 and 3 with rate-proofs. Specifically, in step 2, the server sends a threshold rate and the current timestamp. In step 3, instead of solving a CAPTCHA, the client generates a rate-proof with the specified threshold and timestamp, and submits it to the server. The server has two types of lists:

- **Server-specific:** The server requests a rate-proof over its own list. The name of the list could be the server's URL, and the request may be signed by the server. This determines the rate at which the client visits this specific server.
- **Global:** The server requests a rate-proof over a global list, with a well-known name, e.g. `CACTI-GLOBAL`. This yields the rate at which the client visits all servers that use the global list.

The main idea of CAPTCHA avoidance is that a legitimate client should be able to prove that its rate is below the server-defined threshold. In other words, the server should have sufficient confidence that the client is not acting in an abusive manner (where the threshold of between abusive and non-abusive behaviors is set by the server). Servers can select their own thresholds according to their own security requirements. A given server can vary the threshold across different actions or even across different users or user groups, e.g., lower thresholds for suspected higher-risk users. If a client cannot produce a rate-proof, or is unwilling to do so, the server simply reverts to the current approach of showing a CAPTCHA. CACTI essentially provides a *fast-pass* for legitimate users.

The original CAPTCHA paper [169] suggested that CAPTCHAs could be used in the following scenarios:

1. **Online polls:** to prevent bots from voting,
2. **Free email services:** to prevent bots from registering for thousands of accounts,
3. **Search engine bots:** to preclude or inhibit indexing of websites by bots,
4. **Worms and spam:** to ensure that emails are sent by humans,
5. **Preventing dictionary attacks.** to limit the number of password attempts.

As discussed in Section 3.2, it is unrealistic to assume that CAPTCHAs cannot be solved by bots (e.g., using computer vision algorithms) or outsourced to CAPTCHA farms. Therefore, we argue that all current uses of CAPTCHAs are actually intended to slow down attackers or increase their costs. In the list above, scenarios 2 and 5 directly call for rate-limiting, while scenarios 1, 3, and 4 can be made less profitable for attackers if sufficiently rate-limited. Therefore, CACTI can be used in all these scenarios.

In addition to CAPTCHAs, modern websites use a variety of abuse-prevention systems (e.g., filtering based on client IP address or cookies). We envision CACTI being used alongside such mechanisms. Websites could dynamically adjust their CACTI rate-proof thresholds based on information from these other mechanisms. We are aware that rate-proofs are a versatile primitive that could be used to fight abusive activity in other ways, or even enable new use-cases. However, in this chapter, we focus on the important problem of reducing the user burden of CAPTCHAs.

3.5.2 Design Challenges

In order to realize the conceptual design outlined above, we identify the following key challenges:

TEE attestation. In current TEEs, the process of remote attestation is not standardized. For example, in SGX, a verifier must first register with Intel Attestation Service (IAS) before it can verify TEE quotes. Other types of TEEs would have different processes. It is unrealistic to expect every web server to establish relationships with such services from all manufacturers in order to verify attestation results. Therefore, web servers cannot directly verify the attestation, but still need to ascertain that the client is running a genuine TEE.

TEE memory limitations. TEEs typically have a small amount of secure memory. For example, if the memory of an SGX enclave exceeds the size of the EPC (usually 128 MB), the CPU has to swap pages out of the EPC. This is a very expensive operation, since these pages must be encrypted and integrity protected. Therefore, CACTI should minimize the required amount of enclave memory, since other enclaves may be running on the same platform.

Limited number of monotonic counters. TEEs typically have a limited number of hardware monotonic counters, e.g., SGX allows at most 256 per enclave. Also, the number of counter increments can be limited, e.g., in SGX the limit is 100 in a single epoch [28] – a platform power cycle, or a 24 hour period. This is a challenge because hardware monotonic counters are critical for achieving rollback-protected storage. Recall that CACTI requires rollback-protected storage for all timestamps, to prevent malicious clients from rolling-back the timestamp lists and falsifying rate-proofs. Furthermore, this storage must be updated every time a new timestamp is added, i.e., for each successful rate-proof.

TEE entry/exit overhead. Invoking TEE functionality typically incurs some overhead. For example, whenever an execution thread enters/exits an SGX enclave, the CPU has

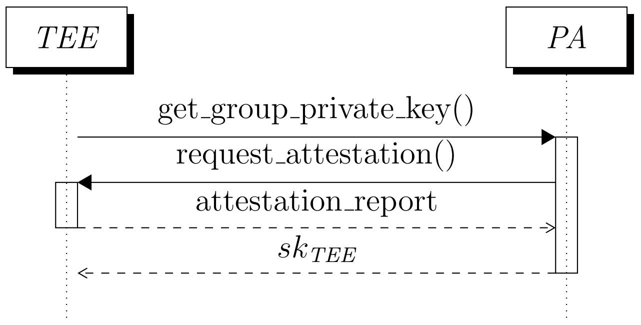


Figure 3.2: CACTI provisioning protocol. The interaction between the Provisioning Authority (PA) and the client's TEE takes place over a secure connection, using the client to pass the encrypted messages. After verifying the attestation report (and any other required information), the PA provisions the TEE with a group private key (sk_{TEE}).

to perform various checks and procedures (e.g., clearing registers) to ensure that enclave data does not leak. Identifying and minimizing the number of TEE entries/exits, whilst maintaining functionality, can be challenging.

3.5.3 Realizing CACTI Design

We now present a detailed design that addresses aforementioned design challenges. We describe its implementation in Section 3.6.

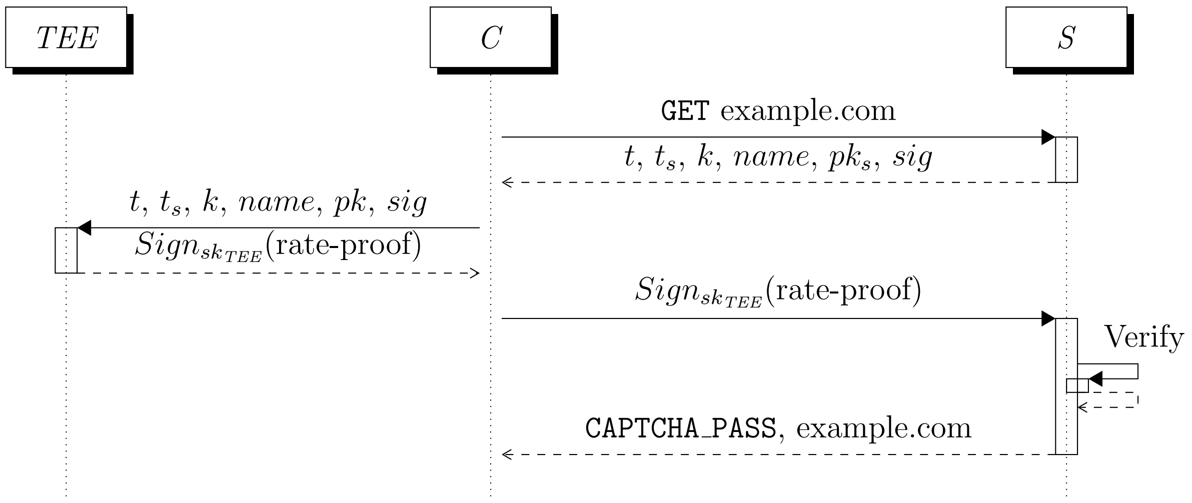
Communication protocol

The web server needs to determine that a supplied rate-proof was produced by a genuine TEE. Typically, this would be done using remote attestation, where the TEE proves that it is running CACTI code. If the TEE provides privacy-preserving attestation (e.g., the EPID protocol used in SGX remote attestation), this would also fulfill our requirement for client privacy, since websites would not be able to link rate-proofs to specific TEEs.

However, as described above, current TEE remote attestation is not designed to be verified by anonymous third parties. Furthermore, as CACTI is not limited to any particular TEE type, websites would need to understand attestation results from multiple TEE vendors, potentially using different protocols. Finally, some types of TEEs might not support privacy-preserving remote attestation, which would undermine our requirement for client privacy.

To overcome this challenge, we introduce a separate *Provisioning Authority* (*PA*) in order to unify various processes for attesting CACTI TEEs. Fundamentally, the *PA* is responsible for verifying TEE attestation (possibly via the TEE vendor) and establishing a privacy-preserving mechanism through which websites can also establish trust in the TEE. Specifically, the *PA* protects user privacy by using the EPID group signature scheme. The *PA* plays the role of the EPID *issuer*, and – optionally – the *revocation manager* [82]. During the provisioning phase (as shown in Figure 3.2), the *PA* verifies the attestation from the client's TEE and then runs the EPID `join` protocol with the client's TEE in order to provision the TEE with a group private key sk_{TEE}. The *PA* certifies and publishes the group public key pk_G. The *PA* may optionally require the client to prove their identity (e.g., by signing into an account) – this is a business decision and different *PA*s may take different approaches. After provisioning, the *PA* is unable to link signatures to any specific client thanks to the properties of the underling BBS+ signature scheme and signature-based revocation used in EPID [82]. We analyze security implications of malicious *PA*s in Section 3.7.1, and discuss the use of other group signature schemes in Section 3.8.2. There can be multiple *PA*s and websites can decide which *PA*s to trust. If a TEE is provisioned by an unsupported *PA*, the website would fall back to using CAPTCHAs.

Once the TEE has been provisioned, the client can begin to use CACTI when visiting supported websites, as shown in Figure 3.3. Specifically, when serving a page, the server includes the following information: a timestamp t, a threshold Th (including start time t_s and count k), the name of the list (or CACTI-GLOBAL for the global list), and (optionally) a public key and signature for rates that enforce a same-origin policy. The client uses this information to



TEE
C
S
GET example.com
t, t_s, k, $name$, pk_s, sig
t, t_s, k, $name$, pk, sig
$Sign_{sk_{TEE}}$(rate-proof)
$Sign_{sk_{TEE}}$(rate-proof)
Verify
CAPTCHA_PASS, example.com

Figure 3.3: CACTI CAPTCHA-avoidance protocol. The client (C) requests a resource from the web server (S). In response, the server provides a timestamp for the current event (t), a threshold consisting of a starting time (t_s) and a count (k), and the *name* of the list. Optionally, the server also provides a signature (sig) over the request and the public key (pk_s) with which the signature can be verified. The client passes this information to its TEE in order to produce a rate-proof, signed by a group private key (sk_{TEE}), which can be verified by the server.

request a rate-proof from their TEE. If the client's rate is indeed below the threshold, the TEE produces the rate-proof, signed with its group private key. The client then sends this to the server in lieu of solving a CAPTCHA.

TEE Design

To realize the conceptual design above, the client's TEE would ideally store all timestamps indefinitely in integrity-protected and rollback-protected memory. However, as discussed above, current TEEs fall short of this idealized representation, since they have limited integrity-protected memory and a limited number of hardware counters for rollback protection. To overcome this challenge, we store all data outside the TEE, e.g., in a standard database. To prevent dishonest clients from modifying this data, we use a combination of hash chains and Merkle Hash Trees (MHTs) to achieve integrity and rollback-protection.

Hash chains of timestamps. To protect integrity of stored timestamps, we compute a hash chain over each list of timestamps, as shown in Figure 3.4. Thus the TEE only needs to provide integrity and rollback-protected storage for the most recent hash in each hash chain. For efficiency, we store intermediate value of the hash chain along with each timestamp outside the TEE.

MHT of lists. Although it would be possible for the TEE to seal the most recent hash of each list individually, the lists may be updated independently, so the TEE would need separate hardware monotonic counters to provide rollback protection for each list. In a real-world deployment, the number of lists is likely to exceed the number of available hardware counters, e.g., 256 counters per enclave in SGX. To overcome this challenge, we combine the lists into a Merkle Hash Tree (MHT). As shown in Figure 3.5, each leaf of the MHT is a hash of the list information (list name and public key) and the most recent hash in the list's hash chain. With this arrangement, the TEE only needs to provide integrity and rollback-protected storage for the MHT root R, which can be achieved using sealing and a single hardware monotonic counter.

Producing a Rate-Proof

The TEE first needs to verify the integrity of its externally-stored data structures (i.e., hash chains and MHT described above), and if successful, update these with the new timestamp and produce the rate-proof, as follows:

1. TEE inputs. The client supplies its TEE with the list information and all timestamps in the list that are greater than or equal to the server-defined start time t_s. The client also supplies the largest timestamp that is smaller than t_s, which we denote $t_{s-\delta}$, and the intermediate value of the hash chain up to, but not including, $t_{s-\delta}$. The client supplies the sealed MHT root and intermediate hashes required to verify that the list is in the MHT.

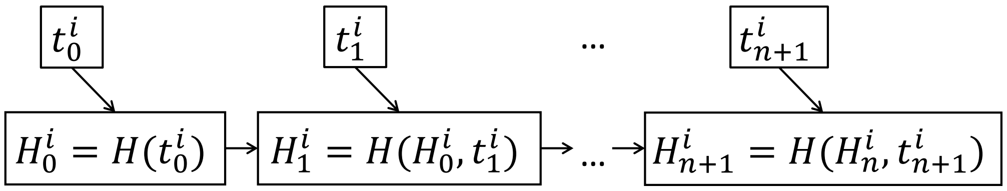


Figure 3.4: Hash chain of timestamps t_j^i for list i. $H()$ is a cryptographic hash function.

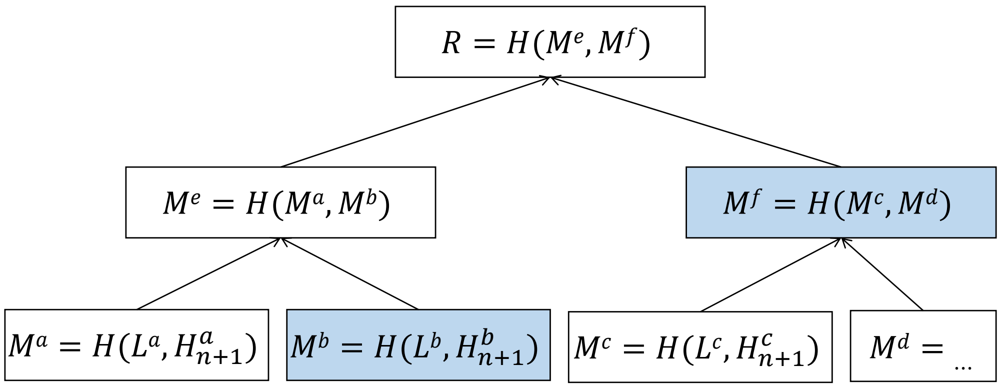

$R = H(M^e, M^f)$
$M^e = H(M^a, M^b)$
$M^f = H(M^c, M^d)$
$M^a = H(L^a, H_{n+1}^a)$
$M^b = H(L^b, H_{n+1}^b)$
$M^c = H(L^c, H_{n+1}^c)$
$M^d = ...$

Figure 3.5: Merkle Hash Tree over lists $a...d$. Each leaf is a hash of the list information L^i (list name and public key) and the most recent hash of the list's hash chain H_{n+1}^i. $H()$ is a cryptographic hash function, R is the root of the MHT, and the nodes in blue illustrate the inclusion proof path for list b.

2. Hash chain checks. The TEE first checks that $t_{s-\delta}$ is smaller than t_s and then recomputes the hash chain over included timestamps in order to reach the most recent value. During this process, it counts the number of included timestamps and checks that this is less than the value k specified in the threshold. The inclusion of one timestamp outside the requested range ($t_{s-\delta}$) ensures that the TEE has seen all timestamps within the range. This process requires $\mathcal{O}(n)$ hashes, where n is the number of timestamps in the requested range.

3. MHT checks. The TEE then unseals the MHT root and uses the hardware counter to verify that it is the latest version. The TEE then checks that the list information and the calculated most recent hash value is indeed a leaf in the MHT. This process requires $\mathcal{O}(log(s))$ hashes, where s is the number of lists. Including the list name in the MHT leaf ensures that the timestamps have not been substituted from another list. If the list has an associated public key, the TEE uses this to verify the signature on the server's request.

4. Starting a new list. If the rate-proof is requested over a new list (e.g., when the user firsts visits a website), the TEE must also verify that the list name does not appear in any MHT leaves. In this case, the client supplies the TEE with all list names and their most recent hash values. The TEE reconstructs the full MHT and checks that the new list name does not appear. This requires $\mathcal{O}(s)$ string comparisons and hashes for s lists.

5. Updating a list. If the above verification steps are successful, the TEE checks that the new timestamp t supplied by the server exceeds the latest timestamp in the specified list. If so, the TEE adds t to the list and updates the MHT to obtain a new MHT root. The new root is sealed alongside the TEE's group private key. The TEE then produces a signed rate-proof, using its group private key. The rate-proof includes a hash of the original request provided by the server, thus confirming that the TEE checked the rate and added the server-supplied timestamp. The TEE returns the rate-proof to the client, along with the new sealed MHT root for the client to store. In the above design, the whole process of producing the rate-proof can be performed in a single call to the TEE, thus minimizing the overhead of entering/exiting the TEE.

Reducing Client-Side Storage

The number of timestamps stored by CACTI grows as the client visits more websites. However, in most use-cases, it is unlikely that the server will request rate-proofs going back beyond a certain point in time t_P.

To reduce client-side storage requirements, we provide a mechanism to *prune* a client's timestamp list by merging all timestamps prior to t_P. Specifically, the server can include t_P in any rate-proof request, and upon receiving this, the client's TEE counts and records how many timestamps are older than t_P. The old timestamps and associated intermediate hash values can then be deleted from the database. In other words, the system merges all timestamps prior to t_P into a single count value c_P. The TEE stores t_P and the count value in the database outside the TEE and protects their integrity by including both values in the list information that forms the MHT leaf. Pruning can be done repeatedly: when a new pruning request is received for $t_{P'} > t_P$, CACTI fetches and verifies all timestamps up to $t_{P'}$ and adds these to c_P to create $c_{P'}$. It then replaces t_P and c_P with $t_{P'}$ and $c_{P'}$ respectively.

This pruning mechanism does not reduce security of CACTI. If the server does request a rate-proof going back beyond t_P, CACTI will include the full count of timestamps stored alongside t_P. This is always greater than or equal to the actual number of timestamps; thus, there is no incentive for the server to abuse the pruning mechanism. Similarly, even if a malicious client could trigger this pruning (i.e., assuming the list is not associated to the server's public key), there is no incentive to do so because it would never decrease the number of timestamps included in rate-proofs.

Since the global list CACTI-GLOBAL is used by all websites, the client is always allowed to prune this list to reduce storage requirements. CACTI blocks servers from pruning CACTI-GLOBAL since this can be used as an attack vector to inflate the client rate by compressing all rates into one value – thus preventing use of CACTI on websites that utilize CACTI-GLOBAL. Thus, we expect pruning of CACTI-GLOBAL to be done automatically by the CACTI host application or browser extension.

3.6 Implementation

We now describe the implementation of the CACTI design presented in the previous section. We focus on proof-of-concept implementations of: client-side browser extension, native host application, and CACTI TEE, as shown in Figure 3.6. Finally, we discuss how CACTI is integrated into websites.

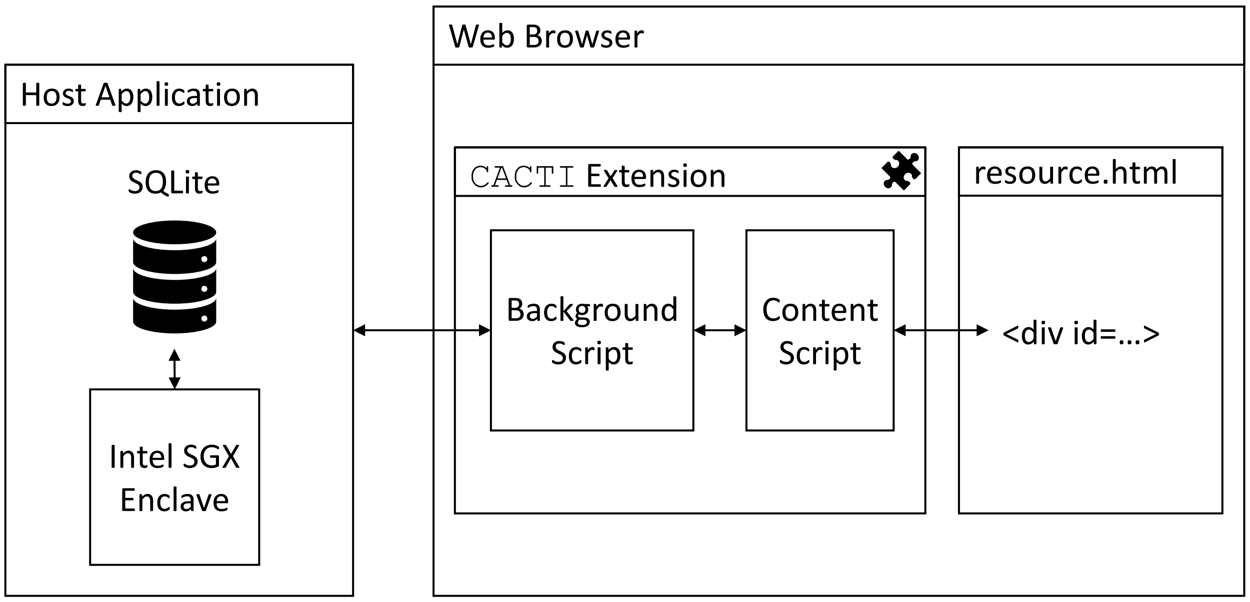


Figure 3.6: Overview of CACTI client-side components.

3.6.1 Browser Extension

The browser extension serves as a bridge between the web server and our host application. We implemented a proof-of-concept browser extension for the Chrome browser (build 79.0.3945.130) [23]. Chrome extensions consist of two parts: a content script and a background script.

- **Content script:** scans the visited web page for an HTML `div` element with the id CACTI-`div`. If the page contains this, the content script parses the parameters it contains and sends them to the background script.
- **Background script:** we use Chrome Native Messaging to launch the host application binary when the browser is started and maintain an open port [49] to the host application until the browser is closed. The background script facilitates communication between the content script and the host application.

User notification. The browser extension is also responsible for notifying the user about requests to access CACTI. Notifications can include information, such as server's domain name, timestamp to be inserted, and threshold used to generate the rate-proof. By default, the background script notifies the user whenever a server requests to use CACTI, and waits for user confirmation before proceeding. This prevents malicious websites from abusing CACTI by adding multiple timestamps without user permission (for possible attacks, see Section 3.7.1). However, asking for user confirmation for every request could cause UI fatigue. Therefore, CACTI could allow the user to choose from the following options: (1) *Always ask* (the default), (2) *Ask only upon first visit to site*, (3) *Only ask for untrusted sites*, (4) *Only ask for more than x requests per site per time period*, and (5) *Never ask*. Advanced users can also modify our extension or code their own extension to enforce arbitrary policies for requesting user confirmation. The notification is displayed using Chrome's Notification API [15].

3.6.2 Host Application

The host application running on the client is responsible for: (1) creating the CACTI TEE, which we implement as an SGX enclave, and exposing its `ECALL` API to the browser extension; (2) storing (and forwarding) timestamps and additional integrity information for secure calculation of rate-proofs (to the enclave); and (3) returning the enclave's output to the browser extension.

The host application is implemented in C and uses Chrome Native Messaging [40] to communicate with the browser extension. Since Chrome Native Messaging only supports communication with JSON objects, the host application uses a JSON parser to extract parameters to the API calls. We used the JSMN JSON parser [34]. Moreover, the host application implements the Chrome Native Messaging protocol [14] and communicates with the browser extension using Standard I/O (`stdio`), since this is currently the only means to communicate between browser extensions and native applications.

The host application stores information in an SQLite database. This database has two tables: `LISTS` stores the list names and associated public keys, and `TIMESTAMPS` stores all timestamps and intermediate values of the hash chains. For each rate-proof request, the host application queries the database and provides the data to the enclave.

Since the timestamps are stored unencrypted, we use existing features of the SQLite database to retrieve only the necessary range of timestamps for a given list. Note that since data integrity is maintained through other mechanisms (i.e., hash chains and MHT), the mechanism used by the host application to store this data does not affect the security of the system. Alternative implementations could use different database types and/or other data storage approaches. Instead of hash chains and MHTs, it is possible to use a database managed by the enclave, e.g., EnclaveDB [157]. However, this would increase the amount of code running inside the enclave, thus bloating the trusted code base (TCB).

3.6.3 SGX Enclave

We implemented the TEE as an SGX enclave using the OpenEnclave SDK [45] v0.7.0. OpenEnclave was selected since it aims to unify the programming model across different types of TEEs. The process of requesting a rate-proof is implemented as a single `get_rate` `ECALL`. For timestamps, we use the UNIX time which denotes the number of seconds elapsed since the UNIX Epoch (midnight 1/1/1970) and is represented as a 4-byte signed integer. We use cryptographic functions from the `mbed` TLS library [37] included in OpenEnclave. Specifically, we use SHA-256 for all hashes and ECDSA for all digital signatures. For EPID signatures, we use Intel EPID SDK (v7.0.1) [18] with the performance-optimized version of Intel Integrated Performance Primitives (IPP) Cryptography library [29]. We use a formally-verified and platform-optimized MHT implementation from EverCrypt [158]. As an optimization, if the MHT is sufficiently small, we can cache fully inside the enclave. When a request for a rate-proof is received, the enclave recalculates the timestamp hash chain and then directly compares the most recent value to the corresponding leaf in the cached MHT, as described in Section 3.5.3.

OpenEnclave currently does not support SGX hardware monotonic counters, so we could not include these in the proof-of-concept implementation. However, a production implementation can easily include hardware counter functionality. Although our implementation uses SGX, CACTI can be realized on any suitable TEE. For example, OpenEnclave is currently being updated to support ARM TrustZone. When this version is released, we plan to port the current implementation to TrustZone, with minimal expected modifications.

3.6.4 Website Integration

Integrating CACTI into a website involves two aspects: sending the rate-proof request to the client, and verifying the response. The server generates the rate-proof request (see Section 3.5.3) and encodes it as `data-*` attributes in the `CACTI-div` HTML `div`. The server also includes the URL to which the generated rate-proofs should be sent. The browser extension determines whether the website supports CACTI by looking for the `CACTI-div` element. The server implements an HTTP endpoint for receiving and verifying rate-proofs . If the verification succeeds, this endpoint notifies the website and the user is granted access.

Integrating CACTI into a website is thus very similar to using existing CAPTCHA systems. For example, reCAPTCHA adds the `g-recaptcha` HTML `div` to the page, and implements various endpoints for receiving and verifying the responses [48]. We evaluate server-side overhead of CACTI, in terms of both processing and data transfer requirements, in Section 3.7.

3.7 Evaluation

We now present and discuss the evaluation of CACTI. We start with a security analysis, based on the threat model and requirements defined in Section 3.4. Next, we evaluate performance of CACTI in terms of latency and bandwidth. Finally, we discuss CACTI deployability issues.

3.7.1 Security Evaluation

Data integrity & rollback attacks. Since timestamps are stored outside the enclave, a malicious host application can try to modify this data, or roll it back to an earlier version. If successful, this might trick the enclave into producing falsified rate-proofs. However, if any timestamp is modified outside the enclave, this would be detected because the most recent value of the hash chain would not match the corresponding MHT leaf. Assuming a suitable collision-resistant cryptographic hash function, it is infeasible for the malicious host to find alternative hash values matching the MHT root. Similarly, a rollback attack against the MHT is detected by comparing the included counter with the hardware monotonic counter.

Timestamp omission attacks. A malicious application can try to provide the enclave with only a subset of the timestamps for a given request, e.g., to pretend to be below the threshold rate. Specifically, the host could try to omit one or more timestamps at the start, in the middle, and/or at the end, of the range. If timestamps are omitted at the start, the enclave detects this when it checks that the first timestamp supplied by the host is *prior to* the start time of request t_s. If timestamps are omitted in the middle (or at the end) of the range, the most recent hash value will not match the value in the MHT leaf.

List substitution attacks. A malicious client might attempt to use a timestamp hash chain from a different list, or claim that the requested list does not exist. The former is prevented by including list information (list name and public key) in the MHT leaf. If there is a mismatch between the name and the timestamp chain, the resulting leaf would not exist in the MHT. For the latter, when the host calls the enclave's `get_rate` function for a new list, the enclave checks the names of all lists in the MHT to ensure that the new list name does not already exist.

TEE reset attacks. A malicious client might attempt to delete all stored data, including the sealed MHT root, in order to reset the TEE. Since the group private key received from the provisioning authority is sealed together with the MHT root, it is impossible to delete one and not the other. Deleting the group private key would force the TEE to be re-provisioned by the provisioning authority, which may apply its own rate-limiting policies on how often a given client can be re-provisioned.

CACTI **Farms.** Similar to CAPTCHA farms, a multitude of devices with TEE capabilities could be employed to satisfy rate thresholds set by servers. However, this would be infeasible because: *(1)* CACTI enclaves would stop producing rate-proofs after reaching server thresholds and would thus require a TEE reset and CACTI re-provisioning – which is a natural rate limit; *(2)* the cost of purchasing a device would be significantly higher than CAPTCHA solving costs. For example, currently the cheapest service charges \$1.8 for solving $1,000$ reCAPTCHAs [5][3], while a low-end bare-bones CPU with SGX support alone costs $\approx$ \$70 [31], in addition to the maintenance and running costs.

CACTI **Botnets.** An adversary might try to build a CACTI botnet consisting of compromised devices with suitable TEEs in order to bypass CAPTCHAs at scale, similarly to a CACTI farm. However, if the compromised devices are not yet running CACTI, the adversary would have to provision them using a suitable *PA*, which could be made arbitrarily costly and time-consuming. Alternatively, if the compromised devices are already running CACTI, the adversary gains little advantage because the legitimate users will likely have been using CACTI to create their own rate-proofs. Furthermore, the legitimate user would probably

[3]See a comparison of CAPTCHA solving services [57]

notice any overuse/abuse of their system due to quickly exceeding the thresholds.

Client-side malware. A more subtle variant of the reset attack can occur if malware on the client's own system corrupts or deletes TEE data. This is a type of denial-of-service (DoS) attack against the client. However, defending against such DoS attacks is beyond the scope of this work, since this type of malware would have many other avenues for causing DoS, e.g., deleting critical files.

Other DoS attacks. A malicious server might try to mount a DoS attack against an unsuspecting client by inserting a timestamp for a future time. If successful, the client would be unable to insert new timestamps and create rate-proofs for any other servers, since the enclave would reject these timestamps as being in the past. This attack can be mitigated if the client's browser extension and/or host application simply check that the server-provided timestamp is not in the future.

Client tracking. A malicious server (or group of servers) might attempt to track clients by sending multiple requests for rate-proofs with different thresholds in order to learn the precise number of timestamps stored by the client. A successful attack of this type could potentially reduce the client's anonymity set to only those clients with the same rate. However, this attack is easy to detect by monitoring the thresholds sent by the server. A more complicated attack targeting a specific client is to send an excessive number of successful rate-proof requests in order to increase the client's rate. The goal is to reduce the size of the target's anonymity set. This attack is also easy to detect or prevent by simply rate-limiting the number of increments accepted from a particular server. Note that the window of opportunity for this targeted attack is limited to a single session, because malicious servers cannot reliably re-identify the user across multiple sessions (since this is what the attack is trying to achieve). The above attacks cannot be improved even if multiple servers collude.

Rogue *PA*s. A malicious *PA* might try to compromise or diminish client privacy. However, this is prevented by CACTI's use of the EPID protocol [82]. Specifically, due to the BBS+ signature scheme [71] during EPID key issuance, clients' private keys are never revealed to *PA*s. Also, EPID's signature-based revocation mechanism does not require member private keys to be revealed. Instead, signers generate zero-knowledge proofs showing that they are not on the revocation list. Therefore, client privacy does not depend on any *PA* business practices, e.g., log deletion or identifier blinding.

Each website has full discretion to decide which *PA*s it trusts; if a server does not trust the PA who issued the member private key to the TEE, it can simply fall back to CAPTCHAs. This provides no advantage to attackers, and websites can be as conservative as they like. If higher levels of assurance are required, *PA*s can execute within TEEs and provide attestation of correct behavior; we defer the implementation of this optional feature to future work.

Overall, we claim that CACTI meets all security requirements defined in Section 3.4 and significantly increases the adversary's cost to perform DoS attacks. Specifically, the **Unforgeability** requirement is satisfied since it is impossible for the host to perform rollback, timestamp exclusion and list substitution attacks. **Client privacy** is achieved because the rate-proof does not reveal the actual number of timestamps included, and is signed using a group signature scheme.

3.7.2 Latency Evaluation

We conducted all latency experiments on an Intel NUC Kit NUC7PJYH [30] with an Intel Pentium Silver J5005 Processor (4M Cache, up to 2.80 GHz); 4 GB DDR4-2400 1.2V SO-DIMM Memory; running Ubuntu 16.04 with the Linux 4.15.0-76-generic kernel Intel SGX DCAP Linux 1.4 drivers.

Recall that the host application is responsible for initializing the enclave, fetching data necessary for enclave functionality, performing `ECALL`s, and finally updating states according to enclave output. Therefore, we consider the latency in the following four key phases in the host application:

- *Init-Enclave*: Host retrieves the appropriate data from the database and calls `init_mt` `ECALL` that initializes the MHT within the enclave.[4]
- *Pre-Enclave*: Host retrieves the required hashes and timestamps from the database.
- *In-Enclave*: Host calls the `get_rate ECALL`. This phase concludes when the `ECALL` returns.
- *Post-Enclave*: Host updates/inserts the data it received from the enclave into the database.

We investigated the latency impact by varying (1) the number of timestamps in the rate-proof (Section 3.7.2), and (2) the number of lists in the database (Section 3.7.2). We evaluated the end-to-end latency in Section 3.7.2. Unless otherwise specified, each measurement is the average of 10 runs.

Note: The ECDSA and EPID signature operations are, by far, the dominant contributors to latency. However, they represent a fixed latency overhead that does not vary with the number of timestamps or servers. Therefore, for clarity's sake, figures in the following sections do not include these operations. We analyze them separately in Section 3.7.2.

Varying Number of Timestamps in Query

We measured the effect of varying the number of timestamps included in the query, while holding the number of lists constant. As shown in Figure 3.7, query latency increases linearly with the number of timestamps included in the query. The most notable increase is in the in-enclave phase, since this involves calculating a longer hash chain. However, even with

[4]Init-Enclave is done only when the enclave starts.

10,000 timestamps in a query, the total latency only reaches ˜40 milliseconds (excluding signature operations).

Varying Number of Lists

Next, we varied the number of lists while holding the number of timestamps fixed at one per list. We considered two separate scenarios: adding a new list and updating an existing list.

Adding a new list. As shown in Figure 3.8, the latency for the pre-enclave phase is lower compared to Figure 3.7. This is because we optimize the host to skip the expensive `TIMESTAMPS` table look up operation if the host knows that this is a new list. The in-enclave phase increases as the number of lists increases due to the string comparison operations performed by the enclave to prevent list substitution attacks. However, this phase can be optimized by sorting the server names inside the enclave during initial MHT construction. The post-enclave latency is due to the cost of adding entries to the `TIMESTAMPS` table. Figure 3.8 assumes the enclave has already been initialized (see Figure 3.9 for the corresponding init-enclave phase).

Updating an existing list. As shown in Figure 3.9, the latency of the init-enclave phase increases as the number of lists increases. This is expected, since the enclave reconstructs the MHT in this phase. The pre-enclave phase also increases slightly due to the database operations.

Signature Operation Latency

Evaluation results presented thus far have not included the ECDSA signature verification or EPID signature creation operations. Specifically, the server creates an ECDSA signature on the request, which the enclave verifies. The enclave creates an EPID group signature on the

Table 3.1: End-to-End Latency of CACTI for different numbers of timestamps and lists. The *Browser* column represents the latency of the browser extension marshalling data to and from the host application. The other columns are as described above. Pre-, In- and Post- are Enclave phases. Sign and Verify are ECDSA and EPID operations, respectively.

	Sign	**Browser**	**Pre-**	**In-**	**Post-**	**Verify**	**Total**
10,000 timestamps in 1 list	6.3 ms	15.2 ms	7.7 ms	181.7 ms	1.0 ms	27.3 ms	239.2 ms
4,096 lists with 1 timestamp each	6.3 ms	15.2 ms	1.8 ms	157.4 ms	2.0 ms	27.3 ms	210.0 ms

response, which the server verifies using the EPID group public key. The average latencies over 10 measurements for these four signature operations are shown in Figure 3.10. We can see that the EPID group signature generation operation is an order magnitude slower compared to the other cryptographic operations including EPID group signature verification. The latency of our enclave is thus dominated by the EPID signature generation operation.

End-to-End Latency

Table 3.1 shows the end-to-end latency (excluding network communication) from when the server begins generating a request until it has received and verified the response from the client. In both settings, the end-to-end latency is below 250 milliseconds. The latency will be lower if there are fewer lists or included timestamps. Compared to other types of CAPTCHAs, image-based CAPTCHAs take ~10 seconds to solve [84] and behavior-based reCAPTCHA takes ~400 milliseconds, although this might change depending on the client's network latency.

Table 3.2: Additional data received and sent by the client for image-based and behavior-based reCAPTCHA, compared with CACTI.

	Received	**Sent**	**Total**
Image-based	140.05 kB	28.97 kB	169.02 kB
Behavior-based	54.38 kB	26.12 kB	80.50 kB
CACTI	0.82 kB	1.10 kB	1.92 kB

3.7.3 Bandwidth Evaluation

We measured the amount of additional data transferred over the network by different types of CAPTCHA techniques. Minimizing data transfer is critical for both servers and clients. We compared CACTI against image-based and behavior-based reCAPTCHA [47] (see Figure 3.1). The former asks clients (one or more times) to find and mark certain objects in a given image or images, whilst the latter requires clients to click a button. To isolate the data used by reCAPTCHA, we hosted a webpage with the minimal auto-rendering reCAPTCHA example [48]. We visited this webpage and recorded the traffic using the Chrome browser's debugging console.

Table 3.2 shows the additional data received and sent by the client to support each type of CAPTCHA. Image-based reCAPTCHA incurs the highest bandwidth overhead since it has to download images, often multiple times. Although not evaluated here, text-based CAPTCHAs also use images and would thus have a similar bandwidth overhead. Behavior-based reCAPTCHA downloads several client-side scripts. Both types of reCAPTCHA made several additional connections to Google servers. Overall, CACTI achieves at least a 97% reduction in client bandwidth overhead compared to contemporary reCAPTCHA solutions.

3.7.4 Server Load Evaluation

We analyzed the additional load imposed on the server by CACTI. Unfortunately, CAPTCHAs offered as services, such as reCAPTCHA [47] and hCAPTCHA [24], do not disclose their source code and we have no reliable way of estimating their server-side overhead. Therefore, we compared CACTI against two open-source CAPTCHA projects published on GitHub (both have more than 1,000 stars and been forked more than a hundred times):

dchest/captcha [46] (Figure 3.11a) generates image-based text recognition CAPTCHAs consisting of transformed digits with noise in the form of parabolic lines and additional clusters of points. It can also generate audio CAPTCHAs, which are pronunciations of digits with randomized speed and pitch and randomly-generated background noise.

produck/svg-captcha [54] (Figure 3.11b) generates similar image-based text recognition CAPTCHAs, as well as challenge-based CAPTCHAs consisting of simple algebraic operations on random integers. Noise is introduced by varying the text color and adding parabolic lines.

Table 3.3 shows the time to generate different types of CAPTCHAs using the above libraries with typical configuration parameters (e.g., eight characters for text CAPTCHAs). Since CAPTCHA verification with these libraries is a simple string comparison, we assume this is negligible. CACTI's server-side processing is due almost entirely to the EPID signature verification operation. We expect that this time could be improved by using more optimized implementations of this cryptographic operation. Additionally, CACTI uses significantly less communication bandwidth than other approaches, which also reduces the server load (which is not captured in this measurement). Most importantly, the biggest gain of CACTI is on the user side; saving more than ˜10 seconds per CAPTCHA for users.

Table 3.3: Server-side processing time for generating a CAPTCHA and verifying the response.

Library	Type	Time
dchest/captcha	Audio	13.3 ms
	Image-based text	1.7 ms
produck/svg-captcha	Image-based text	2.2 ms
	Image-based math	1.4 ms
CACTI	Rate-proof	33.6 ms

3.7.5 Deployability Analysis

We analyze deployability of CACTI by considering changes required from both the server's and client's perspectives:

Server's perspective. The server will have to make the following changes: (1) create and maintain a new public/private key pair and obtain a certificate for the public key, (2) add an additional `div` to pages for which they wish to enable CACTI, (3) create and sign requests using the private key, and (4) add an HTTP endpoint to receive and verify EPID signatures. The server-side deployment could be further simplified by providing the request generation and signature operations as an integrated library.

Client's perspective. The client will have to make the following changes: (1) download and install the CACTI native software, and (2) download and install the browser extension. Although CACTI requires the client to have a suitable TEE, this is a realistic assumption given the large and increasing deployed base of devices with e.g., ARM TrustZone or Intel SGX TEEs.

3.8 Discussion

3.8.1 *PA* Considerations

As discussed in Section 3.5.3, CACTI's use of a provisioning authority (*PA*) provides the basis for client privacy. CACTI does not prescribe the *PA*'s policies. For example, the *PA* has the choice of running the provisioning protocol (Figure 3.2) as a one-off operation (e.g., when installing CACTI) or on a regular basis, depending on its risk appetite. If there are attacks or exploits threatening the Intel SGX ecosystem (and consequently the security of group private keys), the *PA* can revoke all group member keys. This would force all enclaves in the group to re-register with the *PA*. A similar scenario applies if key-rotation is implemented on the *PA*, e.g., the master secret held by the *PA* is rotated periodically. This forces all enclaves to regularly contact the *PA* to obtain new group member keys. Frequent key-rotation introduces a heavier burden on the clients (although this can be automated), but provides better security.

3.8.2 EPID

Even though CACTI uses EPID group signatures to protect client privacy, CACTI is agnostic to the choice of the underlying signature scheme as long as it provides signer unlinkability and anonymity. We also considered other schemes, such as Direct Anonymous Attestation (DAA) [80], as used in the Trusted Platform Module (TPM). However, DAA is susceptible to various attacks [136, 159, 81] and, due to its design targeting low-end devices, suffers from performance problems. In contrast, EPID is used in current Intel SGX remote attestation and is thus a good fit for enclaves. Moreover, as mentioned in the previous section, the *PA* must revoke group member keys in the event of a compromise. EPID offers privacy-preserving *signature-based revocation*, wherein the issuer can revoke any key using only a signature generated by that key. Signature verifiers use signature revocation lists published by issuers to check whether the group member keys are revoked. Using this mechanism, CACTI provides *PA*s with revocation capabilities without allowing them to link keys to individual users. *PA*s can define their own revocation policies to maximize their reputation and trustworthiness.

3.8.3 Optimizations

Database Optimizations

As with most modern database management systems, SQLite supports creating indexes in database tables to reduce query times. Also, as discussed in Section 3.7, placing all timestamps for all servers in one table and conducting `JOIN` operations incurs performance overhead. An alternative is to use a separate table per list. However, we presented CACTI evaluation results without creating any indexes or separate timestamp tables in order to show the worst-case performance. Performance optimizations, such as changing the database layout, can be easily made by third parties, since they do not affect the security of CACTI.

System-level Optimizations

As a system-level optimization, CACTI can perform some processing steps in the background while waiting for the user to confirm the action. For example, while the browser extension is displaying the notification and waiting for user approval, the request can already be sent to the enclave to begin processing (e.g., loading and verifying the hash chain of timestamps and the MHT). The enclave creates the signed rate-proof but does not release it or update the hash chain until the user approves the action. This optimization reduces user-perceived latency to that of client-side post-enclave and server-side EPID verification processes, which is less than 14% of the end-to-end latency reported in Section 3.7.2.

Optimizing Pruning

Although it is possible to create another `ECALL` for pruning, this might incur additional enclave entry/exit overhead (see Section 3.5.2). Instead, pruning can be implemented within the `get_rate` `ECALL`. Since `get_rate` already updates the hash chain and MHT, the pruning can be performed at the same time, thus eliminating the need for an additional `ECALL` and hash chain and MHT update.

3.8.4 Deploying CACTI

Integration with CDNs and 3rd Party Providers

Although CACTI aims to reduce developer effort by choosing well-known primitives (e.g., SQLite and EPID), we do not expect all server operators to be experienced in implementing CACTI components. The server-side components of CACTI can be provided by Content Delivery Networks (CDNs) or other independent providers.

CDNs are widely used to reduce latency by serving web content to clients on behalf of the server operator. CDNs have already recognized the opportunity to provide abuse prevention services to their customers. For example, Cloudflare offers CAPTCHAs as a free rate-limiting service [16] to its customers [39]. CACTI could easily be adapted for use by CDNs, which would bring usability benefits across all websites served by the CDN.

In addition, independent CACTI providers could offer rate-proof services that are easy to integrate into websites – similar to how CAPTCHAs are currently offered by reCAPTCHA [47] or hCAPTCHA [24]. These services would implement the endpoints described in Section 3.6.4

and could be integrated into websites with minimal effort.

Website Operator Incentives

There are several incentives for website operators to support CACTI. Firstly, in terms of usability, CACTI can drastically improve user experience by allowing legitimate users to avoid having to solve CAPTCHAs. Secondly, in terms of privacy, some concerns have been raised about existing CAPTCHA services [39]. By design, CACTI rate-proofs cannot be linked to specific users or to other rate-proofs created by the same user. Thirdly, in terms of bandwidth usage, CACTI requires an order of magnitude less data transfer than other CAPTCHA systems.

User demand for privacy-preserving solutions that reduce the amount of time spent solving CAPTCHAs has led Cloudflare to offer *Privacy Pass* [96], a system designed to reduce the number of CAPTCHAs presented to legitimate users, especially while using VPNs or anonymity networks [59].

PA Operator Incentives

In CACTI, *PA*s are only involved when provisioning credentials to CACTI enclaves (i.e., not when the client produces a rate-proof). This is a relatively lightweight workload from a computational perspective. *PA*s could be run by various different organizations with different incentives, for example:

1. TEE hardware vendors wanting to increase the desirability of their hardware;
2. Online identity providers (e.g., Google, Facebook, Microsoft) who already provide federated login services;
3. For-profit businesses that charge fees and provide e.g., a higher level of assurance;

4. Non-profit organizations, similarly to the Let's Encrypt Certificate Authority service.

CACTI users can, and are encouraged to, register with multiple *PA*s and randomly select which private key to use for generating each rate-proof. This allows new *PA*s to join the CACTI ecosystem and ensures that clients have maximum choice of *PA* without the risk of vendor lock-in.

Client-side components

On the client-side, CACTI could be integrated into web browsers, and would thus work "out of the box" on platforms with a suitable TEE.

3.9 Related Work

Related work for rate limiting lies in the intersection of multiple fields of research, including DoS (or Distributed DoS (DDoS)) protection, human presence, and CAPTCHA improvements and alternatives. In this section, we discuss related work in each of these fields.

Network layer defenses. The main purpose of network layer DoS/DDoS protection mechanisms is to detect malicious network flows targeting the availability of the system. This is done by using filtering [139] or rate-limiting [90] (or a combination thereof) according to certain characteristics of a flow. We refer the reader to [155] for an in-depth survey of network-level defenses. Moreover, additional countermeasures can be employed depending on the properties of the system under attack (e.g., sensor-based networks [151], peer-to-peer networks [156] and virtual ad-hoc networks [132]).

Application layer defenses. Application layer measures for DoS/DDoS protection focus on separating human-originated traffic from bot-originated traffic. To this end, problems that are hard to solve by computers and (somewhat) easy to solve by humans comprise the basis of application layer solutions. Although developing more efficient CAPTCHAs is an active area of research [123, 171, 161, 95], research aiming to subvert CAPTCHAs is also prevalent [145, 174, 121, 119]. In addition to such automated attacks, CAPTCHAs suffer from inconsistency when solved by humans (e.g., perfect agreement when solved by three humans are 71% and 31% for image and audio CAPTCHAs, respectively [84]). [146] suggests that although CAPTCHAs succeed at telling humans and computers apart, by using CAPTCHA-solving services (operated by humans), with an acceptable cost, CAPTCHAs can be defeated. Moreover, apart from questions regarding their efficacy, one other concern about CAPTCHAs is their usability. Studies such as [117, 84] show that CAPTCHAs are not only difficult but also time-consuming for humans, with completion time of ≈10 seconds on average. While behavioral CAPTCHAs are available, they suffer from privacy issues. A prevalent example, reCAPTCHA [47], works by analyzing user behavioral data (which requires sharing this data with the CAPTCHA provider) and claims to work more efficiently if used on multiple pages.

Human presence detection. Human presence refers to determining whether specific actions were performed by a human. VButton [138] proposes a system design based on ARM's TrustZone [66]. Secure detection of human presence is achieved by setting the display and the touch input peripherals as secure peripherals which can only be controlled by the TEE while VButton UI is displayed. With a secure I/O mechanism in place, user actions can be authenticated to originate from VButton UI by a remote server using software attestation. Similarly, Not-a-Bot [124] designs a system based on TPMs by tagging each network request with an attestation assuring that the request has been performed not long after a keyboard or mouse input by the user. Unfortunately, Intel SGX does not support secure I/O and it is not currently possible to implement similar systems on devices with only Intel SGX support. SGXIO [172] proposes an architecture for creating secure paths to I/O devices from enclaves using a trusted stack which contains a hypervisor, I/O drivers and an enclave for trusted boot. In addition, an untrusted VM hosts secure applications. The communication between secure applications and drivers are encrypted using keys generated at the end of the local attestation process. Unfortunately, the implementation of this system is not yet available. Fidelius [104] protects user secrets from a compromised browser or OS by protecting the path from the input and output peripherals to the hardware enclave. Similar to SGXIO, this is a promising step towards general-purpose trusted UI.

Privacy Pass. Privacy Pass [96] implements a browser extension to reduce the burden of CAPTCHAs for legitimate users when visiting websites served by Cloudflare. When a user solves a CAPTCHA, Cloudflare sends the user multiple anonymous cryptographic tokens, which the user can later "spend" to access Cloudflare-operated services without encountering additional CAPTCHAs Although Privacy Pass significantly benefits benign users, it could still be exploited by CAPTCHA farms. Additionally, Privacy Pass' is currently limited to Cloudflare users.

3.10 Conclusion

This chapter discussed CACTI, a novel approach for leveraging client-side TEEs to help legitimate clients avoid solving CAPTCHAs. The unforgeable yet privacy-preserving rate-proofs generated by the TEE provide strong assurance that the client is not behaving abusively. Our proof-of-concept implementation demonstrates that rate-proofs can be generated in less than 0.25 seconds on commodity hardware, and that CACTI reduces data transfer by more than 98% compared to existing CAPTCHA schemes.

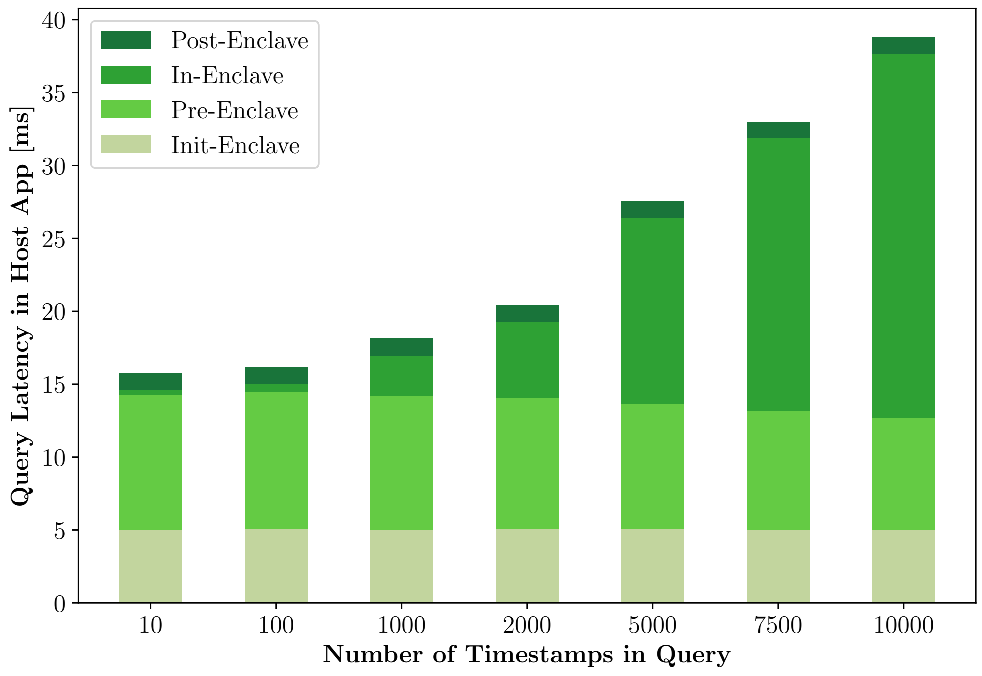


Figure 3.7: Latency of initializing the enclave and creating a rate-proof for different numbers of timestamps in the query (excluding signature operations).

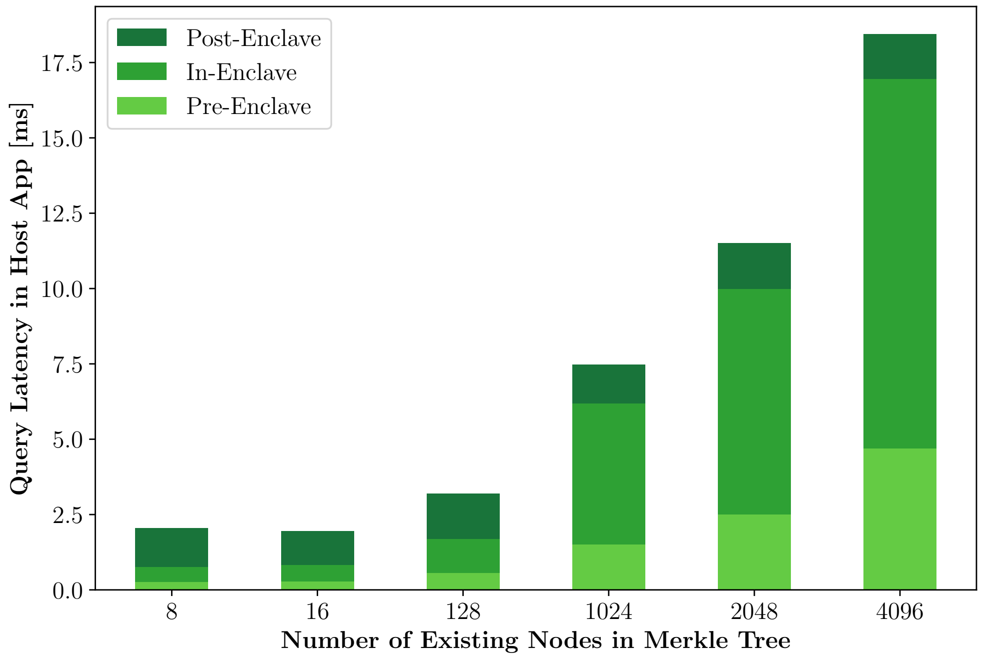


Figure 3.8: Latency of creating the first rate-proof in a new list for different numbers of existing lists (excluding enclave initialization and signature operations).

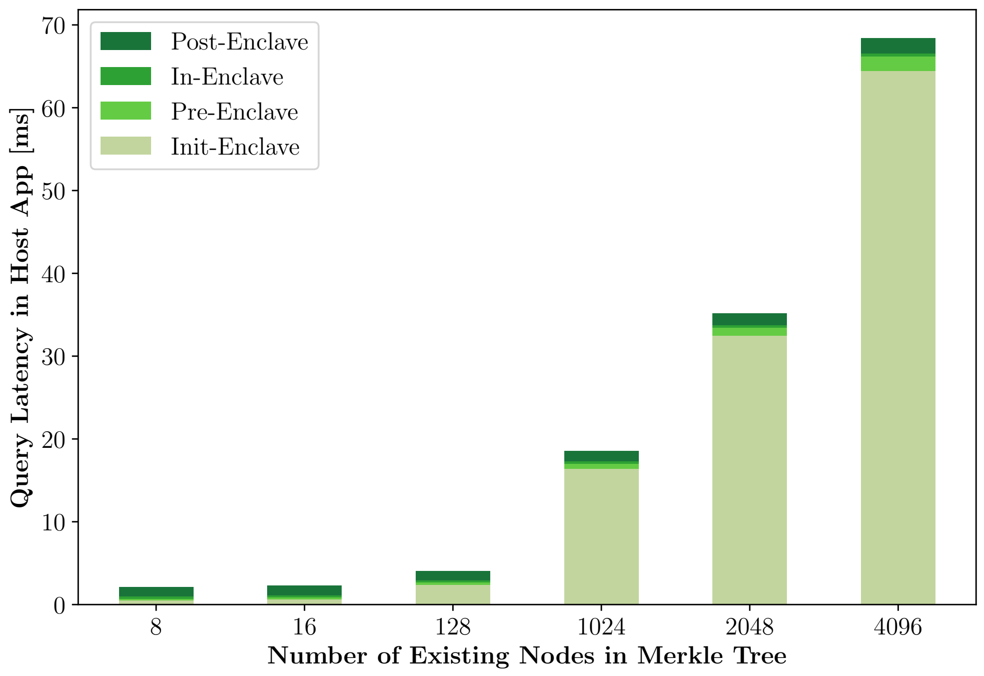


Figure 3.9: Latency of initializing the enclave and updating an existing list for different numbers of existing lists (excluding signature operations).

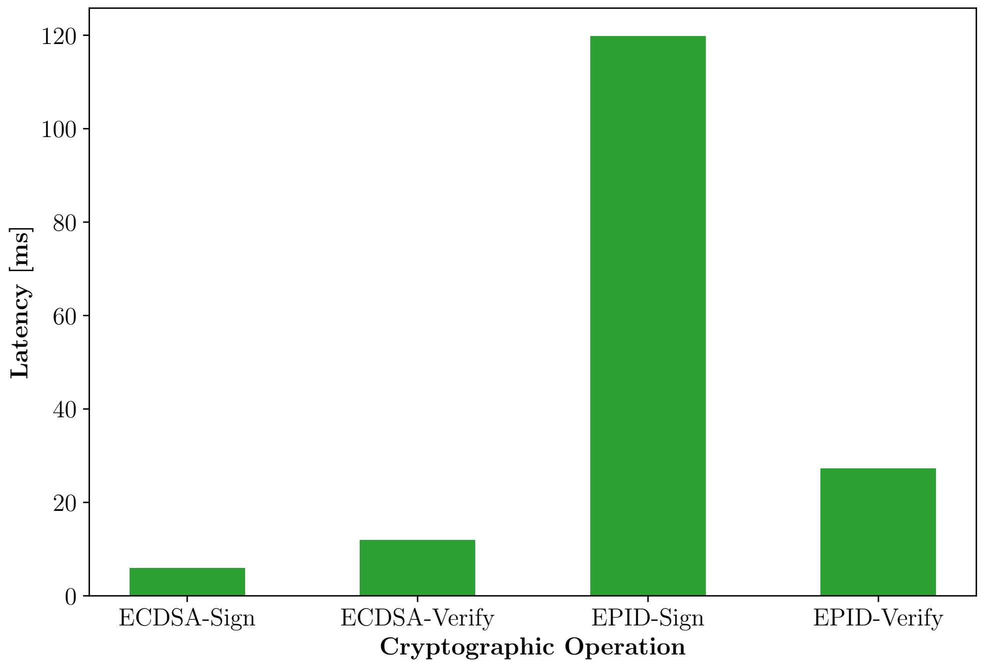
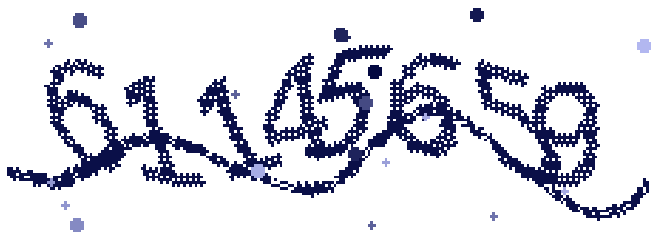


Figure 3.10: Microbenchmarks of signature operations. ECDSA signatures were created and verified using the `mbed` TLS library [37] and EPID signatures with the Intel EPID SDK [18].


(a) dchest/captcha image-based CAPTCHA [46].

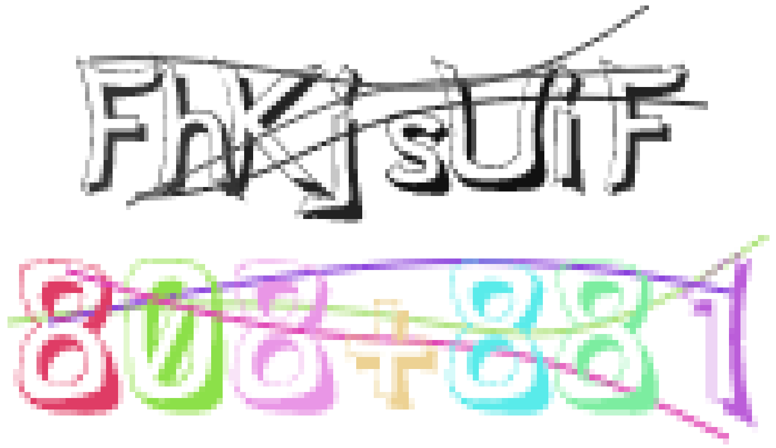

(b) produck/svg-captcha image-based CAPTCHAs [54].

Figure 3.11: CAPTCHAs generated using open-source libraries.

Chapter 4

GDPR-/CCPA-compliant Verifiable Accountless Consumer Requests

4.1 Introduction

Isolated code execution offered by TEEs prevents other software on the system from observing or interfering with the execution state of a program. Using remote attestation, one can verify that the TEE (for ensuring TEE presence See [177]) is running the expected code and create a secure channel to it. This channel can be used to provision various secrets to the attested code. In this chapter we present the design of a *proofs of data ownership* framework based on a minimal secret (a private key) – that can be kept in a TEE using the flow above.

Recent data protection regulations (such as GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act)) grant consumers various rights, including the right to *access*, *modify* or *delete* any personal information collected about them (and retained) by a service provider. To exercise these rights, one must submit a *verifiable consumer request* proving that collected data indeed pertains to them. This action is relatively straightforward for consumers with active accounts with a service provider at the time of data collection, since they can use standard (e.g., password-based) means of authentication to validate their requests. However, a major conundrum arises from the need to support consumers *without accounts* to exercise their rights. To this end, some service providers began requiring these *accountless* consumers to reveal and prove their identities (e.g., using government-issued documents, utility bills or credit card numbers) as part of issuing a verifiable consumer request. While understandable as a short-term cure, this approach is, at the same time, cumbersome and expensive for service providers as well as very privacy-invasive for consumers.

Consequently, there is a strong need to provide better means of authenticating requests from accountless consumers. To achieve this, we propose `VICEROY`, a privacy-preserving and scalable framework for producing *proofs of data ownership*, which can be used as a basis for verifiable consumer requests. Building upon existing web techniques and features (e.g., cookies), `VICEROY` allows accountless consumers to interact with service providers, and later prove – in a privacy-preserving manner – that they are the same person, with minimal requirements for both parties. We design and implement `VICEROY` with the emphasis on security/privacy, deployability and usability. We also thoroughly assess its practicality via extensive experiments.

4.2 Motivation & Contributions

Several new data protection regulations have been enacted in recent years, notably, the European Union's General Data Protection Regulation (GDPR) [153] and the California Consumer Privacy Act (CCPA) [135]. These regulations grant various new legal rights to consumers[1]. For example, consumers gain the rights to *access* personal data collected about

[1]We use the term *consumer* to refer to: (1) the GDPR term *data subject*, (2) the CCPA term *consumer*, and (3) the equivalent terms in other regulations.

them and held by service providers (GDPR Art. 15, CCPA Sec. 1798.100), *request correction* (GDPR Art. 16, CCPA Sec. 1798.106) or *request deletion* of their personal data (GDPR Art. 17, CCPA Sec. 1798.105).

Importantly, these regulations expand the definition of *personal data* beyond that associated with a person's real name. For example, GDPR Rec. 30 states that natural persons "*may be associated with online identifiers provided by their devices, applications, tools and protocols, such as internet protocol addresses, cookie identifiers or other identifiers*". In the Web context, this means that any website[2] collecting information about consumers based on identifiers such as IP addresses or cookies, may be collecting personal information, and thus have to comply with these new regulations. In particular, a website must provide a means by which consumers can access, request correction of, or request deletion of their personal information.

When dealing with a request from a consumer, a website must verify that the requestor is indeed the consumer to whom the personal information pertains. This is critical to prevent erroneous disclosure (which would be a serious violation of any data protection regulation), unauthorized modification, or deletion of personal information. In CCPA terminology, this is called a "*verifiable consumer request*" (CCPA 1798.140(y)).[3]

For a consumer who has a pre-existing account on a given website, submitting a verifiable consumer request ($\mathcal{VCR}$) is relatively straight-forward. To wit, CCPA Sec. 1798.185 requires: "*treating a request submitted through a password-protected account maintained by the consumer ... as a verifiable consumer request*". However, there remains a major challenge of how to support a $\mathcal{VCR}$ from casual or **accountless** consumers, as required per CCPA Sec. 1798.185, while protecting such consumers' privacy.

[2]As shorthand, we use the term *website* to represent the entity operating a website, which (we assume) falls into the category of entities that the GDPR and CCPA call *controller* and *business*, respectively.

[3]These requests are sometimes also referred to as Subject Rights Request (SRR) or Subject Access Request (SAR).

The only mechanisms used in practice that are suitable for accountless consumers are those that require: a device cookie, a government-issued ID, a signed (and possibly witnessed) statement, a utility bill, a credit card number, or taking part in a phone interview [154]. However, not only are these mechanisms cumbersome for consumers (thus deterring them from exercising their rights), but insecure, as demonstrated by prior work [154, 86, 100, 77]. This is because these methods typically require manual processing, making them error-prone and costly for businesses. Moreover, such methods (apart from device cookies) are privacy-invasive for the consumer and potentially open the door for further consumer data exposure. For example, a government-issued ID or utility bill reveals even more private information to the business operating the website – and in case of a breach, to the world.

Reflecting such issues, CCPA's preferred verification approach is for the business to match information provided by the consumer (as part of a $\mathcal{VCR}$) to the personal information about the same consumer already collected by the business. The issue with this approach is that the information used for verification may not be personal enough or just too vague for a consumer to retain, as the $\mathcal{VCR}$ may be made months or even years later.

Therefore, from the aforementioned methods, device cookies seem to be a more natural fit for the Web context. Cookies are already used pervasively to identify the consumer to the website, and to link multiple sets of activities (sessions) by the same consumer. At first glance, asking for device cookies as part of a $\mathcal{VCR}$ appears to meet the main requirements of the GDPR and the CCPA: only the authorized consumer should be in possession of the correct cookie (*unforgeability requirement*), and providing a cookie does not reveal additional information to the business (*privacy requirement*). However, relying on device cookies as *authentication tokens* for $\mathcal{VCR}$s has (at least) two disadvantages:

First, cookies are used (i.e., sent over the network) whenever the consumer interacts with the website. This immediately means that the adversary obtains any cookies sent over unencrypted connections (at any point in the cookie's lifespan). Although secure communication channels (e.g., TLS connections) can certainly help to mitigate this, any vulnerabilities in the browser or website could expose the cookies.

Second, consumers must protect the cookies at rest on their devices, especially in the face of client-side spyware or an adversary who temporarily gains access to their device. Secure storage may become more challenging as the number of stored cookies increases (since consumers should not delete cookies in case they subsequently need to make a $\mathcal{VCR}$).

The MITRE ATT&CK framework lists several recent examples of techniques for stealing web cookies [38]. Irrespective of how the adversary obtains the cookie, the consequences would be equally severe, allowing the adversary to request, modify, or delete all the consumer's data.

Motivated by aforementioned issues, we construct `VICEROY`, a framework that allows *accountless* consumers to request their data in a private manner, while allowing website operators to efficiently and securely verify these requests.

`VICEROY` is based on one-to-one mapping of Web sessions with consumer-generated public keys. More specifically, at session initiation, consumers generate a public key (a $\mathcal{VCR}$ key) and supply it to the Web server. At a later time, to exercise their rights on data collected during a session, consumers digitally sign their request using the private key corresponding to the $\mathcal{VCR}$ key for the session.

To ensure consumer privacy, our key derivation mechanism uses unlinkable public keys derived from a single master public key and only requires secure storage of a small secret (a master private key), regardless of the number of consumer devices. This private key is only accessed when generating $\mathcal{VCR}$s and can be protected using existing secure key storage mechanisms, such as hardware-based key stores. `VICEROY` works with Web sessions out-of-the-box and servers only need minimal changes in order to adopt it.

We view the contributions of this work as being two-fold:

- The design of `VICEROY`– a secure, scalable, and privacy-preserving $\mathcal{VCR}$ mechanism for accountless consumers.
- A proof-of-concept implementation of `VICEROY` for web browsers, and a thorough evaluation of its security, performance, and deployability.

Organization. The remainder of the paper is structured as follows: Section 4.3 presents background on the consumer rights granted by the GDPR/CCPA as well as the concept of verifiable consumer requests ($\mathcal{VCR}$s). Section 4.4 sets out our system and threat models and defines the requirements for our system. Sections 4.5 and 4.6 respectively describe our design and proof-of-concept implementation of `VICEROY`, and Section 4.7 presents our evaluation methodology and results. We discuss various additional aspects of `VICEROY` in Section 4.8. Section 4.9 discusses related work. Section 4.10 concludes the chapter and lists future work.

Code Availability. Anonymized source code for all `VICEROY` components is available at [4].

4.3 GDPR/CCPA Background

This section overviews some preliminaries, including the scope of Personally Identifiable Information and consumer rights under the GDPR and the CCPA.

4.3.1 Personally Identifiable Information (PII)

Both the GDPR and CCPA pertain to information that is both personal and personally identifiable. The combination is often called *Personally Identifiable Information*[4] or PII.

[4]The GDPR uses the term *personal data* and the CCPA uses *personal information.*

Information is *personal* if it relates to a person or household. It may include: contact information, geolocation, applications and devices used, how an application is used, interests, websites visited, consumer-generated content, identities of people with whom a consumer communicated, content of communications, audio, video, and sensor data [130].

Information is *personally identifiable* if the person or household to which it relates is either identified or identifiable. If information is paired with a name, telephone number (landline or cell), email address, government-issued identifier, or home postal address, then it is considered to be personally identifiable [129]. If information is paired with an IP address, a device identifier (e.g., an IMEI), or an advertising identifier, it is **likely** to be considered as personally identifiable [129]. Information paired with an identifier created by a business (e.g., a cookie) is personally identifiable if it can be combined with other information to allow the consumer to whom it relates to be identified [129].

4.3.2 Rights of Access and Erasure

Both the GDPR and the CCPA require a business that collects PII to disclose, typically in its privacy policy, the categories of PII collected, the purposes for collecting it, and the categories of entities with whom that PII is shared [129].

- Both the GDPR and the CCPA give consumers the right to learn about and control the information relating to them that a business has collected. Specifically, consumers have the right to request access to the *specific pieces of PII* that the business has collected (GDPR Art. 15; CCPA Sec. 1798.110(a)(5)).
- Both the GDPR and the CCPA give consumers the right to request that their incorrect PII be corrected (GDPR Art. 16; CCPA Sec. 1798.106).
- Finally, both the GDPR and the CCPA give consumers the right to request that a business delete their PII (GDPR Art. 17; CCPA Sec. 1798.105).

4.3.3 Verifiable Consumer Requests ($\mathcal{VCR}$s)

The consumer's rights to access, and request correction or deletion of, their PII are contingent upon verification that the consumer is indeed the person to whom that PII relates. However, the GDPR and CCPA differ in the requirements of methods of verification. Both regulations require a business to use reasonable measures to verify the consumer's identity (GDPR Rec. 64; CCPA Sec. 1798.140(ak)). If a consumer has a password-protected account with a business, both require a business to treat requests submitted via that account as verified (GDPR Rec. 57; CCPA Sec. 1798.185(a)(7)).

However, both regulations also recognize that PII is often collected about casual consumers, who do not have password-protected accounts. In this case, they envision a consumer request being verified by associating additional consumer-supplied information with PII that the business previously collected about that consumer (CCPA Sec. 1798.130(a)(3)(B)(i)). The CCPA further specifies that any information provided by the consumer in the request can be used solely for the purposes of verification (CCPA Sec. 1798.130(a)(7)). However, if a business has not linked PII to a consumer or a household, and cannot link it without the acquisition of additional information, then neither the GDPR nor the CCPA require a business to acquire additional information to verify a consumer request (GDPR Rec. 57; CCPA Sec. 1798.145(j)(3)). Thus, some requests may be **unverifiable.**

The CCPA [120] recognizes that consumer verification is not absolutely certain. It establishes two thresholds of certainty. The lower threshold, called *reasonable degree of certainty*, may be satisfied by matching at least two pieces of information provided by the consumer (CCPA Regs. §999.325(b)). The higher threshold, called *reasonably high degree of certainty*, may be satisfied by matching at least three pieces of information provided by the consumer, and obtaining a signed declaration from the consumer (CCPA Regs. §999.325(c)). However, other means of verification may also satisfy these thresholds. Verification of the consumer identity must always, at a minimum, meet the *reasonable degree of certainty* threshold (CCPA Regs. §999.325(b,d)). Furthermore, requests to learn specific pieces of PII must meet the *reasonably high degree of certainty threshold* (CCPA Regs. §999.325(c)). Finally, a consumer may instead use a third-party verification service (CCPA Regs. §999.326).

The design of a verification method should balance the administrative burden on the consumer (CCPA Sec. 1798.185(a)(7)) with the likelihood of unauthorized access and the risk of harm (CCPA Regs. §999.323(b)(3)).

4.4 Threat Model and Requirements

Our system model assumes a typical Web environment with two types of principals: *(1)* clients and *(2)* servers. Clients are *consumers* who access Internet services offered by servers. Each client can own multiple devices and at least one of the client's devices can be trusted to store a secret, e.g., a private key. This trusted device could be a dedicated key storage device, such as a YubiKey or other secure hardware wallet. All access to the secret is controlled by the client. Physical and side-channel attacks against the trusted client device, although possible, are beyond the scope of this chapter. We assume secure communication channels between clients and servers, which can be realized using e.g., TLS. In Section 4.7.1, we also evaluate the consequences if this assumption does not hold.

We consider the following three types of adversarial behavior:

- **Malicious clients:** Attempt to perform operations on data that are not theirs.
- **Client-side malware:** Attempts to perform unauthorized operations on client data. We assume that the client's trusted device is free of malware, but all other client devices can potentially be infected.
- **Honest-but-curious (HbC) servers:** Attempt to identify clients who submit re-

quests, or to link multiple requests by the same client. Moreover, servers may decline to respond to valid $\mathcal{VCR}$s. Multiple servers might collude to link client requests and/or to learn the client's identity.

As usual, we assume all relevant cryptographic primitives are implemented and used correctly and cannot be attacked via side-channels or any other weaknesses. Similarly, we assume digital signatures can only be generated by the true owner of the private key.

Based on the above system model, we define the following requirements for `VICEROY`:

- **Unforgeability:** Only the client who originally interacted with the server can create a $\mathcal{VCR}$.
- **Consumer Privacy:** An honest-but-curious server (or a set of colluding servers) should be unable to link a request to a specific client, or to link multiple requests to the same client, unless that was already known to the server.

4.5 `VICEROY` Design & Challenges

This section discusses `VICEROY`'s goals, design features, and challenges encountered. Note that we use the terms **client** and **consumer** interchangeably.

4.5.1 Design Motivation

One straight-forward mechanism to support $\mathcal{VCR}$s from account-less consumers is to require them to provide the same cookie(s) they were issued when visiting the server website. Indeed this is one of the mechanisms that [154] encountered in their survey of how businesses respond to access requests. The rationale is that only the consumer from whom the data was collected should have access to the cookie, since the latter links consumer's activity that constitutes a session.

Such use of cookies has several advantages: First, it is *privacy-preserving* in that, when making a request, the consumer does not reveal any further personal information that the server didn't already have. Furthermore, if the consumer submits multiple requests based on different cookies, the server cannot link them.[5] Second, this mechanism is easily deployable, since cookies are supported by virtually every device that uses the Web.

However, as introduced in Section 4.2, there are also several significant disadvantages: First, this method essentially makes cookies into *symmetric* authentication tokens – anyone with a cookie can create $\mathcal{VCR}$s. This is problematic because the cookies are used in all interactions with the website, and several techniques for stealing such cookies have been demonstrated [38]. Second, since consumers visit many different websites, they would have to *securely* and *reliably* store a potentially large number of cookies. This differs from the usual client-side cookie management, since cookies would be additionally valuable as a means to issue $\mathcal{VCR}$s. Also, if cookies are lost (e.g., due to disk failure), the consumer would be unable to exercise their GDPR/CCPA rights. This underscores the importance of cookie storage reliability. Furthermore, if the server for any reason also stores copies of the cookies, the same security requirements apply. If these cookies were leaked in a server data breach, the server would have to invalidate them in bulk, thus preventing legitimate consumers from submitting $\mathcal{VCR}$s, or risk the attackers requesting consumers' personal data.

4.5.2 Conceptual Design

Motivated by the above challenges, we construct `VICEROY` to avoid the aforementioned drawbacks. We now describe some key features of `VICEROY`.

Asymmetric tokens: One key feature of `VICEROY` is the use of *asymmetric* authentication

[5]Potential "fingerprinting" of the consumer's browser or network interface notwithstanding.

tokens, inspired by techniques such as Origin-Bound Certificates [101] and token binding [56]. When interacting with a server, a client provides the public part of an asymmetric key-pair – the $\mathcal{VCR}$ public key. The client fully controls its privacy as it can choose to use the same public key, or generate a new one, when interacting with different servers, or engaging in multiple sessions with the same server. (A privacy-minded client would generate and use a unique key-pair for each session.)

The server associates the $\mathcal{VCR}$ public key with a particular *session*. Generally, a session is any linkable set of interactions between a client[6] and the server. Since the notion of a session can vary widely between different types of servers, each server can define its own notion, thus providing flexibility. For example, in the Web context, a session most likely corresponds to an `HTTP(S)` session – managed using cookies since `HTTP` is stateless. To submit a $\mathcal{VCR}$ for a session, the client signs the request using the corresponding $\mathcal{VCR}$ private key.

This approach addresses the first of the drawbacks of using cookies, since the private $\mathcal{VCR}$ key is never sent over the network. The use of digital signatures also allows additional information/parameters to be cryptographically bound to the request e.g., a request to correct personal information could, in some cases, already include the corrected information. However, attempting to replace existing Web cookies with client-generated public $\mathcal{VCR}$ keys is infeasible from the deployment perspective, as it would require non-trivial modifications to the way servers use cookies. Although many websites use cookies simply as identifiers to track users, others utilize the full potential of cookies to store client state information. For example, in a large-scale analysis of cookies in the wild, Gonzalez et al. [122] found cookie values containing URLs, email addresses, timestamps, and even JSON objects. To avoid changing the existing and ubiquitous cookie mechanism, we instead propose the use of a *cookie wrapper* – a cryptographic binding between an existing server-generated session identifier (such as a traditional cookie) and a client-generated public key.

[6]In Section 4.8 we show how to extend this to include devices operated by multiple clients – the so-called *"roommate problem"*.

Cookie wrappers: The use of cookie wrappers is the second key feature of `VICEROY`. It significantly improves deployability by allowing servers to add support for `VICEROY` without modifying existing cookie management. Specifically, when a client connects to a server, the server generates a session identifier, as usual. Independently, the client generates an asymmetric $\mathcal{VCR}$ key-pair and sends the public key to the server, as mentioned above. To bind these two items, the server generates a wrapper for the cookie, i.e., signs its cookie along with the client-generated $\mathcal{VCR}$ public key, using its long-term signing key. The server then sends this wrapper to the client for safe-keeping. To issue a $\mathcal{VCR}$, the client signs the request message (which includes the wrapper) using the $\mathcal{VCR}$ private key, and sends this to the server. From the server's perspective, the client's valid signature on the request message (including the wrapper, which, itself, includes the $\mathcal{VCR}$ public key) demonstrates that the client's $\mathcal{VCR}$ public key is indeed authorized to sign this request, i.e., makes the request verifiable.

4.5.3 Design Challenges

We now discuss some challenges in implementing `VICEROY`.

Key Explosion: `VICEROY` avoids using a static public key per client, for privacy reasons, to avoid linkability. A static public key would allow a server (or a set of colluding servers) to link multiple sessions involving the same client. This linkage could take place at session initiation time, i.e., when the client requests a wrapper, or when the client issues a $\mathcal{VCR}$. Also, if a client's static public key is leaked, it becomes possible to track that client's sessions globally. However, requiring each client to have a distinct public/private key-pair per session can cause a "key explosion" in which the client has to securely store many private keys.

Secure Key Management: $\mathcal{VCR}$ private keys for each session must be stored in a secure environment, access controlled by the client. Leakage of these private keys would result in easy impersonation of the client in terms of issuing $\mathcal{VCR}$s, and subsequent ability to: learn, modify, or delete, potentially sensitive session information.

Long-Term Storage: A $\mathcal{VCR}$ may be submitted long (possibly, years) after the corresponding session ends. This requires state information for a potentially numerous sessions (for each client) to be stored in a secure and highly available manner. Naturally, the amount of state per session must be minimal.

Multiple Devices: Since $\mathcal{VCR}$s are triggered by human action, seamlessness and ease-of-use are important. In particular, it should be possible for a $\mathcal{VCR}$ to originate from any of the client's devices, albeit, the trusted device (which stores private keys) is still needed to generate signed (verifiable) requests.

Broad Application Support: Many smartphone applications involve communication with application-specific servers, which, similar to Web servers, collect consumer data. Such data should also be accessible and manageable using VICEROY.

4.5.4 Realizing VICEROY Design

Based on the challenges raised in Section 4.5.3, we explain the key design choices in realizing VICEROY.

Key generation

Servers often use client identifier cookies to gather analytics or to support other features, such as shopping carts. Since the server can already link together all requests that use the same cookie (i.e., a *session*), we do not need to generate $\mathcal{VCR}$ keys for each individual request. Therefore, we choose to use per-session $\mathcal{VCR}$ keys.

To generate such keys, we use the concept of *derivable asymmetric keys* (e.g., the key derivation scheme used in Bitcoin Improvement Proposal (BIP) 32 [7]). Conceptually, this type of key derivation scheme allows a chain of *child public keys* to be derived from a single *parent public key*. Importantly, the derivation of public keys does not require access to the corresponding parent private key. Furthermore, the corresponding child private keys can only be derived from the parent/master private key. We denote the *derivation path* of a key in the form `a/b/c/...`, where e.g., `a/b` is the b^{th} child key of `a`, and `a/b/c` is the c^{th} child key of `a/b`.

This approach addresses two issues: **First**, it minimizes the public key storage requirements of `VICEROY` – only the parent public key must be stored, whilst all other public keys can be derived. When a new session is initiated, the parent public key, which we refer to as the *device public key*, is used to generate a new child public key. This approach resolves the key explosion issue. **Second**, using derivable public keys does not require the presence of the private key for generating new child public keys. This is important for the security of `VICEROY`, since the master private key can remain in secure storage on the client's trusted device, as explained in Section 4.5.4, thus resolving the secure key management issue.

Cookie and wrapper storage

Since clients may use multiple devices and change devices over time, it should be possible for them to issue a $\mathcal{VCR}$ from any of their devices. To support this, we propose to use a synced storage with high availability (e.g. the same type of mechanism used to synchronize bookmarks and other browser settings between the client's devices). (In Section 4.6, we take a look into further details of an example of such storage.) One of the main benefits of our design is that neither the cookies themselves nor the wrappers can be used to issue $\mathcal{VCR}$s, without a signature from the client's private key. This means that the security requirements for the storage of cookies and wrappers are minimal, and can this be safely outsourced to third-party providers.

Private key storage

Preventing unauthorized access to the master private key is critical for achieving the security requirements of VICEROY. Fortunately, our use of derivable asymmetric keys means that the master private key is not used frequently – it is only needed to generate new device public keys (e.g., when enrolling a new device) or to create $\mathcal{VCR}$s (which is expected to be a relatively infrequent operation). The only requirements are that the trusted device holding the master private key must be able to derive child keys (as described in Section 4.5.4) and create signatures.

By design, VICEROY gives clients significant flexibility in terms of how their private keys are stored, in order to accommodate different levels of security. For example, at one end of the spectrum, clients with low security requirements can simply store their private keys on any device they trust (e.g., a phone or laptop). Clients with higher security requirements could store their keys in hardware-backed keystores, such as the Android Keystore [3] or Apple Secure Enclave [53], which are strongly tied to specific devices. On PCs, clients could make use of hardware-enforced enclaves, such as Intel SGX [33], or technologies like Windows Virtualization-base security (VBS) to protect the keys. At the top end of the spectrum, clients with the highest security requirements could store their keys in secure hardware digital wallets, such as those manufactured by YubiKey or Ledger [35]. These clients may also enforce additional physical security controls, such as keeping the hardware wallet in a locked safe until it is needed. Clients may also make additional back-up copies of their master private keys to allow recovery if the trusted device is lost/stolen or fails.

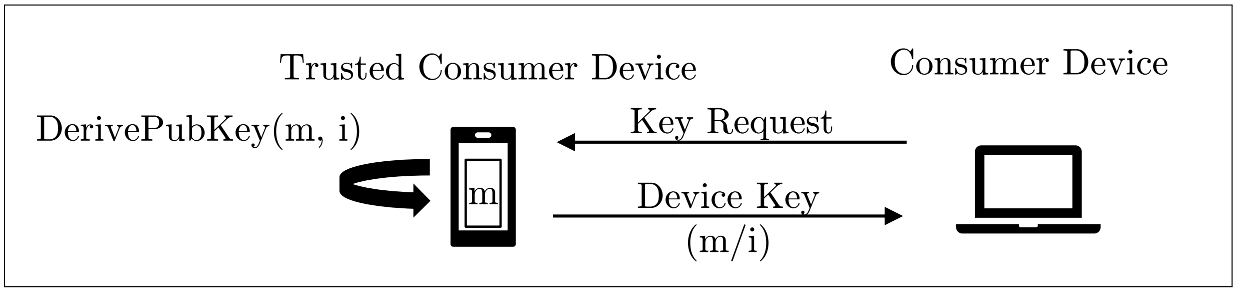


Figure 4.1: `VICEROY` Device Setup. `m` is the master private key, `i` is a device ID number. `m/i` represents the key derivation path.

Device provisioning

Consumers often own and use multiple devices. Since a $\mathcal{VCR}$ can emanate from any device, it is important to support secure device provisioning. Device provisioning in `VICEROY` consists of the following three phases:

Device Setup: A device is provisioned with a device-specific public key generated from the master private key residing in secure storage on a trusted consumer device (as shown in Figure 4.1).

Key Generation: When initiating a session with a server, a client generates a new $\mathcal{VCR}$ key and sends it, along with the client id cookie information, to the server (See Figure 4.2). After receiving the $\mathcal{VCR}$ key, the server returns a wrapper attesting to the association between the client id cookie and the $\mathcal{VCR}$ key. Note that we do not assume a global PKI, so clients are free to generate keys as they wish. This also means that clients do not need to prove ownership of the corresponding private keys, since these keys are not bound to any real-world identities.

$\mathcal{VCR}$ **Issuance:** A client generates a $\mathcal{VCR}$, which optionally includes additional metadata. For example, a client requesting data can include a public key in a $\mathcal{VCR}$ that the server can use it to encrypt the returned data. (More specifically, the server would public-key-encrypt a fresh symmetric key which, in turn, would be used for bulk encryption of the data).

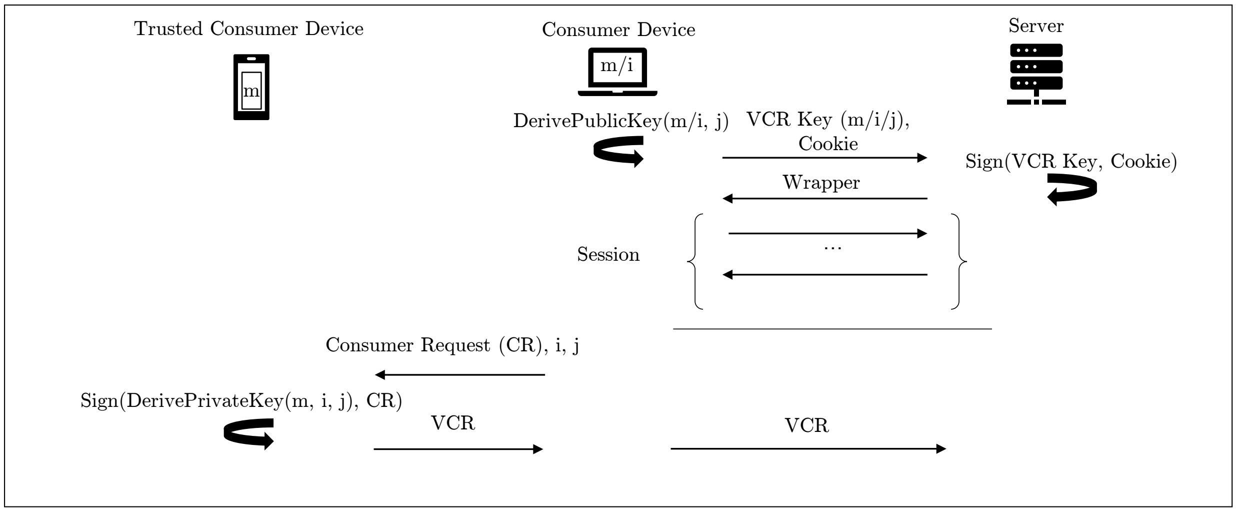


Figure 4.2: `VICEROY` Key Generation and $\mathcal{VCR}$ Issuance Flows. `j` is a session counter per-device.

The request and metadata are signed with the appropriate $\mathcal{VCR}$ private key, as shown in Figure 4.2. This step requires client's consent to access secure storage on the trusted device. Once received a $\mathcal{VCR}$, the server verifies authenticity of the wrapper (i.e., verifies its own signature on it) and verifies the client's signature of the $\mathcal{VCR}$ using the public key from the wrapper. If all these checks succeed, the server proceeds with the requested data operation.

Device Unprovisioning

Finally, it is important to consider the full lifecycle of a device, which may necessitate *unprovisioning*, because either it has been lost/stolen, or it will be sold/recycled. For each case, we separately consider the implications for a regular consumer device and a trusted consumer device (i.e., the one containing the client's private key).

Regular Device: Since these devices do not hold private $\mathcal{VCR}$ keys, the unprovisioning process is very straight-forward. From a security perspective, the client only has to unlink the old device from the respective trusted device to prevent any further $\mathcal{VCR}$ issuance. This can be done from the trusted device, even if the old device has been lost/stolen. If the client has not already done so, they may wish to back-up or transfer any cookies and wrappers from the old device.

Trusted Device: The most difficult case is a loss, theft, or failure of a trusted client device. From the availability perspective, the client should be able to recover the master private key from a back-up. From a security perspective, the trusted device should have some type of access control (e.g., using a PIN and/or a fingerprint) that would protect the private key even from an adversary who has physical access to the device. If the trusted device is being sold/recycled, the client should securely back-up or transfer the master private key to a new trusted device using techniques such as Presence Attestation [178] or equivalent.

4.6 Implementation

We now describe the proof-of-concept implementation of `VICEROY`. It consists of a server, a browser extension, and a native messaging application.

4.6.1 Server

Our server is a `Node.js` [43] based `HTTP` server (based on Express [19]) that is responsible for managing client sessions and implementing the server-side handling of $\mathcal{VCR}$s. The server uses `HTTP` sessions and indexes collected data during a session using *digital identifiers* [64] in the form of session cookies. For demonstration purposes, our server keeps collected data in memory, though it can also be stored in a database or by a third-party (e.g., third-party cookies are used). In our implementation we use ECDSA with curve `secp256k1` [55] for creating signatures, but any secure signature scheme could be used.

The flow starts with a client connecting to a server, e.g., to visit a web page. If this is the client's first visit, the server creates an HTTP cookie that includes a client id (uuid [44]). From here on, all data collected by the server about this client is associated with the client's unique id.[7] In response to the initial client request, the server notifies the client that it supports VICEROY by returning VICEROY endpoints in the HTTP headers. These endpoints are:

- VICEROY **Wrapper Request Endpoint:** This server endpoint is responsible for issuing wrappers to clients. The client sends the client id cookie set by the server during the HTTP session, and a freshly-generated $\mathcal{VCR}$ public key. The server then generates a wrapper that cryptographically binds these two pieces of information. The wrapper is essentially the server's commitment that it will accept a signature produced by the corresponding $\mathcal{VCR}$ private key as verification for consumer requests pertaining to the data associated with that client id.
- VICEROY $\mathcal{VCR}$ **Endpoint:** This server endpoint is responsible for receiving and verifying consumer requests. Specifically, it receives from the client a wrapper and a signed $\mathcal{VCR}$. The server verifies its own signature on the wrapper and then uses the $\mathcal{VCR}$ public key from the wrapper to verify the $\mathcal{VCR}$. Depending on the requested action, the server may return, modify or delete consumer data. To support these actions, the consumer request may include metadata corresponding to the requested action. For instance, for data access requests, metadata may include an encryption key to be used by the server to encrypt data sent to the client.

4.6.2 Browser Extension

The VICEROY browser extension is a Google Chrome extension, residing on the client device. This extension manages $\mathcal{VCR}$ public keys, implements the client-side aspects of the $\mathcal{VCR}$ flow, and includes a client-facing interface for controlling this flow. It also handles communication

[7] VICEROY can also use any existing client identifier cookie scheme – such as Google Analytics' _ga and _gid cookies [22].

between the browser and the trusted device (or application) that holds the master private key.

To realize derivable asymmetric keys in our prototype, we use a mechanism proposed for hierarchical deterministic wallets, commonly known as Bitcoin Improvement Proposal 32 [7], or `BIP32`. In `BIP32`, we have the notion of an *extended key*, which has 256 bits of entropy (called *chain code*) added to a normal public/private key. Extended keys can be used to derive one or more child keys, following the rule that private keys can be used to derive private or public keys, whereas public keys can only derive public keys.[8]

The browser extension has two parts: a background script and a pop-up script, both written in `JavaScript`, with additional `HTML` and `CSS` for the pop-up. If there is no `JavaScript` version of a library available for the browser (e.g., `crypto`, `BIP32`), we use Browserify [12] to convert `Node.js` libraries into `JavaScript` files that can be loaded by the browser.

Background Script

The background script contains the majority of our client-side functionality. It uses the browser's API (`chrome.webRequest.onHeadersReceived`) to scan `HTTP` response headers to detect which servers support `VICEROY`. If present, it parses the `VICEROY` wrapper request and $\mathcal{VCR}$ endpoints, along with the client id cookie, from the headers.

Using the device public key, it derives a session-specific $\mathcal{VCR}$ public key via `BIP32` [8]. The derivation path of a $\mathcal{VCR}$ public key is in the form `m/i/j`, where `m` denotes the master private key, `i` the device ID, and `j` the total number of sessions created on the device. In other words, `m/i` represents the device public key and `m/i/j` represents the child public key for the j^{th} session. After deriving a $\mathcal{VCR}$ public key, the background script sends a `POST` request to the

[8]`BIP32` also has the notion of *hardened* vs. *non-hardened* keys, but in this work we only use non-hardened keys.

$\mathcal{VCR}$ wrapper request endpoint along with client id cookie set during the `HTTP` session.

The background script then stores the server-returned wrapper along with the key derivation path. Since we generate a new wrapper for each session, the number of stored wrappers may become quite large depending on how many new sessions the client establishes (although note that the number of wrappers is less than or equal to the number of cookies the client must store, so this is not infeasible). To improve the efficiency of searching for a particular client id cookie, we use a hashmap of client id cookies and their corresponding wrapper info. We also store the `URL` of the website and the time of the visit alongside the wrapper to assist clients in selecting the session for which they wish to create a $\mathcal{VCR}$.

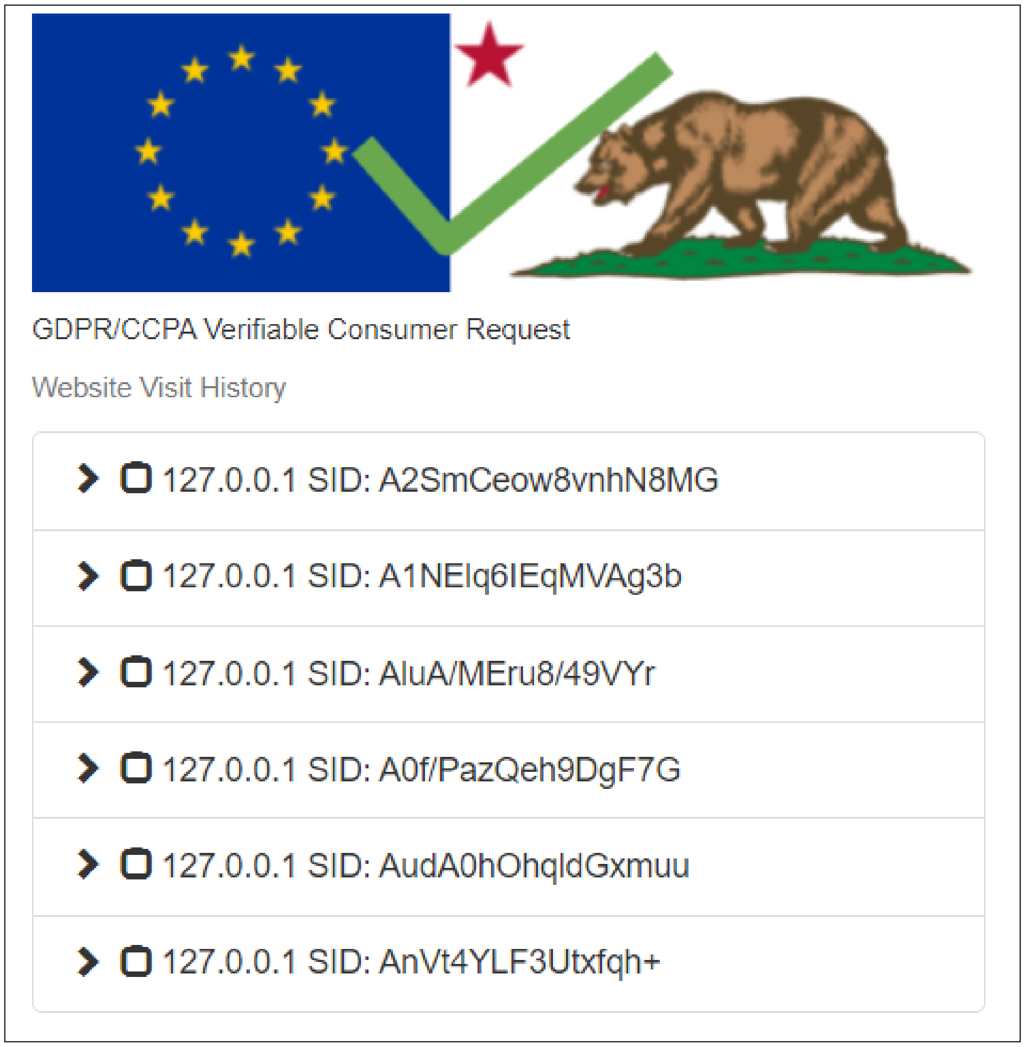
The background script can store this data using any available storage service. For example, to commit this data to the local device, we used the local storage API (`chrome.storage.local`). However, other storage services could also be used, such as Google Chrome's synced storage API (`chrome.storage.sync`) which would allow clients to synchronize `VICEROY` data between different devices.[9]

Note that all the above operations are performed asynchronously in the background, so the client does not observe any additional latency in loading the page.

Pop-up

An extension pop-up is a small web page that appears when the client clicks on the extension icon next to the `URL` bar. As shown in Figures 4.3 and 4.4, the pop-up displays session information for a $\mathcal{VCR}$ key along with the history of web links visited when the session was active. It also displays the types of $\mathcal{VCR}$s (access, modify and delete) that the client can submit.

[9] Although we experimented with this option, we found that Chrome synced storage currently imposes a limit on the amount of data that can be stored at any given time.



GDPR/CCPA Verifiable Consumer Request

Website Visit History

- 127.0.0.1 SID: A2SmCeow8vnhN8MG
- 127.0.0.1 SID: A1NElq6IEqMVAg3b
- 127.0.0.1 SID: AluA/MEru8/49VYr
- 127.0.0.1 SID: A0f/PazQeh9DgF7G
- 127.0.0.1 SID: AudA0hOhqldGxmuu
- 127.0.0.1 SID: AnVt4YLF3Utxfqh+

Figure 4.3: `VICEROY` browser extension pop-up displaying multiple sessions with localhost (`127.0.0.1`). The `SID` is first few bytes of the $\mathcal{VCR}$ public key.

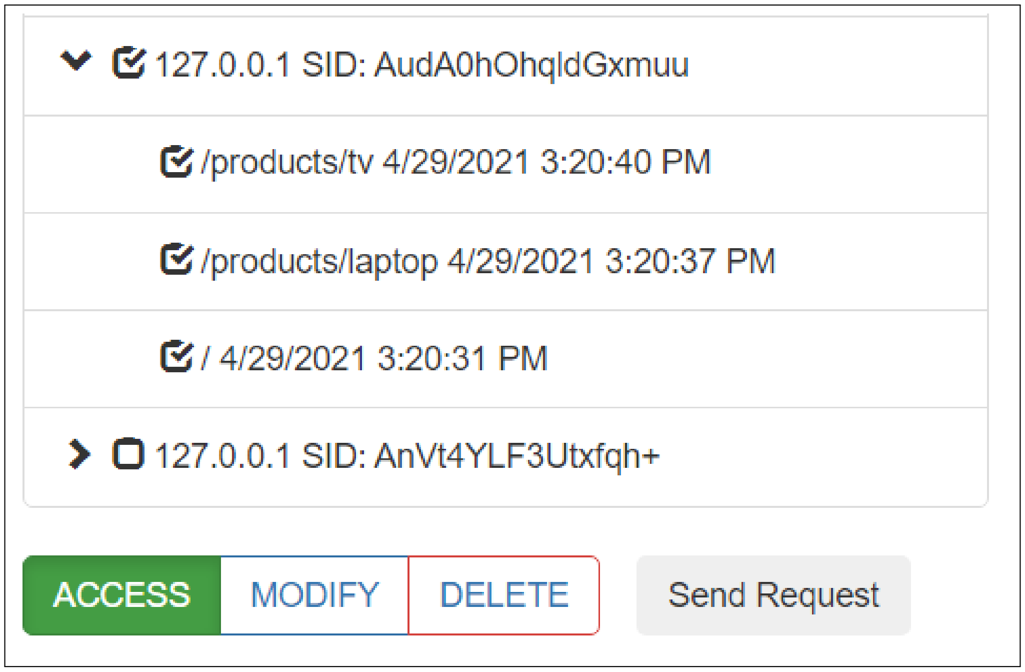

127.0.0.1 SID: AudA0hOhqldGxmuu
/products/tv 4/29/2021 3:20:40 PM
/products/laptop 4/29/2021 3:20:37 PM
/ 4/29/2021 3:20:31 PM
127.0.0.1 SID: AnVt4YLF3Utxfqh+
ACCESS
MODIFY
DELETE
Send Request

Figure 4.4: `VICEROY` browser extension pop-up displaying the history of visits in each session. Clients can select which session and which type of $\mathcal{VCR}$ (`ACCESS/MODIFY/DELETE`) they wish to generate.

To issue a $\mathcal{VCR}$, the client first chooses the session(s) and the type of request. Depending on that choice, the pop-up script prepares a $\mathcal{VCR}$ for signing. The resulting $\mathcal{VCR}$ is then passed to the native application (see Section 4.6.3) for signing. Once the request is signed using the corresponding $\mathcal{VCR}$ private key, it is retrieved by the pop-up (using the background script), and the $\mathcal{VCR}$ is sent to the server's $\mathcal{VCR}$ token verification endpoint. Finally, the server's response is displayed in the client's browser.

4.6.3 Native Messaging Application

To simulate a client-controlled trusted device, we created a `Node.js` [43] application that holds the master private key. Much alike the trusted device, this application signs $\mathcal{VCR}$s received from the background script using private keys derived from the master private key with the provided key derivation path (e.g., `m/0/1`). Communication between this application and the browser is done via the Chrome native messaging protocol [42]. We use the `native-messaging` package [41] to support both Firefox and Chrome. As explained in Section 4.5.4, the key storage mechanism must support derivable asymmetric key operations (e.g., using `BIP32`). There already exist multiple implementations of `BIP32` in different languages (e.g., JavaScript [8], Golang [9], Python [11], Java [10] and C [25]), which can be used.

4.7 Evaluation

This section presents the evaluation of `VICEROY`. We start with the security analysis of `VICEROY` and show how it satisfies requirements in Section 4.4. Next, we analyze the performance of the client and server-side components in terms of latency, bandwidth, and storage requirements. Finally, we discuss the deployability aspects of `VICEROY`.

4.7.1 Security Analysis

Consumer Impersonation Attacks: `VICEROY` relies on public-private key-pairs associated to sessions. An attacker that attempts to impersonate the client and obtain data for a given session needs access to the corresponding $\mathcal{VCR}$ private key for that session. This is difficult, as `VICEROY` requires a trusted consumer device to store the master private key for `BIP32` and access to this device is strictly controlled by the consumer. Moreover, the private key never leaves the trusted device and all $\mathcal{VCR}$ generation takes place within that device, further preventing attacker from obtaining $\mathcal{VCR}$ private keys.

Replay Attacks: This attack can occur when an adversary obtains (by eavesdropping or other means) a genuine $\mathcal{VCR}$ and later replays it to the server. The replayed $\mathcal{VCR}$ is verifiable by the server since it was originally generated by the actual client. There are several intuitive ways to mitigate replay attacks:

1. In principle, the server could reliably detect all replays if it were to keep record of all old (already processed) $\mathcal{VCR}$s. However, for obvious reasons, this is not an attractive option.
2. The client can include a timestamp in the $\mathcal{VCR}$. The server then would only accepts a $\mathcal{VCR}$ if it is fresh, i.e., $\mathcal{VCR}$ timestamp and server's current time are reasonably close. To be practical, this approach requires (at least loosely) synchronized clocks and the server should accept $\mathcal{VCR}$s with certain (configurable) tolerance.
3. Another alternative is a challenge-response protocol where the server issues the challenge. However, this requires the server to keep state of the challenge until it receives the response from the client.

(1) is unworkable due to storage requirements, and in (3), the server's state can be easily abused via DoS/DDoS attacks similar to TCP SYN flooding, though the well-known cookie-based countermeasure[10] might be applicable here. Additionally, application level solutions such as CAPTCHAs [168] or rate-limiting mechanisms (e.g., [147]) could supplement this mechanism. This leaves (2), i.e., timestamps and loosely synchronized clocks. However, within the tolerance period (maximum clock skew) replays are still possible. One natural and effective way to plug this "hole" is to require the server to keep a *cache* of recent $\mathcal{VCR}$s, i.e., those that arrived within the tolerance period.

We note that the exact consequences of a successful replay attack depends on the $\mathcal{VCR}$ type, i.e., access/modify/delete. For an access $\mathcal{VCR}$, data confidentiality can be assured by including the client's data encryption public key in the request. Replaying a modify $\mathcal{VCR}$ might override existing data if a $\mathcal{VCR}$ does not include both old and new values. Finally, a replayed delete $\mathcal{VCR}$ has no real effect since the data to which it refers is presumably already deleted.

[10]See: https://cr.yp.to/syncookies.html

Public Key Injection Attacks: Recall that per-session client identity is tied to the $\mathcal{VCR}$ public key via the wrapper created by the server. An attacker may try to replace the client $\mathcal{VCR}$ public key with their own public key during wrapper request. This can occur if there is *(1)* an active network-level adversary, and/or *(2)* malware on client device. For the former, if a different public key were injected during wrapper generation, the wrapper returned from the server would include this public key. This attack can be thwarted by simply having the `VICEROY` browser extension compare the public key it sent with the public key in the returned wrapper. The latter is difficult to defend against. Since the malware has full control over the client device, it can replace the client public key with its own during wrapper request. Even if the consumer attempts to verify the wrapper using its own public key, the malware can always replace that with its own. The only way to prevent this from happening is to verify wrappers in a TEE such that malware cannot forge the verification result and to mimic a secure channel by the server generating wrappers using keys recognizable by the TEE (e.g., via a PKI mechanism).

Key Leakage: An attacker could exploit `BIP32` to learn private keys. One well-known weakness of `BIP32` is that knowledge of a parent extended public key as well as any non-hardened child private key (descended from that parent public key) can leak the parent extended private key. This would then ultimately leak the entire set of private keys generated from the parent.

`VICEROY` mitigates this attack in the following way. Since the parent extended public key is only exists within the client device, the adversary must first attack the client device and obtain the parent extended public key, e.g., infect the client device with malware. However, even if the adversary learns the parent extended public key, it cannot obtain the non-hardened child private key, because such private keys are only generated during $\mathcal{VCR}$ issuance and they do not leave the trusted device. Since we assume the trusted client device is free of malware, the adversary has no means of obtaining such private key.

Consumer/Device/Request Linking Attacks: We consider three types of linkage: *(1)* two or more $\mathcal{VCR}$s, *(2)* $\mathcal{VCR}$s to a device, and *(3)* $\mathcal{VCR}$ to a client. Since `BIP32` guarantees that derived/child public keys are unlinkable (to each other and to their parents), none of these linkage attacks are possible. Of course, if the parent public key is leaked from the client's trusted device, *(1)* and *(2)* would no longer hold.

Metadata Leakage: This attack mainly applies to insecure client-server connections. If an attacker eavesdrops on a $\mathcal{VCR}$, certain metadata about the request can be inferred, e.g., $\mathcal{VCR}$ action and perhaps new values for `MODIFY` operations. This attack can be mitigated by super-encrypting the entire $\mathcal{VCR}$ with the server's long-term public key.

Insecure Communication channel: The lack of a secure communication channel might expose client data to eavesdroppers, and allow replay and public key injection attacks in addition to metadata leakage. We discuss each of these attacks in detail in their corresponding subsections and show how `VICEROY` prevents each of them.

Client-side malware: Recall that the main purpose of the malware is to conduct unauthorized operations on client data. This is possible either via *(1)* replay attacks or *(2)* generating new $\mathcal{VCR}$s. We discuss how `VICEROY` prevents *(1)* in Section 4.7.1. Since we assume that the trusted client device is free from malware, `VICEROY` prevents *(2)* as well. This also highlights one of the key differences between using asymmetric tokens and symmetric tokens for $\mathcal{VCR}$s. Using asymmetric tokens allow generation of one $\mathcal{VCR}$ per client authorization. In contrast, symmetric tokens, even if they are encrypted and decrypted on demand with client's approval, need to be in plaintext to be sent to the servers. The malware then can use these exposed tokens to forge future $\mathcal{VCR}$s.

In addition, using symmetric tokens allows malware to access data *before* its infection period, thus launching a *temporal attack*. For example, assume the malware infects the client device at time t. If symmetric tokens were used for data operations, the malware could use prestored tokens to gain access to data *prior* to time t. Since we can prevent such attacks with asymmetric tokens, `VICEROY` provides better security and privacy compared to existing symmetric token based systems.

4.7.2 Latency Analysis

In the experiments, we used an Intel(R) NUC with an Intel(R) Core(TM) i5-7260U CPU @ 2.20GHz quad-core chip and 32.0 GB RAM running Ubuntu 18.04 LTS. The version of the Google Chrome browser we used was 91.0.4472.114 (Official Build) (64-bit).

We separated latency experiments into three parts: `VICEROY` wrapper flow, session history update, and $\mathcal{VCR}$ flow. The results represent (unless otherwise stated) an average of 10 runs, with storage left as is between runs. For these experiments, all data is stored in local storage and the measurements may change depending on the underlying storage technology (e.g., storage on memory, hard-drives, or cloud-hosted databases).

Note that most of these latencies will not be visible to the user because they happen asynchronously e.g., after a webpage has been loaded. We include these to provide an indication of the computational cost of `VICEROY`. The only user-visible latency is the $\mathcal{VCR}$ flow, which is expected to be an infrequent operation.

`VICEROY` **Wrapper Flow Latency:** Recall that `VICEROY` uses the regular flow of a `HTTP` session initialization mechanism to receive wrappers. We divide this `VICEROY` wrapper flow into four phases:

- *Key Derivation*: When an unknown client requests a connection, the server returns $\mathcal{VCR}$ endpoints as discussed in Section 4.6. The client parses these endpoints and the client identification cookie (if it exists) embedded in the headers, derives a $\mathcal{VCR}$ public key, and prepares a wrapper request.

Table 4.1: Latency Results for VICEROY Wrappers.

	Key Derivation	Wrapper Generation	Wrapper Verification	Wrapper Storage
VICEROY Token Flow	24.6ms	0.4ms	18.8ms	6.5ms

- *Wrapper Generation*: The server generates a wrapper using the $\mathcal{VCR}$ public key provided by the client (and the client identification cookie).
- *Wrapper Verification*: After receiving the wrapper from the server, the client verifies the wrapper using the server public key and confirms that the wrapper associates the $\mathcal{VCR}$ public key and the client identification cookie.
- *Wrapper Storage*: The client saves the wrapper along with other data such as the $\mathcal{VCR}$ public key derivation path and endpoints.

Table 4.1 shows how long each phases take. Wrapper Verification phase incurs the most cost since it performs a signature verification. In addition, since Key Derivation and Wrapper Verification phases run in the browser extension, they take longer compared to a standalone application. This is apparent in how fast Wrapper Generation phase is even though it includes a public key operation.

Session History Update Latency. VICEROY also has a history mechanism for all sessions where HTTP requests can be associated with their respective sessions. This feature allows clients to see which sites they have visited during a session. VICEROY implements this feature using the request headers and the sent cookies. The cookies in the request headers are looked up on a hash map and then associated with their corresponding sessions. With only one cookie in the header, the matching takes 0.1ms whereas the history update and saving takes 5.5ms on average.

$\mathcal{VCR}$ **Flow Latency.** Lastly, we present the latency results of $\mathcal{VCR}$s. This flow consists of the following two steps:

Table 4.2: $\mathcal{VCR}$ Latency Results.

	$\mathcal{VCR}$ Generation	$\mathcal{VCR}$ Verification
$\mathcal{VCR}$ Flow	14.7ms	0.9ms

- $\mathcal{VCR}$ *Generation*: When a client selects a session and a $\mathcal{VCR}$ type (`ACCESS/MODIFY/DELETE`), the browser extension prepares a $\mathcal{VCR}$ to be signed by the trusted application – here, a native messaging application as described in Section 4.6. This application which holds the master private key for $\mathcal{VCR}$s derives the $\mathcal{VCR}$ public key for the session using the key derivation path provided. It signs the overall request using the private key corresponding to the derived public key. The signature is the returned to the extension.
- $\mathcal{VCR}$ *Verification*: The server receives the $\mathcal{VCR}$ and verifies the included wrapper. In addition, it extracts the $\mathcal{VCR}$ public key for this session from the wrapper and verifies the overall $\mathcal{VCR}$ using the $\mathcal{VCR}$ public key.

Table 4.2 shows various latency results. Similar to Table 4.1, it shows that a signature generation operation done by the native messaging application takes longer, as compared to a standalone server. This is due to data passing delay between popup and background scripts, and running the native messaging protocol between the application and the browser. Based on these results, the server's cost to verify $\mathcal{VCR}$s is minimal.

4.7.3 Bandwidth Analysis

We measured the amount of bandwidth used when obtaining wrappers and verifying $\mathcal{VCR}$s as well as client-side storage needed to store necessary information for `VICEROY`. For demonstration purposes, our server kept the visit history for each client in memory and these list of visits were returned to the clients upon successful verification of the consumer request. We used the same setting as the latency analysis for this evaluation. The Chrome debugging console was used to record the communication between the browser and the server.

Table 4.3: Bandwidth Usage (HTTP header + payload).

	Request	Response	Total
Obtain Wrapper	0.72 kB (0.62 + 0.10)	0.38 kB (0.23 + 0.15)	1.10 kB
Verify $\mathcal{VCR}$	0.99 kB (0.68 + 0.31)	0.28 kB (0.23 + 0.05)	1.27 kB

In this evaluation, the client first sends an `HTTP GET` request to the server at the session start. After receiving the $\mathcal{VCR}$ endpoints and the client identification cookie, a `POST` request is sent to the wrapper request endpoint. To simulate a consumer sending a $\mathcal{VCR}$, we generated and sent a $\mathcal{VCR}$ to the server with type `ACCESS` and the client received the stored data at the server – a visit history with a single entry. This request included the wrapper, $\mathcal{VCR}$ public key and the signature over the request using the corresponding $\mathcal{VCR}$ private key.

Table 4.3 shows the results for our bandwidth evaluation. Numbers represent `HTTP` header and payload sizes that were transmitted between the client and the server. We used `JSON` objects as payloads. We optimized these objects by substituting long key names in `JSON` key-value pairs with single letter characters. For instance, instead of {`"vcr_key": ...`}, we used {`"v": ...`}. Compared to the unoptimized format, bandwidth consumption was reduced by 16% and 14% respectively for `HTTP` requests and responses when obtaining a wrapper. The reduction for `HTTP` requests during the $\mathcal{VCR}$ `ACCESS` request was 14%. For the wrapper flow, `VICEROY` introduced only an additional 1.10 kB in total – which is a small price to pay considering deployability gains.

4.7.4 Storage Analysis

For the client-side data storage evaluation, we used the `chrome.storage.sync.getBytesUsed` function to obtain the approximate used storage size. Similar to the bandwidth evaluation, we the storage for both unoptimized and optimized `JSON`, using the same optimization technique as in the bandwidth evaluation. Additionally, we reduced the space needed to represent

Table 4.4: Client-side Storage Measurement.

	Baseline (0 history)	100 histories
Unoptimized	0.46 kB	10.70 kB
Optimized	0.38 kB	5.06 kB
Improvement	17%	53%

the date as well as the `URL` in the *history* section (refer to Section 4.6.2). The date was represented using the `UNIX` standard and the only the `URL` path was stored in the history section.

Table 4.4 shows the results of client storage measurements. The measurement for the Baseline storage requirements included server endpoints, $\mathcal{VCR}$ and server public keys and other metadata that would be used to issue $\mathcal{VCR}$s – with no client visit histories. `VICEROY` only needed 0.46 kB for this and we were able to reduce it to 0.38 kB using our `JSON` optimization, an improvement of 17%.

Our second measurements included 100 visits to the same `URL`. This simulates the case where a client frequently visits a certain webpage. If we use the unoptimized format, the client is required to store more than 10 kB of data. However, we can reduce the storage size by 53% by using the optimized format.

Overall, we can say that `VICEROY` only introduces a slight overhead in terms of communication bandwidth and client-side storage.

4.7.5 Deployability Analysis

We analyze deployability of `VICEROY` from the perspective of both the server and client in terms of required changes.

Server's perspective. The required changes for the server side are the following: (1) Create and maintain a public/private key pair for generating wrappers, and (2) Create and maintain wrapper and $\mathcal{VCR}$ endpoints. Due to the wrapper design, the server is not required to change how it assigns identifiers to clients and how it collects data. The integration of `VICEROY` to existing servers can be further simplified by releasing `VICEROY` modules for popular server frameworks (e.g., `Node.js`).

Client's perspective. The client will have to make the following changes: (1) Download and install `VICEROY` software to their trusted device that generates and manages master public/private key pair, and (2) Download and install `VICEROY` browser extension to the rest of their devices to request wrappers and send $\mathcal{VCR}$s. These software can be manually installed or installed via device-supported app stores.

4.8 Discussion

This section discusses several applicability issues of `VICEROY`.

4.8.1 Multi-Device Support

Given the proliferation of smartphones, computers, and various IoT gadgets into many spheres of everyday life, we expect that most clients own multiple devices with varying capabilities. `VICEROY` has been designed with this scenario in mind.

Firstly, the computational requirements of `VICEROY` can be met by any device that has the capability to establish TLS connections. For example, a device that can complete a TLS handshake is sufficiently powerful to verify the signature on a wrapper. Storage is not an issue because wrappers can be stored anywhere, even on a 3^{rd} party storage. Furthermore, devices without display capabilities can still use `VICEROY` by requesting wrappers and committing these to synced storage. Any other device, owned by the same client, with an appropriate display can fetch these wrappers and perform data operations.

Secondly, the client does not need to generate a separate master private key per device – a single private key on a trusted device can be used to derive any number of distinct device public keys for use on the other devices. Once a device has been issued with its device public key, it can derive $\mathcal{VCR}$ public keys for each session independently, using `BIP32`, without needing to interact with the trusted device. Issuing a $\mathcal{VCR}$ does require use of the trusted device in order to derive the signing key from the master private key, but we believe this to be reasonable because this is a security-sensitive operation, which is performed relatively infrequently.

Thirdly, in `VICEROY`, $\mathcal{VCR}$s do not need to be issued from the same device that originally interacted with the server. This is in contrast to other approaches that may use device identifiers to authenticate requests. This is important because, as discussed in Section 4.5.4, a device may need to be unprovisioned (e.g., due to loss, theft, failure, selling, or recycling), but the client should still be able to issue $\mathcal{VCR}$s for the interactions they made on that device. This is possible in `VICEROY` if the client backed-up or transferred the relevant cookies and wrappers from the old device.

4.8.2 Multi-$\mathcal{VCR}$ Support

Due to the use of unlinkable $\mathcal{VCR}$ public keys, $\mathcal{VCR}$s cannot be linked to each other or to the originating client. Although this benefits consumer privacy (one of the requirements defined in Section 4.4), it also requires clients to generate and sign $\mathcal{VCR}$s for each session. To reduce this overhead, a client can amend its key derivation mechanism. For example, if a client prefers to use only one $\mathcal{VCR}$ to refer to the combined collected data for all sessions with a particular website, can use the derivation path `m/i/s/j` where `m/i` is the derivation path of the device key as before, `s` is a server id and `j` is a server-specific (rather than global) session counter. The client then collects all wrappers and generates a unified $\mathcal{VCR}$ by signing it with the private key corresponding to the server $\mathcal{VCR}$ public key (`m/i/s`). This server key is sent to the server. In turn, the server can derive $\mathcal{VCR}$ public keys for individual sessions and verify all wrappers. For a new session with the same server, the client simply updates the server id and repeats the process as needed.

4.8.3 Multi-Communication Protocol Support

So far, we focused on `VICEROY` being used over `HTTP(S)`, since this is the most common way to access Web services. However, `VICEROY` can support any stateless protocol by assigning a unique identifier to each request and using that identifier when generating a wrapper. Thus, as described in Section 4.8.1, `VICEROY` is also applicable to applications that do not use `HTTP`, e.g., desktop or applications (other than Web browsers) that use other protocols to interact with online servers. Such applications can use `VICEROY` wrappers to bind client-generated public keys to any type of symmetric session identifier, and use the same protocol to issue $\mathcal{VCR}$s for data associated with that identifier.

4.8.4 Shared Devices, aka *The Roommate Problem*

Some client devices may be shared between multiple individuals, e.g., a smart TV shared by members of a household. This results in the so-called "Roommate Problem", where all data collected from multiple individuals is lumped together. This poses a problem because it is difficult to decide what data is associated to a specific individual. Although this is highly situation-dependent, a general principle for submitting a $\mathcal{VCR}$ for shared data is that all clients who use the device should somehow indicate their consent for the $\mathcal{VCR}$.

With minimal modifications, `VICEROY` can provide a technical solution for this problem. One simple and effective approach is as follows: During initial enrollment of a shared device into a household with n members, each member derives a $\mathcal{VCR}$ public key as the device key from each of their master secrets and provisions it to the device. (We assume that each member has a distinct trusted client device). This way, instead of a single $\mathcal{VCR}$ key, the shared device uses n $\mathcal{VCR}$ keys in the $\mathcal{VCR}$ flow. In other words, n keys are derived and included in the wrappers and $\mathcal{VCR}$s must be signed by the n corresponding private keys. The server verifies all n signatures when it receives a request. This prevents one member from issuing a $\mathcal{VCR}$ without the consent of all others.

4.8.5 3^{rd} Party Storage

One distinctive feature of `VICEROY`, as opposed to simply using cookies, is that possession of a wrapper alone is not sufficient to issue a $\mathcal{VCR}$. This means that wrappers can be stored by third parties and fetched only when needed, i.e., when generating a $\mathcal{VCR}$. This relaxes client-side storage requirements and creates a possible new business opportunity for (paid) service providers that will manage wrappers on behalf of clients.

4.8.6 Broad Identifier Support

`VICEROY` wrappers are a general means of binding client identification cookies to client-generated public keys. Importantly, this method is *non-invasive* and does not impose any constraints on the cookie. This is useful if a server changes its identification method, e.g., changes what is included in its cookies. `VICEROY` can also support new identification methods that may be developed in future.

4.8.7 Third-party Cookie Support

Third-party cookies are claimed to provide better, more personalized advertisements. Although such cookies are commonly considered to be detrimental to privacy [143], they are widely used. VICEROY can support third-party cookies by modifying the browser extension to also capture traffic going to third parties. The modified extension would extract all third-party cookies and store them alongside the first-party cookie. To obtain the wrapper, the client can either: (1) send all cookies to the first-party server, which would then contact all third-party servers and obtain wrappers for the client, or (2) visit the third-party wrapper endpoints directly.

A related cookie type is one that is *synced* across multiple servers. This is a technique commonly known as *cookie syncing*, ID syncing, or cookie matching [64, 88, 17, 167]. Instead of placing multiple cookies on the client device, a set of servers associate the data they collect under a unified identifier. VICEROY can support this type of cookie by having the client visit wrapper endpoints of all involved third-parties, or by having the first-party server visit them on the client's behalf.

4.8.8 Further Privacy Considerations

Although VICEROY provides unlinkability by design, the nature of how $\mathcal{VCR}$s are submitted may point the servers in the direction of clients. For instance, by observing the metadata, such as IP addresses, of different $\mathcal{VCR}$ requests, servers might try to link different $\mathcal{VCR}$ requests to the same client. An obvious solution to this is to use anonymity networks (e.g., Tor [58]) and avoid sending $\mathcal{VCR}$ *bouquets* where multiple requests are submitted through the same connection. One could also introduce random delays between $\mathcal{VCR}$s to prevent correlation based on timing.

For sessions, using above approaches will hinder performance. Therefore, sessions are likely to include metadata which may lead the servers to link them. There is a one-to-one mapping of a $\mathcal{VCR}$ public key and a session, thus revealing $\mathcal{VCR}$ keys to the servers might allow servers to link $\mathcal{VCR}$s as well. To prevent this, we consider two approaches.

First, for data access requests, cryptographic solutions such as Private Information Retrieval (PIR) [91] can be used to hide the identity (i.e., $\mathcal{VCR}$ public key) of the data requested. For modify and delete operations, the problem is challenging because, if the data are in plaintext, servers could detect which piece of data has been updated/deleted. Furthermore, PIR is likely to incur a high bandwidth burden since databases may include data from a long period of time (e.g., 10 years) and they might need to be sent to the client.

An exciting approach to solve both these problems is to utilize Trusted Execution Environments (TEEs) on the server side. By means of remote attestation, after ensuring the expected code is run on a server-side TEE, clients can create a secure channel to the TEE and send their $\mathcal{VCR}$ keys. The TEE then can find the matching row in the database and return the associated data from a memory database. Recent versions of Intel Software Guard Extensions (SGX) [62] can support up to a terabyte of enclave memory, in which this database can be stored. This TEE-secured database can be populated with data collected during a secure connection between a client and a server, e.g., using techniques such as LibSEAL [72]. Furthermore, Intel SGX DCAP [32] allows attestation reports that can attest the origin of the report (i.e., the server) rather than only attesting that the TEE is *a genuine Intel SGX device.*

Server-side TEEs also allow servers to prove to clients how their data are used – also using remote attestation. $\mathcal{VCR}$ responses can be generated with such guarantees, providing more transparency and trust between clients and servers.

4.8.9 Further Applications

Although VICEROY focuses on $\mathcal{VCR}$s from accountless clients, it can also supplement verification of $\mathcal{VCR}$s from account-holding clients. This can be useful, considering that passwords suffer from dictionary attacks and are often re-used on multiple servers.

Furthermore, VICEROY can be used as a basis for scenarios that require client re-authentication. For example, in the context of purchase transactions, receipts are currently used to prove that a client bought something from a merchant (server) in order to accept returns or perform exchanges. With VICEROY, a client can supply a fresh $\mathcal{VCR}$ public key during the purchase transaction and later generate a proof of ownership of the corresponding private key, which would confirm to the server that this is indeed the same customer. This design could allow clients to anonymously conduct merchandise returns or exchanges.

4.9 Related Work

Security of GDPR Subject Access Requests. GDPR and CCPA grant subjects the rights to request access to their personal data collected by businesses, by submitting a $\mathcal{VCR}$ or Subject Access Request (SAR). Unfortunately, insecure (or easily circumventable) SAR verification practices open the door to potential leakage of personal data to unauthorized third parties. Prior work [86, 100, 154] has investigated various social engineering techniques for bypassing existing SAR verification practices.

For example, [86] demonstrated that an unauthorized adversary can abuse the functionality provided by a business to update a victim subject's address (for both email and residential addresses). The adversary could then request access to *"their"* data from this new address. Out of 14 organizations tested, 10 gave out personal information and 7 of these contained sensitive data.

[100] investigated the use of address spoofing techniques (e.g., using `subject@protonmail.com` to spoof the owner of `subject@protonmail.com`), as well as more sophisticated techniques such as manipulation of identity card images. They found that 15 out of 41 organizations with manual verification processes leaked personal data. The remaining 14 organizations required an account-based login, which was impervious to such attacks, but is not available for the accountless consumers we consider in this work.

[154] performed an extensive evaluation of 150 companies' practices for SAR verification. They found that email address-based and account login were the most common, followed by device cookies, government IDs, and signed statements. Some organizations also requested utility bills, phone interviews, or credit card numbers. To bypass SAR verification, they created and sent a vague SAR letter to organizations. Out of 150 organizations, 24% disclosed personally-identifying information.

[77] also analyzed SAR verification practices for popular websites and third-party trackers. They found that, in addition to possibly being insecure, SAR verification could possibly also undermine the privacy of subjects in order to verify the request.

General Studies on GDPR Subject Access Requests. [166] performed a two-sided study of both data subjects and data-collecting organizations, with a focus on online advertising. For data subjects, the authors used consumer surveys to evaluate the usability of data transparency tools offered by the organizations, and to learn more about consumers' perceptions of these tools. They also carried out surveys and interviews of organizations to get their views on the privacy regulations and business practices for SARs. The results paint a picture of discrepancy between the consumer' perspectives and the collected data (which is also corroborated by [73]). Furthermore, consumers seemed to show little interest in seeing raw technical data. Similarly, [167] investigated SAR practices for online advertising companies and used cookie IDs to request collected data. The authors reported that some companies requested ID cards or affidavits, whilst others directly used the cookie IDs in the browser – none of which prove the requestor is really the consumer from whom the data was collected.

[134] studied mobile applications and observed an even more fragile ecosystem with discontinued apps and disappearing consumer accounts while processing SARs. Another conclusion of this study was to move away from email-initiated and manual processes which are prone to errors. In terms of compliance, the analysis by [126] showed 43% compliance with access requests vs. 57% compliance with deletion requests.

In addition to the above, [92] investigated cookie usage and how it is affected by privacy regulations. They report that 11% of EU-related websites set cookies for US-based consumers but not for EU-based consumers. Furthermore, up to 46.7% of websites that appear in both the 2016 and 2018 Alexa top 100,000 sites stopped using persistent cookies without consumer permission. In the standardization realm, [179] focused on "Do Not Sell" requests which informs the websites that they may not share the consumer's information with third parties. They built a browser extension (OptMeowt) which conveys "Do Not Sell" requests to the websites through headers and cookies.

Asymmetric access tokens. Origin Bound Certificates (OBCs) [101] (also see RFC 8471 [56]), aims to strengthen TLS client authentication. In OBC, the client generates a unique self-signed TLS client certificate for each website, in order to remain unlinkable across websites. Although this does not authenticate the client to the website (due to the self-signed certificate), it does allow the server to ascertain whether this is the *same* client from a previous interaction. One benefit of this is that cookies can be bound to an OBC, so that even if they are stolen, they cannot be used by an adversary. Another difference is that per-site certificates would not be suitable for accountless consumers, as these would allow servers to link together different visits from the same client. The alternative of generating per-TLS-session certificates introduces the key explosion problem.

4.10 Conclusion

Motivated by the recent GDPR and CCPA regulations granting (even) accountless consumers rights to access data gathered about their behavior by servers, we constructed and evaluated `VICEROY`, a framework for enforcing these rights. `VICEROY` is secure with respect to malicious clients and HbC servers. It is also privacy-preserving due to the use of unlinkable $\mathcal{VCR}$ keys. `VICEROY` is easy to deploy and incurs (based on our experiments) fairly low overhead. It is designed with TEEs in mind such that only a small secret can be kept in a TEE and used to create verifiable consumer requests with the user's approval.

Chapter 5

Balancing Security and Privacy in Genomic Range Queries

5.1 Introduction

In some application domains TEEs are not sufficiently powerful and/or common to realize end-to-end secure and private mechanisms. For such cases, we resort to cryptographic primitives. For example, the number of available genomic tests have surged due to the discovery of relations between genomic material and certain health traits. These tests, however, must adhere to the "need to know" rule – as little information as possible should be shared with the tester. Therefore, privacy has been the main focus of research in this domain. Security on the other hand is about ensuring that genomic material is authentic and has not been altered. This is an important problem since the genome owner could otherwise tweak the results to their advantage. In this chapter, we show how range queries can be used to provide both security and privacy in the genomic context. We utilize cryptographic methods, such as digital signatures and range proofs to ensure authentication, integrity and privacy.

Recent advances in genome sequencing, coupled with greatly reduced storage and computation costs, make genomic testing increasingly accessible to individuals. Today one can easily get his/her DNA digitized by a sequencing lab and store the result on a local device before performing a range of tests by engaging with a testing facility. Due to the inherent sensitivity of genetic material and often proprietary tests, privacy is the natural and key issue. However, so far insufficient attention paid to genomic security, which can have grave consequences, such as incorrect drug prescriptions or erroneous parentage outcomes.

Unfortunately, in genomic setting privacy and security are at odds with each other. In this chapter, we reconcile them in a particular setting of genomic range queries. We do so by designing a novel technique in the form of a secure and private sparse-set range query between genomic testing facilities and individuals. The proposed technique maintains *authenticity* and *completeness* of user-supplied genomic material, while maintaining its *privacy* by releasing only the minimum thereof. Our experiments show that this approach is practical, for all parties involved. We also extend it to a more generalized problem setting and discuss potential applications.

5.2 Motivation & Contributions

Technical improvements in DNA sequencing technology [141, 116, 125] and reduced costs have paved the way for ubiquitous and affordable genomic testing. As a result, tests that were used in the past mainly by doctors and legal authorities are becoming more available to the public. Such tests range from paternity/parentage to presymptomatic disease diagnosis, wherein the testing facility (denoted as "tester" hereafter) queries the user with various DNA locations, both specifics and ranges.

Due to the often-proprietary nature of these tests, testers need to keep specifics (such as queried locations) secret. Meanwhile, an individual naturally wants to reveal the minimal amount of genomic material, since, besides one's own highly personal and sensitive material, it includes significant information about past, current and future relatives. Consequently, genomic privacy justifiably attracted much attention from the research community and numerous privacy techniques have been proposed [75, 148, 76, 98, 97, 68, 69, 165, 131].

However, genomic security, even though equally important, received considerably less attention. In particular, authenticity and integrity of genomic data are often dismissed or over-simplified, even though they are crucial to the accurate outcome of genomic tests. An erroneous (whether maliciously caused or not) genomic test result can translate into grave health risks when used for medical diagnosis, or social and family risks when used for determining familial relationships. At the first glance, the genomic security problem seems simple and solvable with common cryptographic primitives, such as digital signatures. However, the main challenge stems from conflicting requirements among security, privacy, and efficiency.

Intuitively, genomic security needs data invariance to facilitate integrity checking, while genomic privacy needs flexibility and fine-granularity controlled by the user (who might act as an adversary in the security model). At the same time, efficiency favors processing aggregated genomic data. Not surprisingly, it is challenging to reconcile these three demands.

In this chapter, we focus on efficiently reconciling security and privacy in the context of genomic range queries. Specifically, we propose a technique for secure and private genomic testing using a combination of established cryptographic tools. Our contribution is two-fold:

- We propose a novel secure and private technique for genomic range queries that can be used as a building block in various protocols and genomic representations.
- We report on a prototype implementation and evaluation of the proposed technique.

NOTE: Our notations are summarized in Table 5.1.

5.3 Background

5.3.1 Genomics

Human DNA consists of around 3.2 billion bases (each is one of: [A]denine, [C]ytosine, [G]uanine, and [T]hymine) and only about 0.1% of DNA differs between any two individuals. Although it is not yet known exactly where these differences occur, many types of mutations can be used to identify an individual and determine susceptibility to diseases and/or sensitivity to drugs. Single-Nucleotide Polymorphism (SNP) is a type of genetic mutation representing a change in the nucleotide, such as $[A] \rightarrow [C]$. A given nucleotide is classified as an SNP if $< 1\%$ of the population carries a different nucleotide at that position [52].

Although a fully digitized raw human genome may take up to 200 Gbytes, due to relative sparsity of SNPs, a reference genome can be used to reduce storage and computation costs. Using a compact reference format, such as the $1,000$ Genomes Project variant call format (VCF)[1], a particular DNA can be represented using only 120 Mbytes. In this dissertation, we represent DNA as an array of mutations, each of the form: $\mathbf{V_i} = \{\mathbf{pos}, \mathbf{L_{pos}}\}$, where i is the index of the mutation in the array, pos is its position in the DNA (denoted as $V_i.pos$) and L_{pos} is the base letter at position pos.

Besides SNPs, other types of differences include Short Tandem Repeat (STR)s and Restriction Fragment Length Polymorphism (RFLP)s. STRs are 4-16 base pairs that repeat many times at certain locations. The number of repeats gives enough information to identify an individual. RFLPs are size differences of fragments after digestion of DNA with various restriction enzymes that fragments DNA at certain points.

[1]See: www.internationalgenome.org/wiki/Analysis/vcf4.0

5.3.2 Commitments

A cryptographic commitment scheme is a two-phase protocol between a prover and a verifier where the underlying commitment is *hiding* and *binding*. In the *commit phase*, the prover commits to a value and shares her commitment with the verifier. Later, in the *reveal phase* the prover reveals the hidden commitment and the verifier becomes convinced that this value was indeed chosen by the prover in the first phase. In other words, a commitment scheme uniquely *binds* the commitment to the value it *hides*. In this chapter, we use two types of commitments: Fujisaki-Okamoto and those based on secure cryptographic hash functions. A Fujisaki-Okamoto commitment [118] is of the form: $\mathbf{COM(z) = g^z h^r \mod n}$, where z is the prover-chosen secret value and n is a large composite integer with factorization unknown to either party. Also, g is an element with a large order in $\mathbb{Z}_n^*$, h is an element with a large order in the group generated by g. The discrete logarithms of g to h and h to g are also unknown to either party. The value r is randomly chosen by the prover from $[-2^s n-1, 2^s n+1]$, where s is a security parameter. This commitment statistically does not reveal any information to the verifier.

Commitments based on secure cryptographic hash functions rely on the fact that modern hash functions reveal no information about the value they *hide* if they are used with a sufficiently long random salt. In addition, since secure hash functions offer weak and strong collision-resistance properties, commitments based on such functions are *binding*.

A hash function based commitment is of the form: $\mathbf{H(z \mid r)}$, where H is a cryptographically-secure hash function (e.g., SHA256), z is the committed value, '$|$' denotes concatenation and r is a sufficiently long random bitstring used as a salt.

5.3.3 Range Proofs

A range proof convinces a verifier that a value is in a given range without revealing the actual value. Expansion rate of a range proof is $\delta = \frac{Y-X}{y-x}$, defined in terms of requested range $[x, y]$ and range $[X, Y]$ wherein the value is proven to reside in. Note that if this rate is 1, the range proof convinces the verifier that the value is in the requested range. We consider two range proof protocols from [78]. The first proves that a hidden integer $z \in [x-\theta, y+\theta]$ rather than in $[x, y]$ where $\theta = 2^{t+l+1}\sqrt{y-x}$ and t and l are security parameters. This protocol does not work well with small ranges. (Recall that DNA contains around 3.2×10^9 letters). The second protocol is similar, however, the secret value is first expanded to $z' = z \cdot 2^T$. With a carefully chosen T as $2(t+l+1) + |y-x|$, this protocol's expansion rate $\delta = 1$. We first describe it without the expansion phase.

Assume that the prover wants to prove that a secret value z is in $[x, y]$. To prove this in zero knowledge, it is enough to show that two properties hold: $z - x \geq 0$ and $y - z \geq 0$. To prove the first, the prover first writes $z - x$ as the sum of the greatest square less than z and a positive number, i.e., $z - x = z_1^2 + p$. Then, she creates two Fujisaki-Okamoto commitments, one for z_1^2 and one for p with random numbers r_1 and r_2 both chosen from $[0, 2^s n - 1]$. The prover creates another commitment to z using $r = r_1 + r_2$.

The prover then proves in zero-knowledge that the commitment to z_1^2 hides a square (i.e., ≥ 0) and commitment to p hides a number with absolute value less than $2^{t+l+1}\sqrt{y-x}$ using the method in [89]. Same procedure is performed for $y - z \geq 0$. As a result, the prover shows that $z \in [x - \theta_1, y + \theta_1]$ where $\theta_1 = 2^{t+l+1}\sqrt{y-x}$.

In the expansion phase, z is expanded as $z' = z.2^T$. Using the scheme above, it is shown that $z' \in [2^T x - \theta_2, 2^T y + \theta_2]$ where $\theta_2 = 2^{t+l+T/2+1}\sqrt{y-x}$. If T is chosen as $2(t+l+1) + |y-x|$, so that $\theta_2 < 2^T$, the verifier is convinced that z should be in $[x, y]$.

5.4 System & Security Models

We now describe our system and security models.

5.4.1 System Model

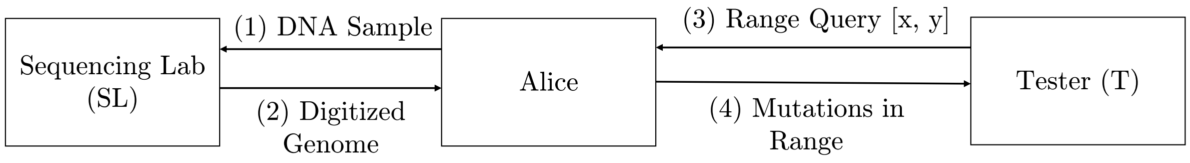


Figure 5.1: A Genomic Test Scenario with Order of Interactions

The system model consists of individual users, sequencing labs and testers, all of which are certified by a trusted authority $Auth$ (see Figure 5.2). We assume a global Public Key Infrastructure (PKI) which facilitates establishing trust in entities' public keys. For the sake of clarity, we only consider the case with only one instance of each entity type, i.e., one individual (Alice), one sequencing lab (SL), and one tester (T). Alice first obtains her digitized signed SNP representation from SL – a regulated and trusted sequencing lab capable of digitizing genetic material. Modern SL examples include hospitals and direct-to-customer (DTC) service providers. In addition, SL signs the digitized genome such that the authenticity of the data, *as a whole* can be verified by anyone with proper certificates. T needs to perform a genomic range test on Alice's genome. The test includes one or more queries sent to Alice. Each query is in the form of two integers corresponding to low and high positions in the genome. Upon receiving the query, Alice replies with relevant information, as shown in Figure 5.1.

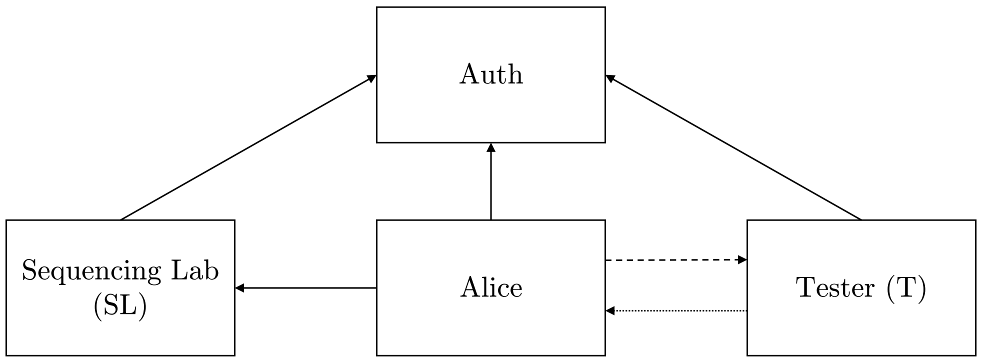


Figure 5.2: Trust between Entities (Truster → Trustee). Solid arrows are used to denote full-trust whereas dashed arrows are procedural trust (i.e., Honest-but-Curious) and dotted arrows denote no trust.

5.4.2 Security Model

In our security model, SL is fully trusted by both Alice and T. Nonetheless, Alice is not trusted by T, since she may want to provide altered genomic data, for various reasons. Alice may cheat in several ways: She may report modified data to T, fake data at a particular position, exclude data from a particular position, or a hybrid thereof. We consider T to be an honest-but-curious entity. It follows the protocol by issuing authorized queries. However, it aims to learn more information from Alice's genomic data than what the test requires. We suppose that queried ranges are pre-approved by some authority, e.g., via a public signed white-list of legitimate ranges for certain genomic tests.

An example of such a range (from 6p22.1 to 6p21.3 and consisting of about 4 megabases, see Cytogenetic location [26]) is *major histocompatibility complex (MHC)* which consists of a set of genes coding for proteins responsible for detecting foreign molecules in cellular level in vertebrates. Various SNPs in this region (rs9264942, rs4418214, rs2395029, rs3131018) have been shown to protect against human immunodeficiency virus (HIV) [164].

5.5 Range Query Formulation

We denote T's query as $Q = [x, y]$, where x and y are locations which indicate that T needs to obtaion all of Alice's genomic mutations in between. Without loss of generality, suppose that Alice has n mutations in the reference-based representation. We denote them as: $\mathcal{V}^* = \langle V_1, V_2, \cdots, V_n \rangle$. If Alice and T trust each other, she would only reveal $\mathcal{V}^*$ to T.

Under our security model, Alice and T are not mutually trusted due to respective privacy and security concerns.e propose a secure and private range query scheme consisting of a query response function run by Alice and a verification function run by T. The former is denoted as $\mathsf{Resp}(Q)$ which takes a query Q as input and outputs a result $(\mathcal{V} = \langle \{V_{i \dots j}\} \rangle, \sigma)$. The latter is denoted as $\mathsf{Verfy}(\mathcal{V}, \sigma)$ and it outputs either $\mathcal{V}$ or $\perp$, depending on the verification outcome.

1. **Authenticity.** All mutations reported by Alice must be authentic and not tampered with. More precisely, $\forall V \in \mathcal{V}, V \in \mathcal{V}^*$. Note that V reflects the mutation position. Hence, if $\mathcal{V}$ satisfies authenticity, it is trivial for T to check its validity, i.e., whether the reported mutation is between x and y. In other words, authenticity implies *soundness*.

2. **Completeness.** All mutations in $\mathcal{V}$ are reported by Alice. Namely, $\forall V \in \mathcal{V}^*, V \in \mathcal{V}$.
3. **Privacy.** There should be no information leakage on mutations other than those T is allowed to learn.

5.6 Proposed Construction

We now discuss several intuitive approaches to the range query problem described in Section 5.5 and compare them along several dimensions. We then describe the proposed technique

Table 5.1: Notation & Descriptions

Notation	**Description**
Alice	Individual with a digitized genome issued by a certified sequencing lab.
SL	Sequencing Lab capable of sequencing genetic material
T	A Tester that perform genomic tests
Q	A range query with boundaries $[x, y]$
$\mathcal{V}^*$	A list of mutations that describe Alice's DNA, used in Γ
Γ	Proposed secure and private range query scheme
$\mathsf{Resp}(Q)$	Response function of Γ, run by Alice and outputs a response $(\mathcal{V} = \langle\{V_{i \ldots j}\}\rangle, \sigma)$
$\mathsf{Verfy}(\mathcal{V}, \sigma)$	Verify function of Γ, run by T and outputs $\perp$ if any verification function fails.
$\mathsf{ZK}(z : R(z))$	Generates a zero-knowledge proof of knowledge of z such that $R(z)$ holds.
$\mathsf{ZK\_Verify}(p, q, r)$	Verifies the correctness of a zero-knowledge proof p with a commitment q, such that for the committed value the predicate r holds.
$\mathsf{Sign}_{SK_s}(t)$	Signs t using secret key of s SK_s
$\mathsf{Verify}_{PK_s}(t, u)$	Verifies the signature u on t using the public key of s PK_s.
V_i	A mutation tuple with the form $\{pos, L_{pos}\}$.
$COM(x)$	A commitment scheme allowing zero-knowledge range proofs over x. Equivalent to $COM(x, s)$ where s is a random bitstring used to create the commitment.
δ	Expansion rate of a range proof. Calculated as $\frac{Y-X}{y-x}$ where $[x, y]$ is range requested and $[X, Y]$ is range proven the value to be in

and argue its security, privacy, and efficiency.

Intuitive Approaches. One trivial solution is to use the full genome representation: at sequencing time, SL signs all position-base pairs for each position in the DNA and later, when T asks for all SNPs in a range, Alice provides all pairs in that range with their corresponding signatures. The tester can easily detect any missing positions. Inclusion of fake bases is impossible since Alice cannot generate signatures on behalf of SL. This approach provides authenticity and integrity and leaks no information about bases outside the queried range. Unfortunately, it has high computation and storage costs due to sheer size of the DNA. To reduce these costs, optimizations such as condensed and aggregated signatures can be applied, though the final cost would be still far from optimal. We refer the reader to [79] for a more detailed comparison of such methods.

Using a reference genome and representing the genome only in terms of mutations with respect to the reference genome reduces storage and computation costs substantially. However, it introduces the completeness problem. Suppose that T queries for mutations in range $[x, y]$ where x and y are DNA positions. If each mutation is signed by the SL, Alice could reply with a list of authentic mutations in that range. However, this would not prevent Alice from excluding one or more mutations. To ensure sequential continuity of mutations, SL could sign tuples consisting of two neighboring mutations sorted in ascending order, as suggested by [99]. However, this would entail revealing two tuples that contain mutations in positions immediately outside the lower and upper boundaries, respectively. Due to sparsity of genomic mutations, this could leak a substantial amount of sensitive information.

Proposed Approach. We assume that Alice receives from SL a digital representation of her DNA in the form of a list of mutations:

$$\mathcal{V}^* = \langle V_{-\infty}, V_1, V_2, \cdots, V_n, V_{+\infty} \rangle$$

wrt the reference genome, where $V_i \in \mathcal{V}^*$ and for each i, $V_i.pos < V_{i+1}.pos$. Note that two special mutations, $V_{-\infty}$ ($V_{-\infty}.pos < 0$) and $V_{+\infty}$ ($V_{+\infty}.pos > 3.2 \cdot 10^9$) are introduced to mark the lower and upper boundaries of the genome. Additionally, Alice receives a sequence of signatures $\gamma = \langle \gamma_s, \gamma_{s+1}, \cdots, \gamma_{s+n} \rangle$ where each $\gamma_i = \mathsf{Sign}_{SK_{SL}}(T_i)$. SK_{SL} is the secret key of SL and the tuple T_i is of the form:

$$T_i = \{COM(V_i.pos, s_{i_1}), COM(V_i, s_{i_2}),\ COM(V_{i+1}.pos, s_{i+1_1}), COM(V_{i+1}, s_{i+1_2})\}$$

COM is a commitment scheme realizing zero-knowledge range queries. It uses a random binary string (e.g., s_{i_1}) as a salt to hide the committed value. Salts are re-used for the same mutation in another tuple (e.g., in T_{i+1}, commitment for $V_{i+1}.pos$ and V_{i+1} uses same salts s_{i+1_1} and s_{i+1_2}, respectively).

Our scheme follows the model defined in Section 5. $\mathsf{Resp}(Q)$ and $\mathsf{Verfy}(\mathcal{V}, \sigma)$ are presented in Figure 5.3. Note that, if either $\mathsf{Verify}()$ or $\mathsf{ZK\_Verify}()$ fails, $\mathsf{Verfy}(\mathcal{V}, \sigma)$ terminates with output $\perp$. See Table 5.1 for description of verification functions.

```
Resp(Q = [x, y])
1 :  $\mathcal{V} \leftarrow \langle V_i, V_{i+1}, \cdots V_j \rangle = \langle V_k \mid x \leq V_k.pos \leq y \rangle$
2 :  $C \leftarrow \langle \{COM(V_{i-1}.pos, s_{i-1_1}), COM(V_{i-1}, s_{i-1_2})\}, \langle \{COM(V_i.pos, s_{i_2}\},$
3 :  $\cdots \{COM(V_j.pos, s_{j_2}\}\rangle, \{COM(V_{j+1}.pos, s_{j+1_1}), COM(V_{j+1}, s_{j+1_2})\}\rangle$
4 :  $\gamma \leftarrow \langle \gamma_{i-1}, \cdots, \gamma_{j+1} \rangle$
5 :  $l \leftarrow ZK(V_{i-1}.pos : V_{i-1}.pos < x)$
6 :  $h \leftarrow ZK(V_{j+1}.pos : V_{j+1}.pos > y)$
7 :  Output($\mathcal{V}, \sigma = \{C, \gamma, l, h\}$)

Verfy($\mathcal{V}, \sigma$)
1 :  $Tup \leftarrow \langle T_{i-1}, \cdots, T_{j+1} \rangle$          // Reconstruct tuples from $\mathcal{V}$ and C
2 :  $\forall Tup_k \in Tup, \mathsf{Verify}_{PK_{SL}}(Tup_k, \gamma_k)$
3 :  ZK_Verify($l, COM(V_{i-1}.pos), V_{i-1}.pos < x$)
4 :  ZK_Verify($h, COM(V_{j+1}.pos), V_{j+1}.pos > y$)
5 :  Output($\mathcal{V}$)
```

Figure 5.3: $\mathsf{Resp}(Q)$ and $\mathsf{Verfy}(\mathcal{V}, \sigma)$ functions.

5.6.1 Security & Privacy

We now analyze security and privacy of the proposed construction, with respect to goals stated in Section 5.5. Suppose that $\mathcal{V}$ is non-empty. There are two tuples of the forms (salts are not shown here):

$$\{COM(V_{i-1}.pos), COM(V_{i-1}), COM(V_i), COM(V_i.pos)\} \tag{5.1}$$

and

$$\{COM(V_j.pos), COM(V_j), COM(V_{j+1}), COM(V_{j+1}.pos)\} \tag{5.2}$$

Alice first reveals the commitments to mutations in the range, allowing T to validate the signatures. Then, Alice proves to T in zero-knowledge:

1. $V_{i-1}.pos$ is out of range.
2. $V_{j+1}.pos$ is out of range.

As a result, T is convinced that all other mutations in positions below $V_{i-1}.pos$ and above $V_{j+1}.pos$ are out of range. T also confirms that V_i and V_j are in range by using the revealed values of both commitments. Note that V_{i-1} and V_{j+1} can be $V_{-\infty}$ and V_{∞} respectively.

Suppose that $\mathcal{V}$ is empty. There is a tuple of the form:

$$\{COM(V_l.pos), COM(V_l), COM(V_{l+1}), COM(V_{l+1}.pos)\} \tag{5.3}$$

where $V_l.pos < x$ and $V_{l+1}.pos > y$. Alice proves the inequality to T in zero-knowledge.

Goal 1. Authenticity is ensured due to security of the underlying digital signature scheme used by SL to sign tuples.

Goal 2. Completeness is achieved using sequential linking of elements (mutation tuples) thus allowing T to detect any exclusion. Also, based on Goal 1, it is impossible for Alice to introduce any additional mutations.

Goal 3. Due to the use of zero-knowledge proofs and *hiding* commitments, no information about positions of any out-of-range mutation is revealed.

One special case occurs when V_{i-1} and V_l (in Equations 5.1 and 5.3) and V_{j+1} and V_{l+1} (in Equations 5.2 and 5.3) are: $V_{-\infty}$ and V_{∞}, respectively. Such sentinel commitments should be indistinguishable from other commitments. However, if needed to be revealed, they should clearly denote genome boundaries. This can be achieved by using positions 0 and $3.2 \cdot 10^9 + 1$ as $V_{-\infty}$ and $V_{+\infty}$ commitments, respectively, and allowing actual mutation positions to start from 1.

5.6.2 Instantiation of Proposed Construction

As an example commitment scheme COM, we use Fujisaki-Okamoto commitments [118]. For range proofs we use Boudot's technique [78] with expansion rate $\delta = 1$, i.e., the proofs convince the verifier that the committed value is in the actual given range, and not in the expanded range. Range proofs are used, in a sense, to prove that a position is outside the queried range (or equally, inside a range that is outside of the queried range). *Less than* and *greater than* are implemented as follows:

> To show that a value v is less than x in zero-knowledge, the range proof convinces the verifier that v in commitment $COM(v)$ is in range $[min - 1, x - 1]$, where min is the smallest possible value of v. Similarly, for greater-than proofs, the range proof convinces the verifier that $v \in [x + 1, max + 1]$, where max is the maximum possible value of v. In the genomic context, $max = 3.2 \times 10^9$ and $min = 1$.

5.7 Efficiency Considerations

Ideally, every step of a genomic test should be as efficient as possible. However, depending on the task, some of the workload can be shunted to a different step. If offline computation is possible or desired, some preprocessing can be done to increase efficiency of online steps. To this end, we implement the protocol from Section 5.6 as follows:

5.7.1 Commitments and Salt Generation

Fujisaki-Okamoto commitments [118] allow us to create range proofs without revealing mutation positions. However, calculation of exponents required during the creation of Fujisaki-Okamoto commitments makes this scheme an inefficient choice for mutation commitments. Hence, we use the SHA-2 hash function with a large salt for mutation commitments. Depending on storage requirements, salt generation can be done using a key derivation function, such as HKDF [133], or a Pseudo Random Number Generator (PRNG) with a sufficiently large seed. This would reduce the storage cost of salts. An alternative is to use a random number combined with the genomic position (similar to a sequence number) for salt generation, thus allowing efficient re-generation of salts.

We construct a tuple as:

$$\{FO(V_i.pos), SHA2(V_i, s_i), FO(V_{i+1}.pos), SHA2(V_{i+1}, s_{i+1}))\}$$

where FO is the Fujisaki-Okamoto commitment and s_i is a 128-bit salt.

5.7.2 Signatures

To provide authenticity, each tuple described in Section 5.7.1 is signed by SL at the initial sequencing time. To reduce signature sizes, we use EC-DSA with the elliptic curve `secp256r1`, which yields 512-bit signatures.

5.8 Evaluation

We now report on the evaluation of the proposed construction. We have implemented a prototype in Java on a PC with an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz chip and 16GB of RAM. We use the code from [63] to implement commitments and proofs realizing range queries, and Bouncy Castle [36] for other crypto primitives. For Fujisaki-Okamoto commitments we use the following parameters: $s = 552$ while t and l in the range proofs are 128 and 40 respectively.

Since genomic mutations are distributed at every $1,000$ bases [60] in average, pre-processing includes 3×10^6 mutations. The entire offline phase (performed by SL) takes about 4.2 hours on the aforementioned platform. Specifically, SL needs to compute one signature, one range proof commitment and one SHA2 commitment. In the online phase, T makes $r + 1$ signature verifications and 2 range proof verifications while Alice generates two range proofs. We measure the CPU time spent for the main cryptographic operations using our Java prototype.

Times for individual operations are as follows:

- Fujisaki-Okamoto commitment: 3.5ms,
- Zero-knowledge proof generation: 47.7ms
- Proof validation: 37ms
- SHA2 commitment: 0.3ms, including salt creation and its validation takes 0.1ms, which is, respectively, 12 and 370 faster than doing the same using Fujisaki-Okamoto commitments.

Note that salts can be alternatively (re-)generated using a seed and a PRNG, or HKDF, which would reduce the storage costs. We show results of using salts obtained from a secure random source, Java's SecureRandom class [50]. Verification cost performed by T scales

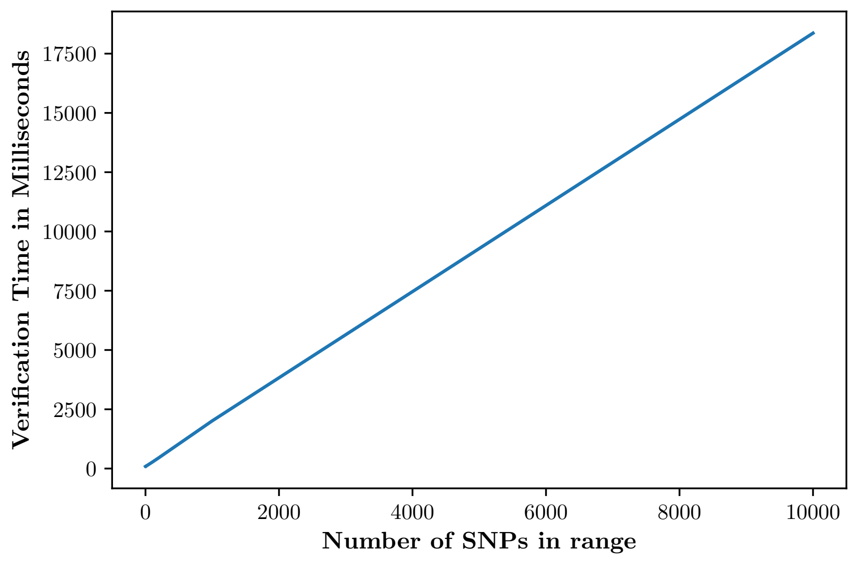


Figure 5.4: Verification times for T given number of SNPs in range

linearly with the number of SNPs in the queried range (as shown in Figure 5.4), and range proof verification cost is dominated by the signature verification cost.

5.9 General Setting

Although our main context is genomic range queries, the proposed technique can be applied in other settings. For example, suppose that a security forensics expert, after obtaining appropriate legal permission, needs to examine an ISP's log pertaining to a particular account's activities within a period of an attack. This setting is similar to genomic testing in terms of security and privacy requirements as well as sparsity of sensitive data. From the security perspective, the expert needs to ensure that logs are authentic, correct and complete. From the privacy perspective, the ISP does not want to reveal any other activity, in order to protect its customers' privacy. The relevant data could be sparsely distributed in a large data set, if the concerned account is not very active during period under review. Similar applications include queries of firewall logs. Our proposed technique can be adapted to address these security and privacy challenges. To this end, below we define a generalized notion of secure and private range query over sparse integers.

5.9.1 Secure & Private Range Queries over Sparse Integers

A sparse set of integers consists of integers sparsely and randomly chosen from a fixed range. We first define the density function for integer sets and then proceed to a more formal definition of sparse integer sets.

Definition 5.1. *Consider a set R, of integers in range $[a, b]$, and a subset S of R. The density of S is defined as $Density(S, R) := \frac{|S|}{|R|}$.*

Definition 5.2. *A subset S of R is a sparse integer set if $Density(S, R) < \epsilon$, for some small ϵ, e.g.,* 0.001.

Definition 5.3. *A range query involves a querier and a replier. Replier chooses S, a subset of R, prior to the query. A query range $[x, y]$ is chosen by querier. Both querier and replier know R.*

Definition 5.4. *Following Definition 5.2, a successful execution of a query with range $[x, y]$ in R reveals integers in $S \cap [x, y]$ to querier.*

A transcript between a replier (Alice) and a querier (Bob) of a range query over sparse integers is given in Transcript 1.

Transcript 1 Range Query over Sparse Integers

```
1: Alice randomly picks a sparse integer set $S$ of $k$ integers in $R$.
2: Bob queries a range $[x, y]$ in $R$.
3: Alice returns the set $S' := S \cap [x, y]$.
```

Goals

Here we define a list of goals that should be achieved by a secure and private solution to this problem:

1. **Authenticity.** A replier should not be able to modify S after it is chosen.
2. **Completeness.** The querier should be convinced that the integers returned by the replier are correct and no other integers lie in queried range other than the ones returned.
3. **Privacy.** The querier should not be able to learn any information on the integers outside of the queried range.

5.9.2 Construction

We now describe a concrete construction for secure and private range queries over sparse integers with respect to the goals stated in Section 5.9.1.

To establish trust between entities, we introduce a trusted offline authority which is denoted with *Auth*. Alice plays the role of the replier and Bob is the querier. Offline phase of the protocol is given in Transcript 2 and the online phase where Alice and Bob interact in Transcript 3.

Transcript 2 Offline Phase of Secure & Private Range Query over Sparse Integers

```
1: Alice randomly picks a sparse integer set $S$ of $k$ integers in $R$. $S$ is sorted in the increasing
   order. Two special integers $s_0$ and $s_{k+1}$ are introduced to mark the ends where $s_0 < a$
   and $s_{k+1} > b$.
2: Alice commits to each integer and creates $k + 1$ tuples linking the commitment for
   an integer to the commitment for the next integer. Each tuple is of the form
   $\{COM(s_i), COM(s_{i+1})\}$ where $s_i \in S$, $i \in [0, k]$. $COM$ is a hiding and binding com-
   mitment scheme that allows zero-knowledge range proofs over committed values.
3: Each tuple is signed by $Auth$. Here $Auth$ can make sure that Alice has chosen $k$ integers
   in $R$ and $s_i < s_{i+1}$ for each $i$.
```

Transcript 3 Online Phase of Secure & Private Range Query over Sparse Integers

```
1: Bob queries a range $[x, y]$ in $R$.
2: Alice reveals the integers in $S \cap [x, y]$ with relevant signed tuples. Bob checks the cor-
   rectness of commitments and verifies the signatures of each tuple.
3: Consider the tuple $\{COM(s_i), COM(s_{i+1})\}$ where $s_i$ is the largest integer smaller than
   $x$. Alice, in zero-knowledge, proves to Bob that $s_i < x$ using the commitment for $s_i$.
   Bob verifies this tuple's signature.
4: Consider the tuple $\{COM(s_j), COM(s_{j+1})\}$ where $s_{j+1}$ is the smallest integer larger
   than $y$. Alice, in zero-knowledge, proves to Bob that $s_{j+1} > y$ using the commitment for
   $s_{j+1}$. Bob verifies this tuple's signature.
```

Note that Step 3 or Step 4 is enough to prove that a tuple is out of range since integers are sorted before they are committed to. In addition, these two steps are enough to show that any tuple other than the ones returned are out of range. A visual representation is given in Figure 5.5.

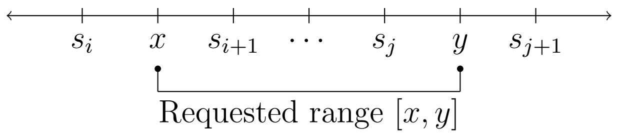


Figure 5.5: Proving $s_i < x$ (Step 3 in Transcript 3) states s_w where $w \leq i$ is out of range. Similarly, proving $s_{j+1} > y$ (Step 4 in Transcript 3) states s_w where $w \geq (j+1)$ is out of range.

5.9.3 Security & Privacy: Proof Sketch

Referring to each goal defined in Section 5.9.1, we show security and privacy of the proposed construction.

1. Goal 1 is satisfied by the binding property of the commitment scheme (COM) used along with signatures on the tuples containing each element. The binding property states that the committed value cannot be changed and no other value can be efficiently computed such that $COM(m_1) = COM(m_2)$ where m_1 is the committed value and $m_2 \neq m_1$. The signatures on the commitments ensure that none of the members of S can be changed (i.e., a change will be detectable since the signatures will be invalid).

2. The signatures on commitment tuples provide that any inclusion or exclusion to/from the set S will be detectable. Hence, Goal 2 is satisfied. Note that if each element in S is not linked to the next element and signatures are created on each element individually, then exclusion of tuples would be possible.

3. Consider the tuples out of the queried range $[x, y]$,
i.e., $\{COM(s_i), COM(s_{i+1})\}$ and $\{COM(s_j), COM(s_{j+1})\}$ in Figure 5.5. If s_i is less than the lower bound, zero-knowledgeness of the proof guarantees that any information on s_i (other than $s_i < x$) will not be revealed. In the same vein, if s_{j+1} is greater than the upper bound, only information revealed is $s_{j+1} > y$.

5.10 Related Work

Privacy of genetic material and tests has attracted many researchers and multiple solutions have been proposed that utilize cryptographic methods. Genomic security on the other hand has been in the background due to perceived lack of challenges – often mistakenly. In this section, we briefly go over the cryptographic privacy building blocks in the genomics domain and then discuss related techniques for achieving data authenticity and integrity in the context of range queries.

Genomic Privacy. PSI (Private Set Intersection) is a protocol that allows multiple parties to compute intersection of their sets without revealing the individual sets. This method has a few flavors that was adopted to provide genomic privacy. In [76], PSI-CA (CA for cardinality), which reveals only the size of the intersection over two set inputs, were used for paternity tests. This solution relied on the fact that sizes of DNA fragments that were cut by restricting enzymes could be used to determine paternal relations. In the same work, APSI (Authorized PSI) was used for personal medicine where FDA or another authority would authorize the markers that would be checked in the DNA. [97] used an Android smartphone to store encrypted genomic data and use PSI techniques to provide results of personal medicine, paternity and ancestry tests using the smartphone as the computation medium.

Homomorphic encryption is another tool for privacy preserving solutions for genomic tests. Using additive property of homomorphic encryption, [68] proposed methods for finding the edit distance between DNA sequences which was then improved in [69]. Oblivious automata was also used by [165] to perform matching and approximate searching. [131] used the homomorphic additive property to allow queries (such as SNP matches) on encrypted data and [74] used similar methods to compare DNA parts. These methods in [74] were more efficiently implemented by [93] using Elliptic Curve based Additively Homomorphic El-Gamal cryptosystem.

Range Query Security. Range query completeness was explored for outsourced databases where a user assigns the responsibility of handling queries on their data to a data publisher. Although this reduces the workload of a user, a malicious publisher can try to deceive any querier with incomplete or inauthentic query responses. [128] focused on minimizing privacy leakages on data attributes using data partitioning algorithms that are aware of the distribution of query ranges. [137] used Merkle hash and B^+ trees, and aggregated signatures to provide authenticity and integrity (with less strict privacy requirements compared to [152]) improve efficiency in the dynamic database case. [152] developed methods based on continual linking of elements (similar to [137]) and collision resistant hash functions to prevent such malicious actions.

Further cryptographic techniques to the range query problem incorporated range proofs. Earlier examples of range proofs were [140, 83, 89]. [140] used the bit-length of the committed value to prove that the number is in range $[0, 2^k - 1]$ where k is the number of bits in the committed value. Method used by [83] could only prove that the committed value lies in a wider range; $[-a, 2a]$ instead of $[0, a]$. [89] convinces a verifier that the committed value lies in a range where the expansion rate is 2^{t+l+1} where t and l are security parameters. [78] proposed two efficient protocols for proving that the committed value which is in $[a, b]$ lies in $[a - \theta, b + \theta]$ where $\theta = 2^{t+l+1}\sqrt{b-a}$ and t and l are security parameters. The second protocol is the one used in this work and has the expansion rate of 1. The commitments used in this work are Fujisaki-Okamoto commitments [118]. [87] proposed range proofs based on set membership protocols by extending the idea of using u-ary notation and proving that the secret $z \in [0, u^l - 1]$.

5.11 Conclusion

While genomic privacy has attracted much attention due to the dire consequences of possible leaks, insufficient attention is paid to genomic security, i.e., authentication and integrity of genomic data. This chapter proposes a cryptographic technique for efficient, secure and private genomic range queries. It ensures the *unforgeability* and *completeness* of genomic material needed by the tester from the user, and also protect user *privacy* as it leaks no extra information to the tester than necessary.

To achieve these properties, we used (1) zero-knowledge proofs to show that a committed value (the mutation position) is outside the range, and (2) signatures to prevent alterations of mutations and linkages among them, in order to preclude exclusions. We also abstracted away from genomics by defining a more general problem of secure and private range queries over sparse integers and discussed apply our approach.

Chapter 6

Conclusion & Future Work

This dissertation presented a number of privacy-centric attacks and defense mechanisms in multiple domains with varying technology penetration.

In Chapter 2, we presented the first thermal residue attack against passwords, namely `Thermanator`. This attack focused on recovering passwords from thermal residues left by human fingertips on plastic keyboards after password entry. We showed an attacker model with a post-factum window of opportunity to exploit password entry. This property greatly increases the threat level of such attacks since the likely adversaries (e.g., insider attackers) can reduce the possibility that they are going to get caught. `Thermanator` can recover password key-sets 30 seconds after password entry and subsets thereof 1 minute after entry.

In Chapter 3, we presented CACTI, a privacy-preserving CAPTCHA alternative that uses TEEs to generate *rate proofs*. These proofs are digitally signed by a TEE and allow clients to skip CAPTCHAs while keeping servers protected from attacks by bots. Rate proofs convince a server that the client is not acting in an abusive manner.

In Chapter 4, we presented a solution to an emerging problem with the introduction of privacy legislations. GDPR and CCPA requires servers to allow clients (also referred to as consumers) exercise certain rights on the data collected from them. These rights may be exercised a long time after the data were collected and include access, modify and delete operations. This introduces the problem of client authentication – or equivalently, verifying consumer requests, where clients need to prove that they are the same client that the data were collected from. For clients with accounts, the same authentication method used to access accounts can be used to fulfill such a requirement. However, the problem arises when clients do not have accounts on the websites. To solve this problem, in Chapter 4, we discussed a public key cryptography based data ownership mechanism, namely `VICEROY`. `VICEROY` uses `BIP32` to generate unlinkable $\mathcal{VCR}$ keys which are attached to Web sessions. Clients then can generate verifiable consumer requests ($\mathcal{VCR}$s) which are digitally signed requests using the $\mathcal{VCR}$ keys. A trusted device of a client only needs to store a small secret to use `VICEROY`.

In Chapter 5, we focused on the genomic domain. This domain has low technology penetration and therefore more traditional techniques are commonly used. In this chapter, we presented a privacy-preserving range query protocol that provided authentication and completeness of the genome using digital signatures and continual linking. The need for security for genomic data roots from an adversarial model where the attacker is the genome owner. The purpose of the attacker might be to alter their digitized genome in a way that it provides advantage to them for a genomic test. Such an attack could be used to change the result of a paternity test or could be to get access to prescription-only medicine.

Future Research Directions. Increasing availability of existing (such as thermal cameras) or emergence (such as TEEs) of new technologies paved the way for previously unexplored attacks and defenses. In this dissertation, we consider the lifecycle of a secret to choose domains where there are such technologies which could be used to protect secrets. To this end, we looked into side-channel attacks, TEE-provided privacy-preserving technologies and finally crypto-based less technologically advanced solutions. We presented the first thermal residue side-channel attack against passwords entered on external keyboards. `Thermanator` allowed attackers obtain partial key-presses in a password as late as 60 seconds after password entry. However, this side channel did not provide information on the order of key presses. This is due to lack of consistency in key-presses in terms of pressure, contact time and finger used to pressed the key – which changes the area that thermal residue appears. One future direction is to investigate eliminating these factors. Furthermore, thermal residue side-channel attacks can be combined with other attacks, such as keyboard acoustics, to improve attack success. Last but not least, emerging technologies are likely to enable more side-channel attacks in the future. For instance, Internet of Things (IoT) devices are equipped with different sensors which might reveal information on collected or entered data.

The increasing use of Web has created concerns regarding how data are collected and used by servers. Therefore, there is a need for privacy-preserving solutions that limit the collection of data. This requires replacing existing Web concepts/features with privacy-conscious ones. CACTI aims to achieve this by substituting CAPTCHAs with rate proofs. Future directions include creating similar privacy-preserving alternatives. Futhermore, depending on the needs for existing legislation and the problems they pose, one can consider implementing such solutions in a privacy-preserving way or providing transparency to clients in terms of server actions. Remote attested TEEs can be a major tool for achieving this goal.

For the genomics domain, we focused on the SNP representation of a genome to provide secure and privacy-preserving range queries. Different representations of DNA can offer different challenges which need to be dealt with while providing security, privacy and usability. Furthermore, different genomic tests may require different computations on genomic data and security and privacy challenges may be different for such cases.

Bibliography

[1] Americans and privacy: Concerned, confused and feeling lack of control over their personal information. https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-pe Accessed: 2021-07-06.

[2] An12326 secure gpio and usage. https://www.nxp.com/docs/en/application-note/AN12326.pdf. Accessed: 2021-07-08.

[3] Android keystore system. https://developer.android.com/training/articles/keystore. Accessed: 2021-04-20.

[4] Anonymous submission ccs'21 b. https://www.dropbox.com/sh/866gwbud2x4fuaq/AAD5FJnmKuL3at5O1Zy8mZyFa?dl=0.

[5] AntiCAPTCHA. https://anti-captcha.com/mainpage, [Online] Accessed: 2020-05-22.

[6] Big data analytics. https://www.ibm.com/analytics/hadoop/big-data-analytics. Accessed: 2021-07-06.

[7] Bip 0032. https://en.bitcoin.it/wiki/BIP_0032. Accessed: 2021-04-05.

[8] Bip32. https://github.com/bitcoinjs/bip32. Accessed: 2021-04-01.

[9] bip32. https://github.com/sammyne/bip32. Accessed: 2021-04-20.

[10] Bip32. https://github.com/NovaCrypto/BIP32. Accessed: 2021-04-20.

[11] Bip32 implementation using python. https://github.com/ismailakkila/bip32. Accessed: 2021-04-20.

[12] Browserify. https://github.com/browserify/browserify. Accessed: 2021-04-01.

[13] California consumer privacy act. https://www.oag.ca.gov/privacy/ccpa. Accessed: 2021-03-18.

[14] Chrome Native Messaging Protocol. https://developer.chrome.com/extensions/nativeMessaging#native-messaging-host-protocol, [Online] Accessed: 2020-02-09.

[15] Chrome Notifications. `https://developer.chrome.com/apps/notifications`, [Online] Accessed: 2020-02-14.

[16] Cloudflare Rate Limiting. `https://www.cloudflare.com/rate-limiting/`, [Online] Accessed: 2020-05-19.

[17] Cookie matching. `https://developers.google.com/authorized-buyers/rtb/cookie-guide`. Accessed: 2021-04-26.

[18] EPID SDK. `https://github.com/Intel-EPID-SDK/epid-sdk`, [Online] Accessed: 2020-02-14.

[19] Express. `https://expressjs.com/`. Accessed: 2021-05-01.

[20] General data protection regulation. `https://gdpr-info.eu/`. Accessed: 2021-03-18.

[21] Global visual hacking experimental study: Analysis. `https://multimedia.3m.com/mws/media/1254232O/global-visual-hacking-experiment-study-summary.pdf`. Accessed: 2021-07-07.

[22] Google analytics. `https://developers.google.com/analytics/devguides/collection/analyticsjs/cookie-usage`. Accessed: 2021-02-18.

[23] Google Chrome. `https://www.google.com/chrome/`, [Online] Accessed: 2020-02-11.

[24] hCaptcha. `https://www.hcaptcha.com/`, [Online] Accessed: 2020-05-21.

[25] Hierarchical deterministic wallets. `https://github.com/marctrem/BIP32c`. Accessed: 2021-04-20.

[26] How do geneticists indicate the location of a gene? `https://ghr.nlm.nih.gov/primer/howgeneswork/genelocation`. Accessed: 2019-07-02.

[27] Insider threats examples: 17 real examples of insider threats. `https://www.tessian.com/blog/insider-threats-types-and-real-world-examples/`. Accessed: 2021-07-09.

[28] Intel Dynamic Application Loader Developer Guide: Monotonic Counters. `https://software.intel.com/en-us/dal-developer-guide-features-monotonic-counters`, [Online] Accessed: 2020-02-05.

[29] Intel Integrated Performance Primitives Cryptography. `https://github.com/intel/ipp-crypto`, [Online] Accessed: 2020-05-28.

[30] Intel NUC Kit NUC7PJYH. `https://ark.intel.com/content/www/us/en/ark/products/126137/intel-nuc-kit-nuc7pjyh.html`, [Online] Accessed: 2020-02-11.

[31] Intel Pentium Processor G4400. `https://ark.intel.com/content/www/us/en/ark/products/88179/intel-pentium-processor-g4400-3m-cache-3-30-ghz.html`, [Online] Accessed: 2020-05-19.

[32] Intel SGX DCAP. https://01.org/intel-softwareguard-extensions/downloads/intel-sgx-dcap-1.6-release.

[33] Intel® software guard extensions. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html. Accessed: 2021-04-20.

[34] JSMN JSON Parser. https://github.com/zserge/jsmn, [Online] Accessed: 2020-02-13.

[35] Ledger. https://www.ledger.com/. Accessed: 2021-04-20.

[36] The legion of the bouncy castle. https://www.bouncycastle.org/. Accessed: 2019-02-12.

[37] Mbed TLS. https://github.com/ARMmbed/mbedtls, [Online] Accessed: 2020-02-14.

[38] MITRE ATT&CK: Steal Web Session Cookie. https://attack.mitre.org/techniques/T1539/.

[39] Moving from reCAPTCHA to hCaptcha. https://blog.cloudflare.com/moving-from-recaptcha-to-hcaptcha/, [Online] Accessed: 2020-05-19.

[40] Native Messaging. https://developer.chrome.com/extensions/nativeMessaging, [Online] Accessed: 2020-02-13.

[41] native-messaging. https://www.npmjs.com/package/native-messaging. Accessed: 2021-05-01.

[42] Native messaging protocol. https://developer.chrome.com/docs/apps/nativeMessaging/#native-messaging-host-protocol. Accessed: 2021-05-01.

[43] Node.js. https://nodejs.org/en/. Accessed: 2021-02-18.

[44] Npm uuid. https://www.npmjs.com/package/uuid. Accessed: 2021-02-18.

[45] Open Enclave SDK. https://openenclave.io/sdk/, [Online] Accessed: 2020-02-14.

[46] Package captcha. https://github.com/dchest/captcha, [Online] Accessed: 2020-05-21.

[47] reCAPTCHA. https://www.google.com/recaptcha/intro/v3.html, [Online] Accessed: 2020-02-05.

[48] reCAPTCHA v2. https://developers.google.com/recaptcha/docs/display, [Online] Accessed: 2020-02-13.

[49] runtime.Port. https://developer.chrome.com/extensions/runtime#type-Port, [Online] Accessed: 2020-02-12.

[50] Securerandom (java platform se 8). https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html. Accessed: 2019-02-13.

[51] Sgx-hardware list. https://github.com/ayeks/SGX-hardware. Accessed: 2021-07-10.

[52] Snp. https://www.nature.com/scitable/definition/single-nucleotide-polymorphism-snp-295. Accessed: 2019-04-17.

[53] Storing keys in the secure enclave. https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave. Accessed: 2021-04-20.

[54] svg captcha. https://github.com/produck/svg-captcha, [Online] Accessed: 2020-05-21.

[55] tiny-secp256k1. https://www.npmjs.com/package/tiny-secp256k1?activeTab=versions. Accessed: 2021-05-01.

[56] The token binding protocol version 1.0. https://tools.ietf.org/html/rfc8471. Accessed: 2021-04-26.

[57] Top 10 Captcha Solving Services Compared. https://prowebscraper.com/blog/top-10-captcha-solving-services-compared/, [Online] Accessed: 2020-05-22.

[58] Tor Project. https://www.torproject.org/.

[59] Using Privacy Pass with Cloudflare. https://support.cloudflare.com/hc/en-us/articles/115001992652-Using-Privacy-Pass-with-Cloudflare, [Online] Accessed: 2020-06-01.

[60] What are single nucleotide polymorphisms (snps)? https://ghr.nlm.nih.gov/primer/genomicresearch/snp. Accessed: 2019-02-05.

[61] What is big data? https://www.oracle.com/big-data/what-is-big-data/. Accessed: 2021-07-06.

[62] What Technology Change Enables 1 Terabyte (TB) Enclave Page Cache (EPC) size in 3rd Generation Intel® Xeon® Scalable Processor Platforms? https://www.intel.com/content/www/us/en/support/articles/000059614/software/intel-security-products.html.

[63] Zero-knowledge proofs. https://github.com/ing-bank/zkproofs. Accessed: 2019-02-12.

[64] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689, 2014.

[65] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13, page 7. ACM New York, NY, USA, 2013.

[66] ARM Holdings. ARM Security Technology, Building a Secure System using TrustZone Technology, 2009.

[67] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 3–11. IEEE, 2004.

[68] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 39–44. ACM, 2003.

[69] M. J. Atallah and J. Li. Secure outsourcing of sequence comparisons. *International Journal of Information Security*, 4(4):277–287, 2005.

[70] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 255–270, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[71] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In *International conference on security and cryptography for networks*, pages 111–125. Springer, 2006.

[72] P.-L. Aublin, F. Kelbert, D. O'Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch. Libseal: Revealing service integrity violations using trusted execution. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, 2018.

[73] J. Ausloos and P. Dewitte. Shattering one-way mirrors. data subject access rights in practice. *Data Subject Access Rights in Practice (January 20, 2018). International Data Privacy Law*, 8(1):4–28, 2018.

[74] E. Ayday, J. L. Raisaro, and J.-P. Hubaux. Privacy-enhancing technologies for medical tests using genomic data. Technical report, 2012.

[75] E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 95–106. ACM, 2013.

[76] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 691–702. ACM, 2011.

[77] C. Boniface, I. Fouad, N. Bielova, C. Lauradoux, and C. Santos. Security analysis of subject access request procedures. In *Annual Privacy Forum*, pages 182–209. Springer, 2019.

[78] F. Boudot. Efficient proofs that a committed number lies in an interval. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 431–444. Springer, 2000.

[79] T. Bradley, X. Ding, and G. Tsudik. Genomic security (lest we forget). *IEEE Security & Privacy*, 15(5):38–46, 2017.

[80] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, 2004.

[81] E. Brickell, L. Chen, and J. Li. A static diffie-hellman attack on several direct anonymous attestation schemes. In *International Conference on Trusted Systems*, pages 95–111. Springer, 2012.

[82] E. Brickell and J. Li. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, page 21–30, New York, NY, USA, 2007. Association for Computing Machinery.

[83] E. F. Brickell, D. Chaum, I. B. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 156–166. Springer, 1987.

[84] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving CAPTCHAs? A large scale evaluation. In *2010 IEEE symposium on security and privacy*, pages 399–413. IEEE, 2010.

[85] A. Burton. The range and variability of the blood flow in the human fingers and the vasomotor regulation of body temperature. *American Journal of Physiology-Legacy Content*, 127(3):437–453, 1939.

[86] M. Cagnazzo, T. Holz, and N. Pohlmann. Gdpirated – stealing personal information on- and offline. In *European Symposium on Research in Computer Security*, pages 367–386. Springer, 2019.

[87] J. Camenisch, R. Chaabouni, et al. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.

[88] C. Castelluccia, L. Olejnik, and T. Minh-Dung. Selling Off Privacy at Auction. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, United States, Nov. 2014. ISOC.

[89] A. Chan, Y. Frankel, and Y. Tsiounis. Easy come-easy go divisible cash. updated version with corrections. Technical report, GTE Tech. Rep, 1998.

[90] C.-M. Cheng, H. Kung, and K.-S. Tan. Use of spectral analysis in defense against DoS attacks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 3, pages 2143–2148. IEEE, 2002.

[91] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[92] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, and E. Weippl. Measuring cookies and web privacy in a post-gdpr world. In *International Conference on Passive and Active Network Measurement*, pages 258–270. Springer, 2019.

[93] G. Danezis and E. De Cristofaro. Fast and private genomic testing for disease susceptibility. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 31–34. ACM, 2014.

[94] J. Danisevskis. Android Protected Confirmation: Taking transaction security to the next level. `https://developer.android.com/training/articles/security-android-protected-confirmation`, [Online] Accessed: 2020-02-05.

[95] R. Datta, J. Li, and J. Z. Wang. IMAGINATION: a robust image-based CAPTCHA generation system. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 331–334, 2005.

[96] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018(3):164–180, 2018.

[97] E. De Cristofaro, S. Faber, P. Gasti, and G. Tsudik. Genodroid: are privacy-preserving genomic tests ready for prime time? In *Proceedings of the 2012 ACM workshop on Privacy in the electronic society*, pages 97–108. ACM, 2012.

[98] E. De Cristofaro, S. Faber, and G. Tsudik. Secure genomic testing with size-and position-hiding private substring matching. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 107–118. ACM, 2013.

[99] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet 1. *Journal of Computer Security*, 11(3):291–314, 2003.

[100] M. Di Martino, P. Robyns, W. Weyts, P. Quax, W. Lamotte, and K. Andries. Personal information leakage by abusing the {GDPR}'right of access'. In *Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019)*, 2019.

[101] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 317–331, 2012.

[102] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.

[103] X. Ding and G. Tsudik. Initializing trust in smart devices via presence attestation. *Computer Communications*, 131:35 – 38, 2018.

[104] S. Eskandarian, J. Cogan, S. Birnbaum, P. C. W. Brandon, D. Franke, F. Fraser, G. Garcia, E. Gong, H. T. Nguyen, T. K. Sethi, V. Subbiah, M. Backes, G. Pellegrino, and D. Boneh. Fidelius: Protecting user secrets from compromised browsers. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 264–280, 2019.

[105] A. et al. Smudge attacks on smartphone touch screens. *Woot*, 10:1–7, 2010.

[106] A. et al. A pilot study on the security of pattern screen-lock methods and soft side channel attacks. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 1–6. ACM, 2013.

[107] A. et al. Stay cool! understanding thermal attacks on mobile-based user authentication. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3751–3763. ACM, 2017.

[108] D. et al. Comparison of human skin opto-thermal response to near-infrared and visible laser irradiations: a theoretical investigation. *Physics in Medicine & Biology*, 49(21):4861, 2004.

[109] M. et al. Heat of the moment: Characterizing the efficacy of thermal camera-based attacks. In *Proceedings of the 5th USENIX conference on Offensive technologies*, pages 6–6. USENIX Association, 2011.

[110] M. et al. (sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 551–562. ACM, 2011.

[111] P. et al. Heat capacity of poly (butylene terephthalate). *Journal of Polymer Science Part B: Polymer Physics*, 42(23):4401–4411, 2004.

[112] P. et al. Diminutive digits discern delicate details: fingertip size and the sex difference in tactile spatial acuity. *Journal of Neuroscience*, 29(50):15756–15761, 2009.

[113] S. et al. Study of potential attacks on rubber pin pads based on mobile thermal imaging.

[114] S. et al. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, volume 2001, 2001.

[115] Z. et al. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):3, 2009.

[116] Y. Feng, Y. Zhang, C. Ying, D. Wang, and C. Du. Nanopore-based fourth-generation dna sequencing technology. *Genomics, proteomics & bioinformatics*, 13(1):4–16, 2015.

[117] C. A. Fidas, A. G. Voyiatzis, and N. M. Avouris. On the necessity of user-friendly CAPTCHA. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2623–2626, 2011.

[118] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Annual International Cryptology Conference*, pages 16–30. Springer, 1997.

[119] H. Gao, W. Wang, and Y. Fan. Divide and conquer: an efficient attack on Yahoo! CAPTCHA. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 9–16. IEEE, 2012.

[120] C. A. General. California consumer privacy act regulations. `https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/oal-sub-final-text-of-regs.pdf`, 2020.

[121] P. Golle. Machine learning attacks against the Asirra CAPTCHA. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542, 2008.

[122] R. Gonzalez, L. Jiang, M. Ahmed, M. Marciel, R. Cuevas, H. Metwalley, and S. Niccolini. The cookie recipe: Untangling the use of cookies in the wild. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9, 2017.

[123] R. Gossweiler, M. Kamvar, and S. Baluja. What's up CAPTCHA? A CAPTCHA based on image orientation. In *Proceedings of the 18th international conference on World wide web*, pages 841–850, 2009.

[124] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks. In *NSDI*, volume 9, pages 307–320, 2009.

[125] S. J. Heerema and C. Dekker. Graphene nanodevices for dna sequencing. *Nature nanotechnology*, 11(2):127, 2016.

[126] D. Herrmann and J. Lindemann. Obtaining personal data and asking for erasure: Do app vendors and website owners honour your privacy rights? *arXiv preprint arXiv:1602.01804*, 2016.

[127] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA*, 11(10.1145):2487726–2488370, 2013.

[128] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 720–731. VLDB Endowment, 2004.

[129] S. Jordan. A comparison of notice and consent requirements under the gdpr, the ccpa/cpra, and the fcc broadband privacy order, 2021.

[130] S. Jordan, S. Narasimhan, and J. Hong. Collection, use, and sharing of personal information, 2021.

[131] M. Kantarcioglu, W. Jiang, Y. Liu, and B. Malin. A cryptographic approach to securely share and query genomic sequences. *IEEE Transactions on information technology in biomedicine*, 12(5):606–617, 2008.

[132] C. A. Kerrache, N. Lagraa, C. T. Calafate, and A. Lakas. TFDD: A trust-based framework for reliable data delivery and DoS defense in VANETs. *Vehicular Communications*, 9:254–267, 2017.

[133] H. Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[134] J. L. Kröger. Subject access request response data - 105 ios and 120 android apps, 2020.

[135] C. Legislature. California consumer privacy act of 2018 (as amended by the california privacy rights act of 2020). `https://www.oag.ca.gov/privacy/ccpa`, 2020.

[136] A. Leung, L. Chen, and C. J. Mitchell. On a possible privacy flaw in direct anonymous attestation (DAA). In *International Conference on Trusted Computing*, pages 179–190. Springer, 2008.

[137] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 121–132. ACM, 2006.

[138] W. Li, S. Luo, Z. Sun, Y. Xia, L. Lu, H. Chen, B. Zang, and H. Guan. Vbutton: Practical attestation of user-driven operations in mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 28–40, 2018.

[139] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 195–206, 2008.

[140] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *International Workshop on Public Key Cryptography*, pages 60–71. Springer, 1998.

[141] E. R. Mardis. A decade's perspective on dna sequencing technology. *Nature*, 470(7333):198, 2011.

[142] S. Mare, M. Baker, and J. Gummeson. A study of authentication in daily life. In *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*, pages 189–206, 2016.

[143] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy*, pages 413–427, 2012.

[144] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

[145] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[146] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: CAPTCHAs-Understanding CAPTCHA-Solving Services in an Economic Context. In *USENIX Security Symposium*, volume 10, page 3, 2010.

[147] Y. Nakatsuka, E. Ozturk, A. Paverd, and G. Tsudik. CACTI: Captcha avoidance via client-side TEE integration. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, Aug. 2021.

[148] M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 48(1):6, 2015.

[149] J. Noyes. The qwerty keyboard: A review. *International Journal of Man-Machine Studies*, 18(3):265–281, 1983.

[150] Occupational Safety and Health Administration and others. Osha technical manual. *Section VIII*, 1999.

[151] X. Ouyang, B. Tian, Q. Li, J.-y. Zhang, Z.-M. Hu, and Y. Xin. A novel framework of defense system against DoS attacks in wireless sensor networks. In *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–5. IEEE, 2011.

[152] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM, 2005.

[153] E. Parliament and Council. General data protection regulation, regulation (eu) 2016/679 (as amended). `https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04`, 2016.

[154] J. Pavur and C. Knerr. Gdparrrrr: Using privacy laws to steal identities. *arXiv preprint arXiv:1912.00731*, 2019.

[155] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)*, 39(1):3–es, 2007.

[156] P. Perlegos. *DoS defense in structured peer-to-peer networks.* Computer Science Division, University of California, 2004.

[157] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 264–278. IEEE, 2018.

[158] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet, N. Kulatova, T. Ramananandro, A. Rastogi, N. Swamy, C. Wintersteiger, and S. Zanella-Beguelin. EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider. Cryptology ePrint Archive, Report 2019/757, 2019.

[159] C. Rudolph. Covert identity information in direct anonymous attestation (DAA). In *IFIP International Information Security Conference*, pages 443–448. Springer, 2007.

[160] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[161] M. Sanghavi and S. Doshi. Progressive captcha, Apr. 30 2009. US Patent App. 11/929,716.

[162] J. Sauro. Estimating productivity: composite operators for keystroke level modeling. In *International Conference on Human-Computer Interaction*, pages 352–361. Springer, 2009.

[163] A. Shamir and N. Van Someren. Playing hide and seek with stored keys. In *International conference on financial cryptography*, pages 118–124. Springer, 1999.

[164] T. I. H. C. Study et al. The major genetic determinants of hiv-1 control affect hla class i peptide presentation. *Science (New York, NY)*, 330(6010):1551, 2010.

[165] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528. ACM, 2007.

[166] T. Urban, M. Degeling, T. Holz, and N. Pohlmann. " your hashed ip address: Ubuntu." perspectives on transparency tools for online advertising. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 702–717, 2019.

[167] T. Urban, D. Tatang, M. Degeling, T. Holz, and N. Pohlmann. A study on subject data access in online advertising after the gdpr. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 61–79. Springer, 2019.

[168] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *International conference on the theory and applications of cryptographic techniques*, pages 294–311. Springer, 2003.

[169] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In E. Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 294–311, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[170] C. Wang, S. T. Jan, H. Hu, D. Bossart, and G. Wang. The next domino to fall: Empirical analysis of user passwords across online services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 196–203, 2018.

[171] J. Z. Wang, R. Datta, and J. Li. Image-based CAPTCHA generation system, Apr. 19 2011. US Patent 7,929,805.

[172] S. Weiser and M. Werner. SGXIO: Generic trusted I/O path for Intel SGX. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 261–268, 2017.

[173] W. Wodo and L. Hanzlik. Thermal imaging attacks on keypad security systems. In *SECRYPT*, pages 458–464, 2016.

[174] J. Yan and A. S. 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, 2008.

[175] M. Zalewski. Cracking safes with thermal imaging. `http://lcamtuf.coredump.cx/tsafe/`, 2005. Accessed: 2018-04-02.

[176] Z. Zhang, X. Ding, G. Tsudik, J. Cui, and Z. Li. Presence Attestation: The Missing Link in Dynamic Trust Bootstrapping. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 89–102, New York, NY, USA, 2017. Association for Computing Machinery.

[177] Z. Zhang, X. Ding, G. Tsudik, J. Cui, and Z. Li. Presence attestation: The missing link in dynamic trust bootstrapping. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 89–102, 2017.

[178] Z. Zhang, X. Ding, G. Tsudik, J. Cui, and Z. Li. Presence attestation: The missing link in dynamic trust bootstrapping. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2017.

[179] S. Zimmeck and K. Alicki. Standardizing and implementing do not sell. In *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, pages 15–20, 2020.