5-2024

A Quantitative Analysis of Security Keys and Commit Signing on Github

Parker N. Collier
pcollie4@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Parker N. Collier entitled "A Quantitative Analysis of Security Keys and Commit Signing on Github." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Scott I Routi, Major Professor

We have read this thesis and recommend its acceptance:

Doowon Kim, Audris Mockus

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:
I am submitting herewith a thesis written by Parker N. Collier entitled "A Quantitative Analysis of Security Keys and Commit Signing on Github." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Scott Routi, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Scott Routi

Dr. Doowon Kim

Dr. Audris Mockus

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a thesis written by Parker N. Collier entitled "A Quantitative Analysis of Security Keys and Commit Signing on Github." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Scott Routi, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Scott Routi

Dr. Doowon Kim

Dr. Audris Mockus

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

A Quantitative Analysis of Security Keys and Commit Signing on Github

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Parker N. Collier

May 2024

For my parents, Daniel and Grey Collier, whose support got me this far.

Acknowledgments

I would like to express my gratitude to my advisor Dr. Scott Routi, and the rest of my committee, Dr. Doowon Kim, and Dr. Audris Mockus, whose academic help has been invaluable during my time here at UTK. I would also like to acknowledge my academic advisors, Joanna Rathbone and Mike Taylor. Without their adept guidance I would not have thought to pursue this degree. Lastly I would like to thank my friend, Matt Wyatt, for always encouraging me to learn more.

Keep moving forward.

Abstract

This thesis analyzes the use and impact of security and signing keys on Github, the foremost public code development platform. These keys are used for developer authentication and code commit signing, but little research has been done on the usage of these keys. We set out to collect every available key associated with a Github user and performed quantitative analysis on the gathered data. Our data was gathered using Github's publicly available REST and GraphQl API's. We found that very few users create keys for signing commits, and there are a number of keys on the database that could be considered weak by modern standards. Personal keys for user identification is not widely accepted. A better understanding of how developers interact with these systems is needed to develop software that is both usable and secure.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Git is a prominent code versioning tool used by developers to create software. Code versioning is used to keep track of changes from multiple users in a single repository. Git tracks changes rather than versions. These changes in are pushed, or uploaded, by developers in the form of a commit. Virtually all collaborative development projects utilize some form of code versioning. One of Git's primary features is allowing the majority of development to be done locally on a developer's personal machine. Local development reduces network overhead and allows for developers to work from any location. Git based systems are the most widely used method for code development and collaboration. There are various implementations of this system such as Bitbucket, Github, and Gitlab.

Github is the most used git implementation [23]. There are more than 300 million public repositories on Github. Many of these are widely viewed and depended upon in production systems. Github also acts as a platform for code deployment. Any public repository can be pulled and executed by any Github user. Repositories also contain an optional releases section, this allows developers to offer statically generated executable files for download. Github offers a service called Actions that allows for automatic deployment to online services that support web-hooks like websites or

cloud servers. This means a compromised repository could spread malicious software to users, in some cases automatically.

Github is designed for collaboration. Any user can fork, or copy, a public Github repository to their local machine edit the code, then attempt to merge their changes into the code base. It is up to the owners of repositories to vet these merge requests by checking that the code itself is sound or that the merge is coming from a trusted developer. Admins of the original repository have to merge, or approve, the changes made to the repository. While this makes open source programming and cooperation convenient, it also creates a significant vector for threat actors to inject malicious code into a development pipeline.

Any machine can be used to upload code to a repository. Github's security is dependent on strong access control and this is achieved via cryptographic key authentication. Users generate a key that is stored securely on their local machine. Github is able to use this key to verify the users access rights to a given repository. This removes the need for a password when pushing or pulling code. As of 2021, Github requires the usage of these asymmetric SSH keys where previously it was optional. The strength and effective usage of these keys are critical. They effectively act as passwords and have the same security implications if leaked or broken. A malicious actor with access to a private key could arbitrarily upload code to any repository the original key owner has access to. Github has made changes to its security policy in recent years, but there is no research on the effective strength and usage of these keys.

Another concern with the security of Github's development pipeline is developer identification. Github pushes and commits are linked to an email by default, however, there is no verification system in place to ensure these emails are valid or connected to a user. This creates an opportunity for user impersonation. Authentication keys somewhat mitigate this risk by ensuring that a user has access to a repository. They do not, however, prevent malicious actors from attempting to push dishonest commits. Developers within a repository could push code under a false name. False commits

could be used to push malicious code, and put the blame on an innocent developer within the repository. Since these commits are made locally, generation of false commits can be automated at scale with relative ease. Commit meta data, like timestamps, can also be manipulated. This allows for the creation of false commit history on Github, which can be used to give accounts false credibility.

Github provides a solution to this issue of identification and commit integrity. Commits can be signed digitally via an asymmetric cryptographic key. If a developer has a signing key, they would be protected from impersonation by other users.

We have not been able to find research on the usage and effectiveness of security keys on Github. We set out to answer a few questions about Github key security and commit signing.

Research Question 1 What are the quality of authentication keys? General information on keys like what algorithms are most commonly used, security strength of keys, and how many keys users have, helps to understand the security implications of the key system.

Research Question 2 What is the prevalence of and quality of signing keys. Signing is optional, so we need to make a census of how many users choose to engage with the system, and understand why they do it.

Research Question 3 How often are commits signed on Github? A user having a signing key does not necessarily indicate they use the key frequently or at all. The number of signed commits is more indicative of singing usage than the number of users who have a key.

We performed the first wide scale qualitative analysis of cryptographic keys on Github. We constructed a dataset of all Github users and their available public keys. These keys can be accessed via the Github Rest API or Githubs GraphQl implementation. GraphQl is a newer tool that allows requests for multiple users keys at a time, but is not fully implemented with the API and currently can only retrieve

authentication keys. For this reason we use Rest API to gather commit keys and GraphQl to gather authentication keys.

There is a variety of key quality for signing keys. The majority of keys use popular strong algorithms like RSA and ed25519 with standard key lengths. Github recently changed their policy to phase out older SSH algorithms, but users are still allowed to use a swath of key algorithms and key lengths. We have found weak keys are still in use after the security update and weak algorithms are still being used to generate new keys. We found that key usage is low in the case of commit singing. Less less than 2% of users have any form of singing key. In addition to low usage, signing key security is lower than login key security. The recent security update did not apply to all commit signing keys. We have found a significant number of keys that would not be allowed to be used for authentication in use for commit signing. This indicates a lower priority on commit signing as a whole.

Chapter 2

Related-Works

2.1 Background

Understanding the implications of key data on Github requires some context on the two major systems in use; git commit systems and cryptographic key systems. Cryptographic algorithms change frequently. Security standards change as new algorithms are developed and old algorithms become insufficient. Knowing the history of these algorithms and systems helps to understand current practice.

Updating code files and projects in Github is done via commits, which track the changes made to a project. Sending a commit from a users local machine to a shared github repository is called a push, and requires some form of verification. The fastest and most convenient form of authentication is to use an ssh login key where the public key is held by Github and the private key is stored locally on the users machine. By default these commits are attributed to an account via their locally attached email, however there is no validation ensuring the actual identity of the developer. This provides a significant vector for someone to pose as a different user when contributing to a Github repository. Users have the option to sign these commits using an asymmetric signing key system like GPG or SSH. These keys are similarly user generated and maintained.

2.1.1 Cryptographic Keys

Github supports the use of security keys for two main purposes: login authentication and commit signing. Users generate and maintain their own asymmetric keys and upload their public keys to the platform. These keys are publicly available to all users via Github's API. These keys can be in SSH or GPG format, a variety of cipher suites and key lengths are supported. There is some risk to allowing for a large number of cipher suites as shown with various TLS attacks such as FREAK[30]. Allowing all users to see what keys are being used introduces the risk of threat actors searching for users with less secure configurations.

Asymmetric cryptographic keys are generated in a pair format, with one key being public and one key being private. The private key is securely maintained by the generator while the public key is given out to others. When communicating, information is encrypted with the private key and decrypted with the public key. This allows for a one way communication system that verifies the identity of the private key holder. In Githubs model users generate a key pair, upload the public key to their Github account, and maintain their private key locally.

We are looking at the usage of these keys for the sake of digital signatures and verification. A digital signature algorithm uses a private key and the message content of the item it is signing to create a signature. Then the public key is used in a signature decoding algorithm. This allows for the public key to be passed along with the message and signature itself. Then any person with the public key and message can ensure the message was digitally signed by the owner of the corresponding private key.

Security standards change frequently as technology improves and new systems are created. Cryptographic algorithms change the most, new algorithms are introduced every few years, and old algorithms are found to contain vulnerabilities. It is worth knowing the difference in these algorithms to understand the implications our key analysis.

Four asymmetric algorithms are found on Github: RSA, ECDSA, Ed25519, and DSA.

RSA (Rivest–Shamir–Adleman) Named after the first surnames of its three developers, it was first described in 1977. Brute forcing RSA keys requires solving a large prime number factoring problem. According to the National Institute of Standards Technology (NIST) the current recommended key size for RSA is 2048 bits long. RSA is among the slower asymmetric key algorithms, and has large key sizes. RSA is still one of the most widely used asymmetric algorithms in the world and represents a majority of the keys on Github.

ECDSA (Elliptic Curve Digital Signature Algorithm) ECDSA is built on Elliptic Curve Cryptography, first described in 1985. ECDSA itself was accepted by IEEE in 1999. Elliptic curves are a geometric algorithm with unique properties making it ideal for key generation. Elliptic curve algorithms reduce to a discrete log problem, making them very secure. ECDSA keys are much smaller than RSA. 256 bit ECDSA keys are equivalent to 2048 bit RSA keys. Elliptic Curve algorithms are also up to 1000 times faster than RSA algorithms. The major drawback of ECDSA is its dependence on strong random numbers for the nonce. A nonce is a type of seed value used in many forms of cryptography, ECDSA needs a new nonce every time a signature is generated. If new random numbers are not selected every time a signature is made, the key can be broken. Sony had such an error on their server network in 2010 that led to a large security breach [12].

EdDSA (Edwards-curve Digital Signature Algorithm) EdDSA is another elliptic curve variant developed in 2011. EdDSA is similar to ECDSA in many regards. It has 256 bit keys for 128 bit security. The largest difference EdDSA removed the need for random number generation on every signature. Instead the nonce is generated hashing the private key and message. Besides removing a potential vulnerability in the previous elliptic curve implementations, removing

random number generation also greatly reduces the effective runtime of the algorithm. Ed25519 is the most popular EdDSA algorithm and has become the standard on a variety of platforms including OpenSSH and GnuPG.

DSA Digital Signature Algorithm DSA is a system designed for digital signatures. It was first proposed in 1991 and was accepted into FIPS in 1994. DSA derives its security from a mixture of modular exponentiation and discrete logarithm problems. DSA and RSA are similar in performance - RSA is better at encryption, while DSA is better at decryption. However both applications are often used in tandem so the difference becomes negligible. While DSA is still a secure algorithm, it has a few qualities that make it more sensitive than other solutions. DSA requires the generation of random value for signatures. If these values are predictable, reused, or partially leaked, the signature can be broken. These concerns have resulted in some developers moving away from the algorithm. For instance Microsoft, the current owner of Github, has declared this system weak and does not recommend its use. Github has partially phased out the use of this algorithm for key generation.

There are two software platforms used for key signing on Github SSH and GPG. Both platforms support every cipher suite found on Github.

SSH Secure Shell Hosting SSH is a network tunneling tool developed in 1995 and primarily used for secure remote access to an outside network. It primary functionality is focused on connecting to remote machines. SSH security keys are used for authentication. Git uses this system to authenticate its users via cryptographic keys. Git does also allow the use of these keys for signing commits.

GPG Gnu Privacy Guard GPG is an encryption software program primarily used for digital signatures. It was developed in 1999 and remains one of the most common solutions for digital signatures. GPG is only capable of signing commits

on GitHub and cannot be used for remote authentication. GitHubs new key standards did not include keys generated for GPG, which still allows for the use of DSA algorithms for key generation.

2.2 Related Works

Github and general git systems are a popular topic of research. The majority of work in the field is focused on the usability of git commit systems and finding ways to improve them. One system often studied is merge conflict resolution. Gleph Ghiotto's paper *On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by GitHub* [10] as well as Gustavo Vale's work *Challenges of Resolving Merge Conflicts: A Mining and Survey Study* [29] are focused on the feasibility of automatic merge resolution. They both performed large scale data collection of GitHub merges and analyzed them for the sake of classification. Gustovo's team also performed surveys on developers to ask about conflict resolution strategies. They both came to similar conclusions that merge resolutions fall into a variety of categories. A single merge tool would not be able to properly reconcile all merges, but could potentially eliminate a portion of them. There is some work on alternative software to help manage commits. Matias Martinez focuses on creating a tool to perform fine grain tracking of commits across a git repository [18] hoping to enable more fine tuned control over changes. Anita Sarma's on the other hand, suggests a non git based solution called Palantir, whose purpose is to preemptively warn developers of conflicts [28]. Both of these solutions are dependent on secure change history, and help to highlight the relevance of author identification.

Security research on Github is usually focused on secrets that are erroneously contained in the repositories themselves, such as API tokens or passwords used in services outside of GitHub. Two papers focus on security key leakage on Github. Michel Meli from North Carolina State University performed a quantitative analysis over a large number of live GitHub repositories [19] and found that sensitive

information is being leaked in nearly 100,000 repositories. Runhan Feng's team at the Shanghai Jiao Tong University constructed a large data set using the GitHub API to train a model capable of automatically detecting secret leakage on GitHub [6]. Their work is focused on detecting leaked passwords rather than cryptographic secrets.

There is some work looking at GitHub as a security discussion platform, analyzing human sentiment in commit messages and issues. A paper out of the University of South Florida developed a model to detect mentions of security incidents from a variety of platforms [11]. They suggest a sort of forecasting solution, comparing the response time of various platforms when there is a security incident. A similar paper by Daniel Pletea *Security and Emotion: Sentiment Analysis of Security Discussions on GitHub* [25] looks to GitHub discussion specifically for sentiment analysis about security. Using a natural language model they found nearly 10% of discussion on GitHub is related to security, and that security discussion is more negative than other topics. There is other work on potential vulnerabilities in the Github system. We found two papers auditing the security of code generated by Github Copilot [24, 7]. Both papers found that around 40% of code generated by Copilot contained some vulnerability.

There are a number of other projects gathering large amounts of data from GitHub. We found a couple of papers on the quality of data collected from GitHub. Eirini form the University of Victoria Canada published an analysis of git repositories in 2014. [13]. This work highlighted the perils of treating all data on GitHub as an open development project. They found that a large number of repositories are inactive and two thirds are personal. Valerio Cosentino collected a number of other works related to GitHub data and vetted the integrity of the methodology of those works [5]. They express a concern with poor repeatability due to most works opting to construct a data set from scratch. They recommend datasets be shared to allow for better outside analysis. The largest collection of GitHub data to date comes from our own University. Audris Mockus's world of code project maps every public git commit to authors and allows for large scale analysis of commit and repository data

[16, 17]. The focus of all of these papers is on data quality or collection. Our work differs as we are concerned with security and usability. A paper similarly focused on security in the category is one on the threat model of Github Actions Workflows from the University of Genova, Italy [4], They sampled a variety of workflows on GitHub. Workflows are automated build and deployment systems provided by many of the top git implementations. Out of 131,168 GitHub Workflows, they found 24,905 security issues.

Asymmetric key usability is a broadly discussed topic. Secure email is a popular research topic that has been published on multiple times within our own lab. We found these papers in addition to our own that outline common problems and solutions to the secure email. [27, 14, 8, 26, 15]. Each of these works contains significant discussion on the difficulty of usable digital signature systems. They all cite key mismanagement as a failure point in the secure email systems. We have also found some work on improving SSH key systems. [3, 2]. These two works are focused on adding additional layers of security to the existing SSH system. There has been some investigation of key management and user interaction on the SSH platform. Oliver Gassers's work *A deeper understanding of SSH: Results from Internet-wide scans* [9] performed wide scale IP scans to perform SSH handshakes with other networks. They constructed a large data set used to appraise the current state of SSH usage. This work acted as a general census and does not have a focus on security. Martin R. Albrecht's team did perform a security analysis on SSH in their paper *A Surfeit of SSH Cipher Suites* [1]. They used a similar scanning method to Gasser. They used this scan to detect how many networks use weak encryption cipher suites such as CBC. After performing successful attacks on servers with these ciphers they recommended these weak ciphers be removed from SSH. This work is similar to ours but comparatively broad in scope, with no focus on code development.

Chapter 3

Methodology

3.1 Data Collection

We scanned GitHub for data to answer our three research questions. What are the quality of user keys on Github? What algorithms are the most used, how strong are the keys themselves? This requires data on the keys themselves. Secondly we ask how many users have commit signing keys. This includes corollary data about the accounts the keys are tied too. For instance, how active is the account? How popular are the repositories owned by the account? What systems are they using? Finally how many commits are signed? How many repositories contain a signed commit? How popular are repositories with signed commits?

We gathered data for each of these three research questions.

RQ1 Basic user information was quickly scraped from GitHub. Using this comprehensive list we began scanning all users for available keys. This includes both authentication and signing keys. This process is slow, and expected to finish around six months after the due date of this thesis. So a subset of around 20% of the accounts will be analyzed in this paper.

RQ2 More advanced user information was collected from the API after the initial list was constructed. This process is greatly expedited when all users identifiers

are known. This information includes most recent user activity, a count of all active repositories, issues, and contributions.

RQ3 Using World of Code. We were able to perform large scale scans on all public commits on Github. This allowed us to answer overarching questions about the popularity of commit signing on Github.

We have collected profile information from all 88 million Github Users and are actively collecting all keys present on their accounts, tables 3.1 and 3.2 show describe the data collected. Our collection method used a Python script making constant API requests to Github. This script is hosted on a docker instance on one of our personal lab severs. We employed pythons ayncio libraries in order to make more requests per hour. The request rate of the script is metered to evenly access the API in order to make a more consistent access pattern and avoid mass requesting information from Github in a short time.

Github's primary API access system limits requests per user agent - the default rate of requests a user agent has access to is 5000 per hour. Our initial strategy was to create 50 Github accounts and generate an API access token on each account to increase the number of requests we could make per hour. This method did not work as these accounts did not have two-factor authentication, Github's bot detection systems flagged and banned accounts within a few days. After corresponding with personnel at Github, we learned there was no way to save these accounts from banishment and were guided to use a Github Oauth Application for large scale API access.

Our code base was retooled to use an Oauth Application created under the UTK USER lab Github organization. Access tokens generated under this system are required to be connected to a user account. Live user accounts from other researchers within the USER lab were used to generate API access tokens. These tokens were generated using Githubs device flow system. After generating the tokens we once again made contact with Github Support via the support ticket system and were able

Table 3.1: Data Collection. Main Pass

Name	Description
Username	Github user name.
UserId	Integer Value Associated with the user, roughly chronological.
NodeId	Base64 Value Associated with the user.
SiteAdminStatus	Whether or not the user is an enterprise admin.
UserType	Specifies if the agent is an individual user or acting as organization.
SigningKeys	Any GPG or SSH keys associated with the User.
AuthKeys	Any Authentication keys associated with the User.

Table 3.2: Data Collection. Secondary Pass

Name	Description
Name	User specified name (optional).
Location	User specified location (optional).
Email	User specified email (optional).
Updated	Last time the user did anything on GitHub.
Organizations	Number of Organizations the user is associated with.
Repositories	Number of Repositories the user owns.
Recent Repositories	Number of Repositories the user has committed to in the past year.
Issues	Number of issues associated with the user.

to have the rate limit per hour increased to 12500 requests for tokens associated with our Oauth Application.

The keys and identifying user information are stored in a Sqlite database. Sqlite is a lightweight single file SQL implementation. A Python script makes many simultaneous requests taking advantage of the async library to efficiently wait on multiple processes. The request data is parsed and stored in the database as it is retrieved. The Database is stored on the same server running the python-docker instance.

These rate increases are not enough to collect all user keys before the deadline of this thesis, so a subset of the keys were be analyzed in the meantime. At the time of writing, 17% of users have been scanned for GPG and SSH commit signing keys and 21% have been scanned for authentication keys.

3.2 Analysis

The primary purpose our analysis was to gain an understanding of users with GitHub Keys. We gathered a large sample of user keys from GitHub, both authentication and signing. The keys themselves were analyzed to answer basic questions about the state of security on the platform. Walks through the database were made to get counts of every type of key. For each of the three categories (authentication, SSH signing, and GPG signing) we walked the database and count the number of each key, the number of each key algorithm, the length of each key relative to the algorithm, and in the case of the signing keys, the date of creation for the key. We performed additional scans over the database to find the total number of users who have signing keys to account for the overlap in GPG and SSH signing ownership. This provides information required to estimate the security of keys and the percentages of algorithm use in the database per key category.

Key strength was found using the keys algorithm and length. It was calculated as needed and then formatted into counts of keys by algorithm and category. Once

we had these strength measurements we were able to investigate the overall average strength of each key category and find the percentage of weak keys in each key type, A percentage is important here as the amount of collected keys per each type varies drastically. Comparison between the types of keys is also useful. We compared key ownership of authentication keys to ownership of signing keys. This is done by performing scans of how many users have both signing and authentication key. This is useful to draw correlation between key ownership on the platform.

In addition we investigate the discrepancy between the standards for GPG signing keys and SSH signing keys on GiHub. We compared the security difference in these keys over time to see the effect of security policy changes applied to SSH signing keys vs the unchanged GPG keys. Security can be found by using the keys algorithm and length to normalize its strength to a universal measurement. This time analysis was done as a separate walk through the database. Key creation timestamps were used to group the keys chronologically by year. Once the data was in this form we were able to investigate the security of signing keys over time.

We collected additional data on users, including statistics about their created issues, repositories, organizations, recent contributions, and general recent activity on the platform. This information was collected via GraphQl. This system allows for us to gather a variety of data at one time with custom requests, however if a request becomes to complex it simply fails. Due to limitations of the GraphQl request system, we were not able to gather all available user metadata at once, and will need to gather additional contextual data in the future.

This user information was used to create a general outline of the average Github user. We first inspected baseline data including all gathered user data in order to understand the differences between users who have keys and users who do not. This included the number of active users, the number of repositories a user has, the number of repositories a user has recently committed to, and the number of organizations a user is part of. This information was collected via similar walks through each item in the database. Items in this data base are checked against users in the key database

in order to separate the user information into categories depending on which keys they had. We separated them into six categories: all users, users with signing keys, users with authentication keys, users that do not have signing keys, users that do not have authentication keys, and users that have no keys. Each of these sets of users had analysis performed on it. We found the engagement of users in terms of recent contributions, issues, repositories owned, and organizations. We find the average number in each category, the standard deviation, and the percent of users in a range of values. User last active data is presented as percentage of users who were last on GitHub by year.

This information is then correlated to users with keys to see how they behave in comparison to the norm. We look for correlations between higher GitHub Activity and key ownership. We were able to draw significant connections between key ownership and general involvement on GitHub. We also constructed a secondary user key data set to investigate the key usage of users who are known to be active on GitHub. We looked at their user ship by percentage as well as their preferences on key algorithms and security strength.

Our World of Code analysis currently consists of large scale counts of commit information across public GitHub repositories. The world of code repository is hosted on a series of shared storage networks. Access is given out on an application basis and data analysis occurs over a SSH network tunnel connection. We scanned a portion of the commits in the world of code repository to get an initial understanding of how many commits on the platform are signed. Work with the world of code platform is still ongoing as of the time of this thesis.

3.2.1 Implementation

Statistical analysis was performed using Jupyter Notebook on Python. Jupyter Notebook is a cell based coding system built on python designed for data analysis and visualization. Numpy was used to perform more taxing statistical analysis such as

key strength calculations and standard deviation calculations. The GnuPG package was used to pull metadata from collected GPG keys.

A few methods were used to speed up data access on sqlite. Users were indexed by their user ID, which is a unique integer value. This drastically speeds up individual user lookup which is used commonly for both data collection and analysis. Sqlite does not allow for concurrent database access. Multiple database files were used to hold data from different scans which was reconciled later after collection. This allowed us to utilize a multi threaded approach. Threads could then be split into different jobs for the different scans performed to reduce the load per thread. To reduce the size of the main table we attempted a sharding solution, splitting the table into multiple tables with a maximum row count of 10 million. This sharding approach showed negligible performance improvement.

Security strength, is defined by NIST as the amount of work, or number of operations, required to break a cryptographic algorithm or system.[21] It is expressed as bit strength, a bit strength of 80 would require 2^{80} attempts to break on average. This value is calculated differently depending on the algorithm used to generate the key. The following algorithm defined in FIPS 140-2[22] was used to calculate the bit strength of RSA keys.

$$x = \frac{1.923 \times \sqrt[2]{L \times ln(2)} \times \sqrt[3]{(ln(L \times ln(2)))^2} - 4.69}{ln(2)}$$

Where L is the length of the RSA key and x is bit strength. DSA keys are calculated using this formula.

$$y = min(x, N/2)$$

Where y is bit strength, x is the result of the equation used to calculate RSA key strength using the length of the Diffie-Hellman public key as L, and N is the length of the Diffie-Hellman private key.

The strength of elliptic curve algorithms was determined using table 3.3 provided in NIST 800-57 [21].

Table 3.3: Security Strength estimates by algorithm as defined by NIST 800-57

Security Strength	Symmetric Key Algorithms	FFC*	IFC†	ECC‡
<80	2TDEA	L = 1024 N = 160	k = 1024	f = 160-223
112	3TDEA[68]	L = 2048 N = 224	k = 2048	f = 224-255
128	AES-128	L = 3072 N = 256	k = 3072	f = 256-383
192	AES-192	L = 7680 N = 384	k = 7680	f = 384-511
256	AES-256	L = 15360 N = 512	k = 15360	f = 511+

*RSA

†DSA, DH, MQV

‡ECDSA, EdDSA, DH, MQV

All statistical analysis was performed using a Python Notebook managed by a devcontainer. For actual data aggregation and operations, we kept the data in SQlite and operated on the data line by line through sqlite commands via pythons SQlite3 module. Due to the size of these databases being in the millions, common solutions like pandas were not viable for performing operation on the full data set. Lazy loading solutions like line by line file reading and SQlite scans were used instead.

As information was read from the database, Python dictionaries were used to store relevant information like the number of occurrence of each type of key, the length of these keys, the counts of the number of users in each key category. This allowed for important information to be condensed into a small data structure that can be held in memory. This data was saved to a file in JSON format for long term storage to prevent recalculating results. This information was enough to perform analysis over usage percentages, standard deviations, and averages.

User data was analyzed in a similar fashion. We walk through the database of user information and used the saved user ID's in the table to perform lookups in the database files containing keys to construct data sets of users who met conditions such as key ownership and recent activity.

Information taken from World of Code was gathered differently. The World of Code database is hundreds of Gigabytes and as consequence is access directly via SSH network connections. Database operations were primarily performed using command line tools to iterate over a series of mappings between the various types of data on the platform. The command script we used to collect the number of signed commits on the platform was:

```
for i in 0..127; do perl  /lookup/lstCmt.perl 3 $i; done | grep -c
"gpgsig"
```

This made an overarching scan of the stored commits and searched them for a string unique to singed commits. Our initial scan was reduced in size from `0..127` to `0..10`. This was to allow for a series of scans that would not monopolize the server

resources for too long and would allow for data to be collected in parts for faster initial analysis.

Chapter 4

Results

Here we discuss the results of our analysis. These results are able to answer the research questions established at the beginning of this paper. Namely, what is the quality of authentication keys, what is the usage rate of signing keys, and how many commits are signed?

4.1 User Analysis

We performed our investigation on 13 million GitHub users. The most notable fact to take from this data is the vast majority of GitHub users are not active and have never contributed to a repository. Over 50% of users have not been active on Github in the past four years, and 94% of users have not made a recent contribution to any repository and 44% of users do not own a repository and 80% of users have never made an issue. We define recent as within the past year. This level of inactivity from the average GitHub user is significant. This tells us only a small portion of users on GitHub are active developers who are pushing code to repositories. The primary focus of key security is to ensure the integrity of code on GitHub. These users are responsible for the majority of the code on the platform. Their accounts have wide access to large amounts of code on the platform, making their account security significantly more important than those who do not contribute code. This

also opens some concern for the misuse and targeting of old accounts that are no longer active but used to be.

Most user accounts have not been active in recent years. We have found that 74% of all users have not had any activity on their GitHub account since 2023. In the case of users with any type of key, more than 90% of accounts have been active since 2023. Figures 4.1-4.3 Graph the percentage of users who were last active in a given year for all users, users with authentication keys, and users with signing keys.

We find significantly different engagement rates between users who have cryptographic keys and users who do not. Out of all 13 million scanned user 6% have made any contributions in the past year, 2% are part of at least one organization, 20% have made an issue and 44% own at least one repository.

Users with authentications keys much more active. 51% have made contributions, 46% are in an organization, 79% have made an issue, and 96% have a repository.

Users with commit signing keys are significantly more active than users with authentication keys. 64% have made recent contributions, 46% are in organizations, 92% have made an issue, and 99% own a repository. Figures 4.4-4.7 compare these values, and tables 4.1-4.6 show average rates for each of these categories for a variety of user types. We gather from these that users without keys are the least engaged on the GitHub platform.

This shows that accounts with keys attached to them are among the most active accounts on the platform. We also see this behavior when considering the last active date of users on the platform.

We performed a one sided test of percentages to find significance values for each category of activity comparing users with login keys, users with signing keys, and general users to ensure the disparity in activity is statistically significant. We found that for every category the confidence value p was lower than 0.0001. This allows us to say the disparity in activity between the types of users compared above is significant with 99.9% confidence.

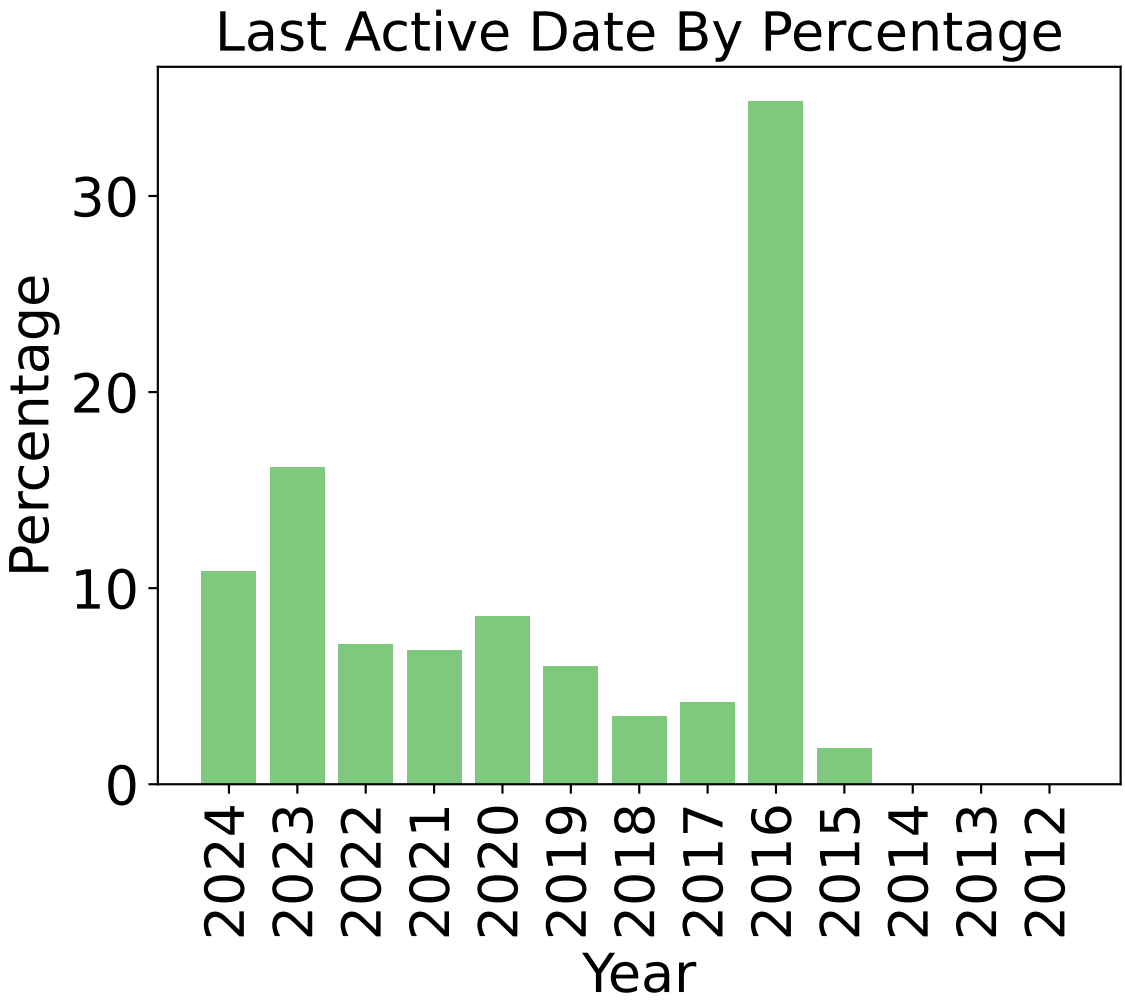

Figure 4.1: Percentages of when users last appeared on GitHub.

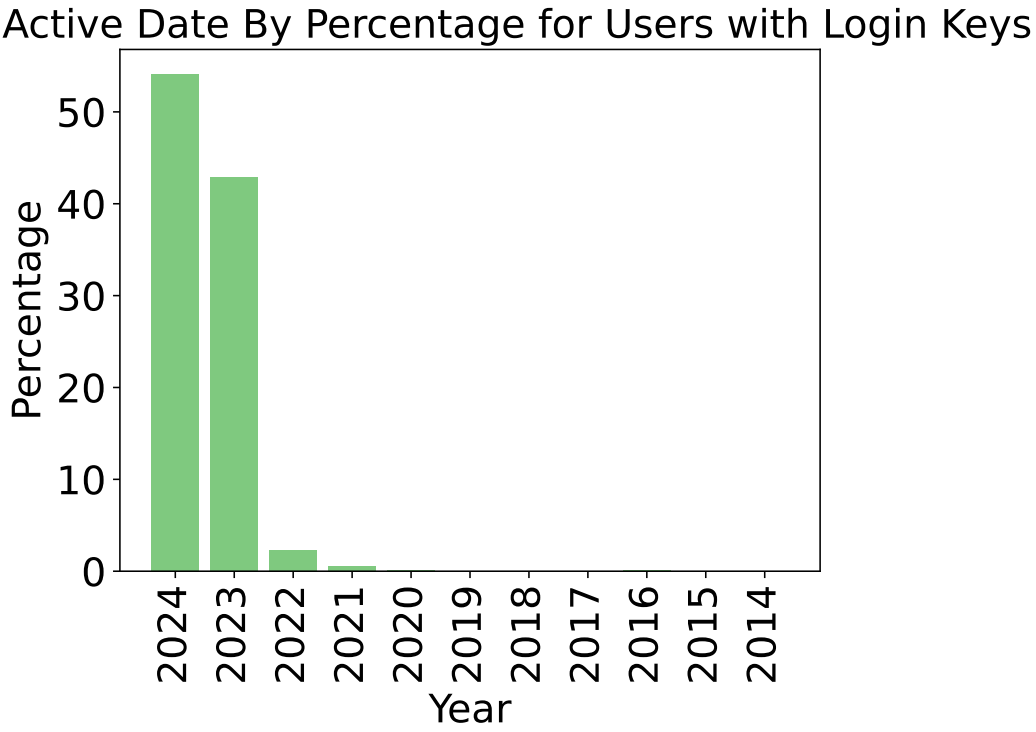

Figure 4.2: Percentages of when users with login keys last appeared on GitHub.

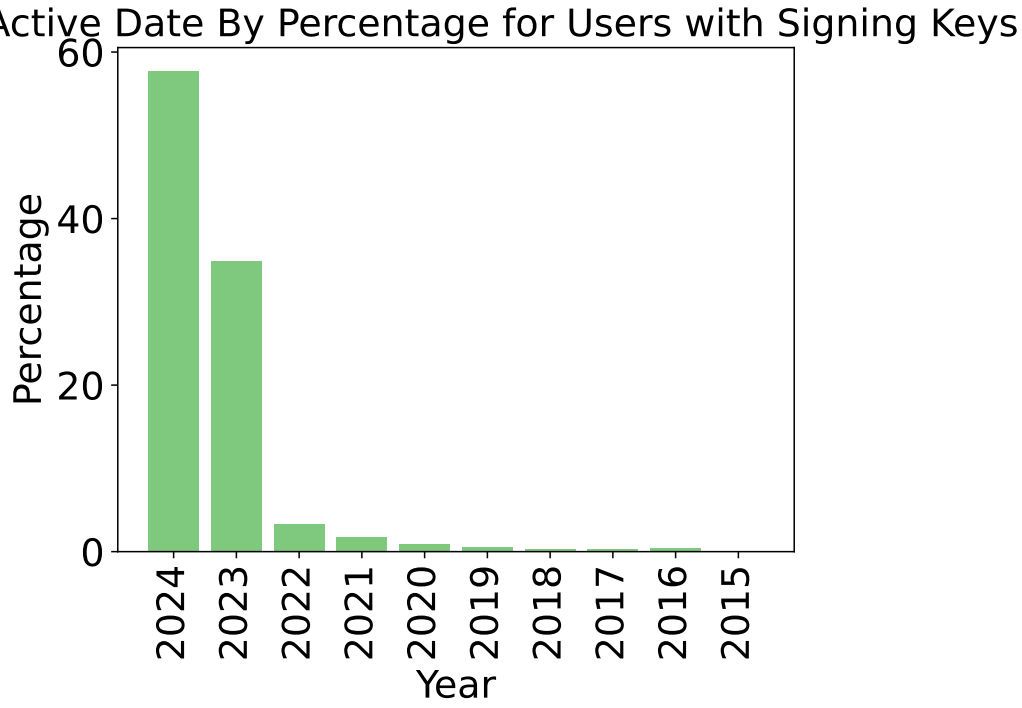

Figure 4.3: Percentages of when users with signing keys last appeared on GitHub.

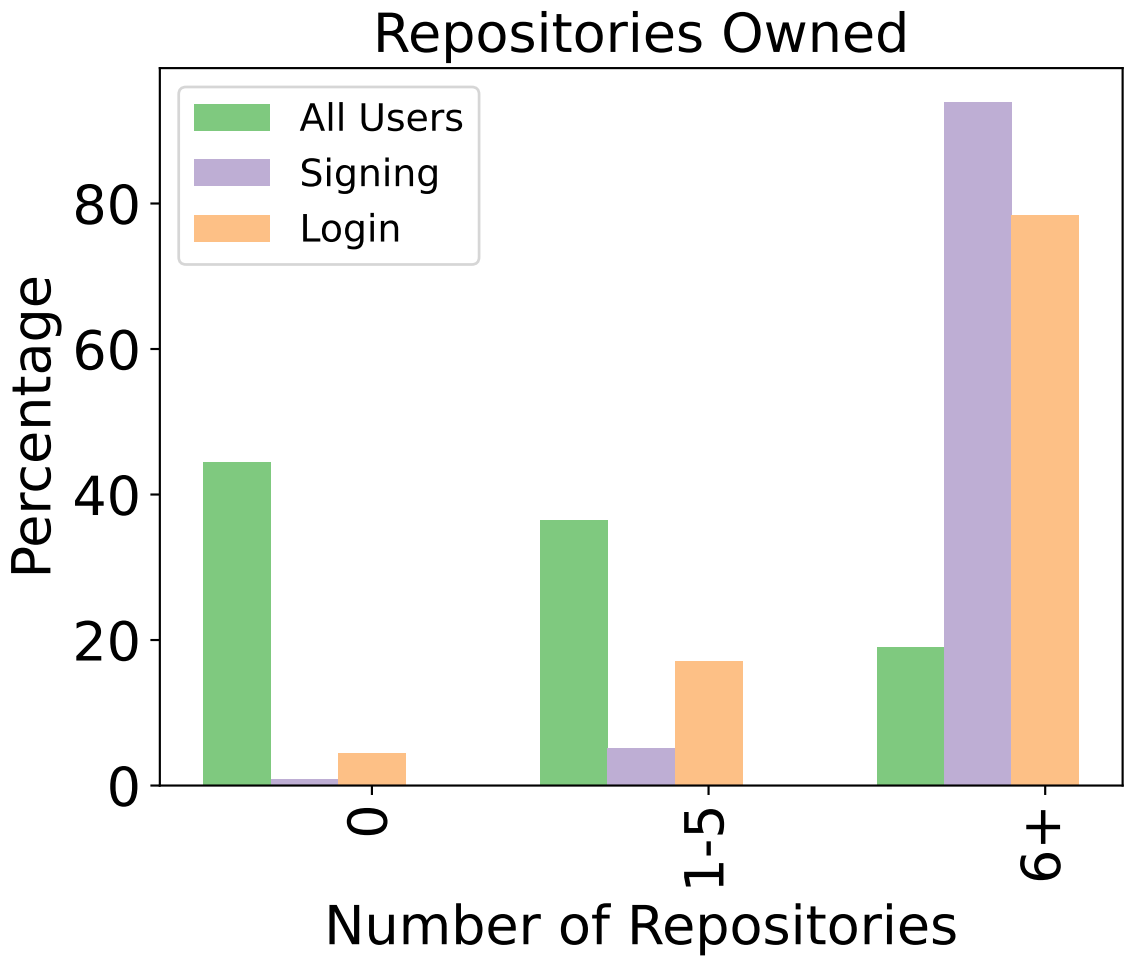

Figure 4.4: Percentage of Users who have a number of Repositories under associated with their account.

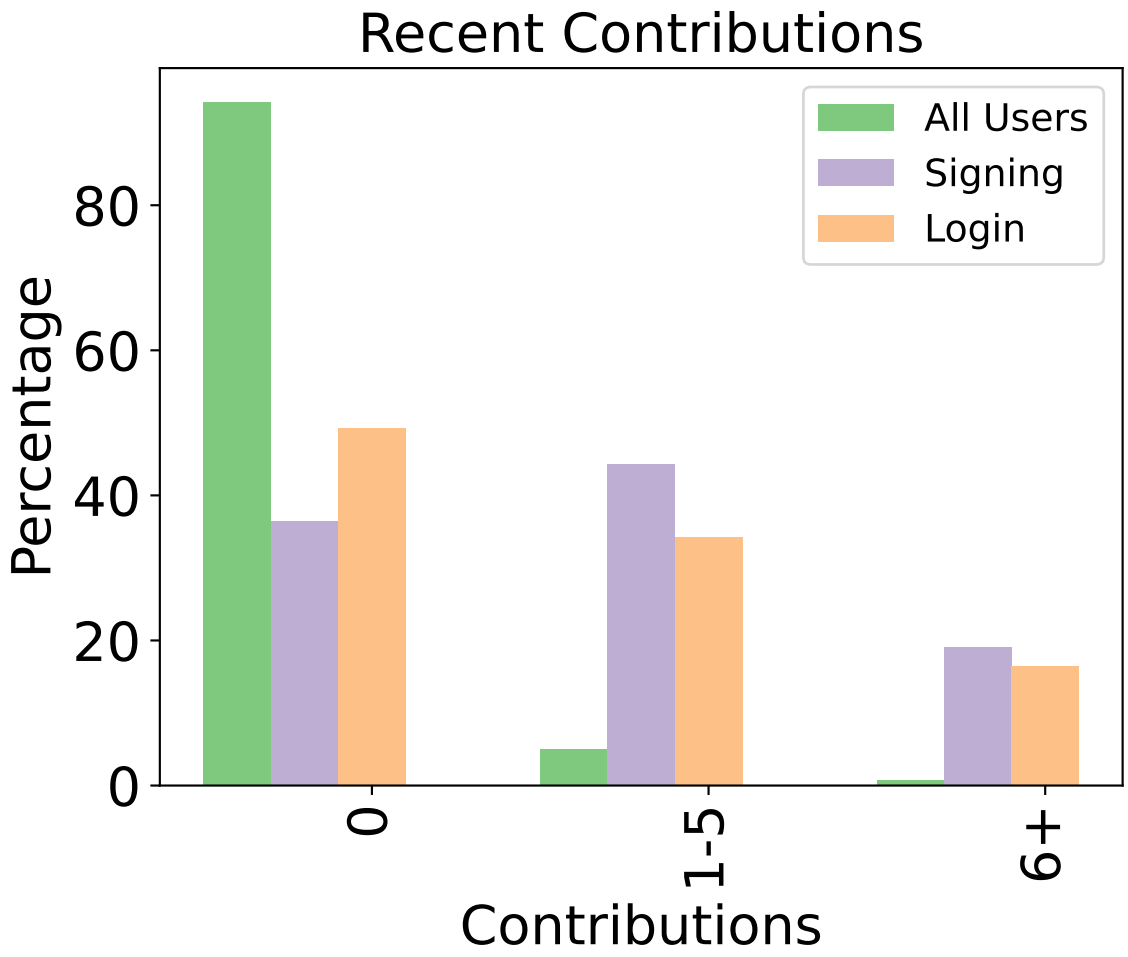

Figure 4.5: Percentage of Users who made a recent contribution to a number of repositories.

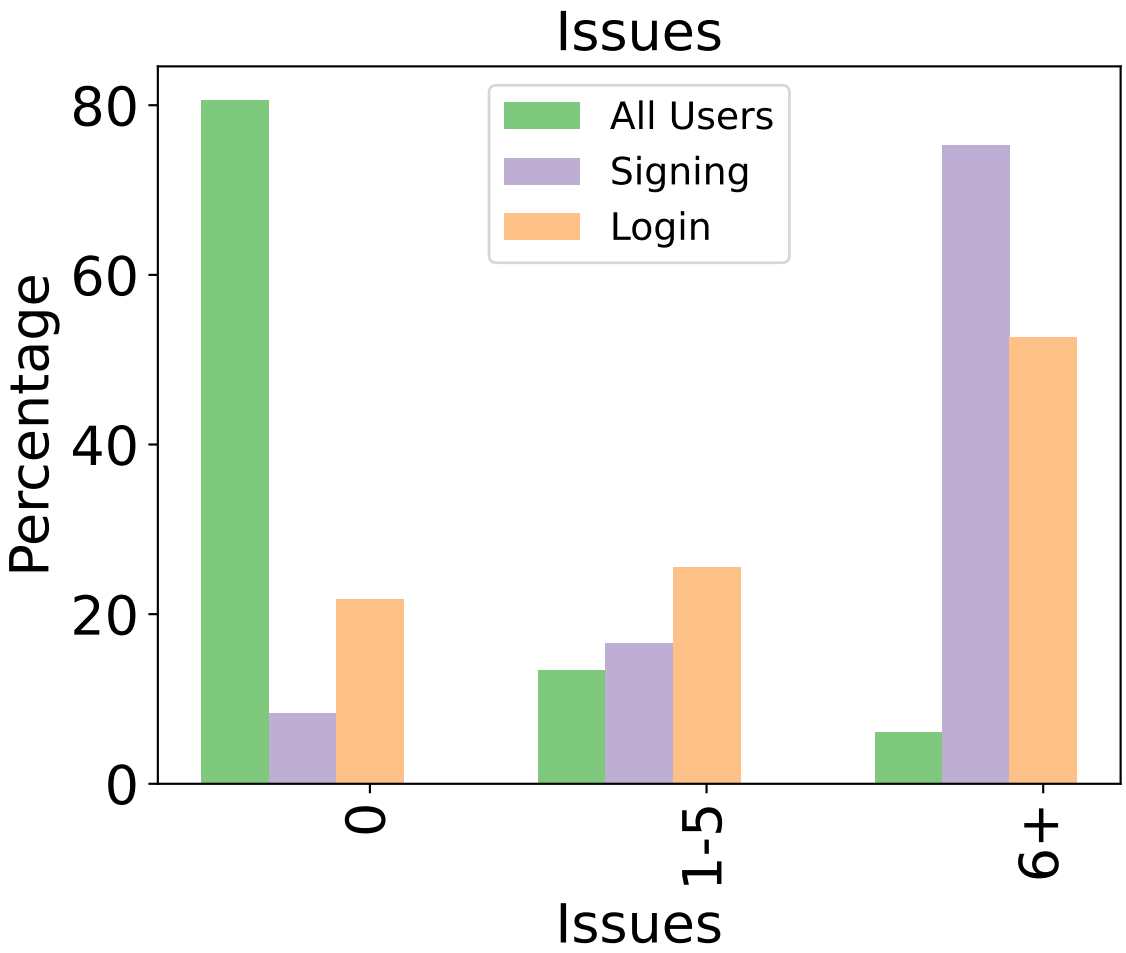

Figure 4.6: Percentage of Users who have made some number of Issues.

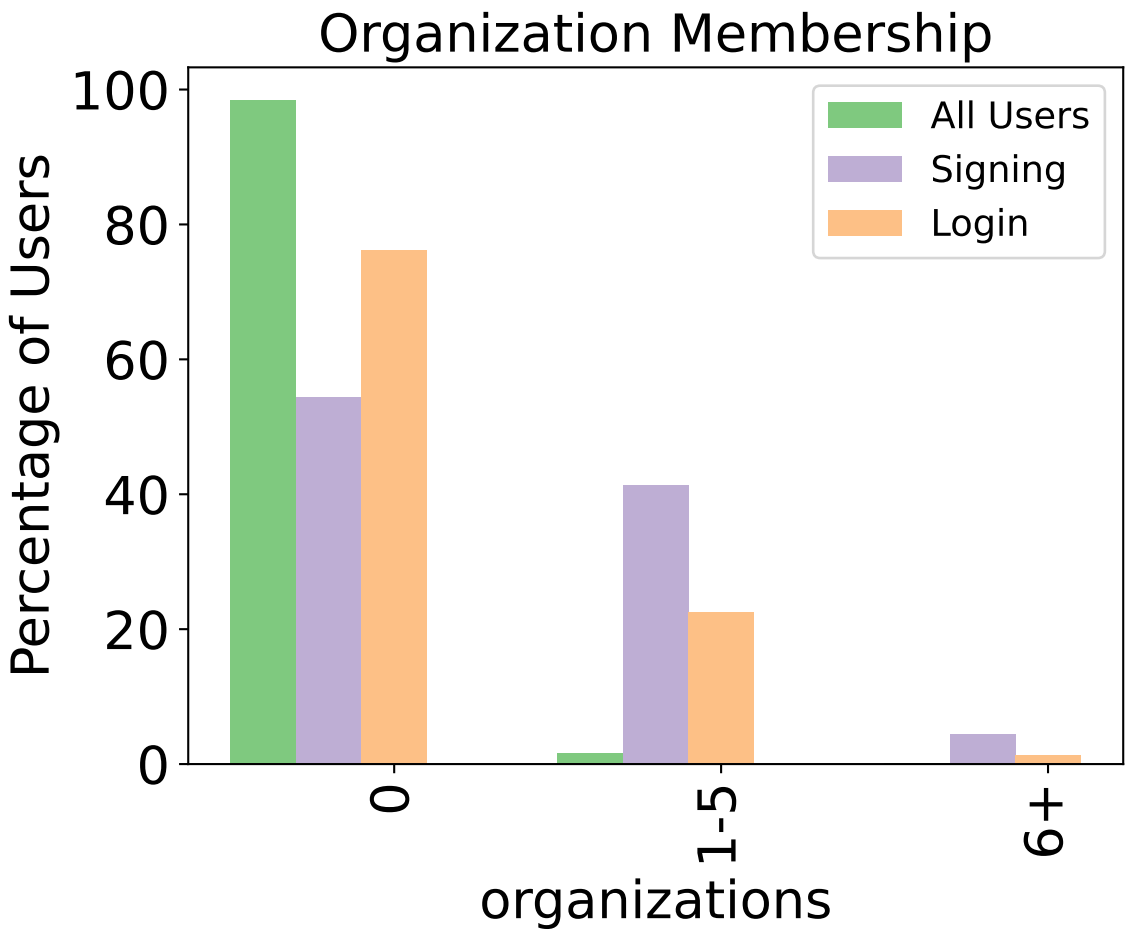

Figure 4.7: Percentage of Users who belong to a number of Organizations.

Table 4.1: Engagement Values for All Users

Category	Mean	Standard Deviation
Contributions	0.2114	4.0913
Repositories	5.5832	37.7044
Organizations	0.0257	0.2803
Issues	3.0279	85.9706

Table 4.2: Engagement Values for Users With Login Keys

Category	Mean	Standard Deviation
Contributions	1.6956	10.0911
Repositories	24.4814	61.1777
Organizations	0.198	0.8065
Issues	20.5299	162.8192

Table 4.3: Engagement Values for Users With Signing Keys

Category	Mean	Standard Deviation
Contributions	3.6889	15.6651
Repositories	35.2887	102.6553
Organizations	0.4986	1.3781
Issues	45.5339	180.8436

Table 4.4: Engagement Values for Users Without Login Keys

Category	Mean	Standard Deviation
Contributions	0.0814	3.0126
Repositories	3.9278	34.4134
Organizations	0.0106	0.1602
Issues	1.4948	75.4122

Table 4.5: Engagement Values for Users Without Signing Keys

Category	Mean	Standard Deviation
Contributions	0.1741	3.7619
Repositories	5.2641	36.2507
Organizations	0.0206	0.2379
Issues	2.5713	84.258

Table 4.6: Engagement Values for Users Without Keys

Category	Mean	Standard Deviation
Contributions	0.0766	2.9935
Repositories	3.8637	33.4781
Organizations	0.0098	0.1512
Issues	1.4179	75.099

4.2 Key Analysis

On initial investigation, a small percentage of users have authentication or commit signing keys - 7.2% and 1.2% respectively. We investigate the usage and security of both types of key separately below.

4.2.1 Authentication Keys

Out of 20 Million Users, 1.3 Million have SSH Login keys, or about 7.2%. This result is more interesting as Github has recently removed the option of password authenticated code deployment and now requires keys for commits. Some explanations for this number could be the recent phase out and removal of weaker key ciphers such as DSA. It could also speak to the number of active developers on Github. We know that the majority of accounts are not active and have not contributed code to any repository. If users are not pushing code to repositories they would have no reason to create authentication keys. There is also some concern with the reliability of GitHub request data, as the implementations for Rest API and GraphQl api differ.

59.61% of SSH authentication keys are RSA, 39.08% are Ed25519 while the other 1.31% are ECDSA. The prominence of RSA is likely due to the long standing popularity of RSA as an algorithm. The rise of Ed25519 usage is encouraging. It is currently the most secure and efficient asymmetric key algorithm. ecdsa was the most common elliptic curve solution before Ed25519 and seems to have been mostly replaced by the newer solution. Table 4.7 contains these results.

Users frequently have more than one key associated with their account. Users with login keys have 1.8 login keys on average. While users could transfer their login key between devices, many opt to generate new keys for new devices as they are needed.

The majority of authentication keys meet the recommended 112bit security strength. All keys using elliptic curve algorithms are at least 128bit. However there are a number of weak keys that appear in our dataset. Approximately 0.4% of login keys are below the recommended strength threshold. These are entirely made

Table 4.7: Counts of SSH Authentication keys, and number of occurrences of each supported Cipher Suite

Cipher	Occurrence	Percentage
ssh-rsa	1480021	59.61
ssh-ed25519	970451	39.09
ecdsa-sha2-nistp256	18009	0.73
ecdsa-sha2-nistp521	9924	0.4
sk-ssh-ed25519@openssh.com	2647	0.11
sk-ecdsa-sha2-nistp256@openssh.com	950	0.04
ecdsa-sha2-nistp384	637	0.03

up of RSA keys with below recommended key lengths. Table 4.8 shows actual counts, security strengths, and algorithms for weak authentication keys. We see the majority of these weak keys are SSH RSA keys with a key length of 1024 which is now considered outdated. Figure 4.8 graphs these key ciphers by strengths and popularity.

4.2.2 Signing Keys

Out of 14 million users, 20,000 have SSH signing keys and 150,000 have GPG signing keys. Accounting for overlap, 1.2% of users have any form of commit signing key. This makes GPG the most popular commit signing system by far. These users who do have signing keys are more active than users with authentication keys. This shows users with more contributions are more likely to use signing keys. We find that 60% of users with signing keys have made recent contributions compared to 6% of average users.

SSH commit signing keys favor Ed25519 over RSA. 67% keys are Ed25519, 28% are RSA, and 5% are ecdsa. There are several possible explanations for this discrepancy. As of 2021 Github added support for newer elliptic curve algorithms such as Ed25519 and phased out support for algorithms they consider weak such as DSA. We have not found any SSH keys in the database created before 2022. As a result we theorize many of these Ed25519 keys might have been made as replacements for older, no longer supported key types. Users with GPG keys have 1.19 keys on average and users with SSH keys have 1.16

However this restriction is only placed on SSH keys. DSA algorithms still appear as the third most popular GPG key. 84% of keys are RSA, 13% are Ed25519 0.8% are DSA, and 0.3% are ECDSA. It is unclear why Github has different standards for SSH keys and GPG keys. This could be a sign that they are less concerned with commit keys as a whole, as GPG keys can only be used for commit signing. The continued allowance of these keys creates a mixed standard for key maintenance on GitHub and does result in weaker keys overall.

Table 4.8: Counts of Weak Authentication keys

Cipher	Key Length	Occurence	Bit Strength
ssh-rsa	1024	5513	80
ssh-rsa	2018	36	109
ssh-rsa	1023	25	80
ssh-rsa	2024	14	110
ssh-rsa	2028	13	110
ssh-rsa	2000	7	109
ssh-rsa	2038	5	110
ssh-rsa	2040	4	110
ssh-rsa	1028	4	80
ssh-rsa	1536	3	97
ssh-rsa	1039	2	81
ssh-rsa	1096	2	83
ssh-rsa	2014	2	109
ssh-rsa	1025	1	80
ssh-rsa	1026	1	80
ssh-rsa	1036	1	80
ssh-rsa	1040	1	81
ssh-rsa	1078	1	82
ssh-rsa	1115	1	83
ssh-rsa	1177	1	85
ssh-rsa	1200	1	86
ssh-rsa	1248	1	88
ssh-rsa	1892	1	106
ssh-rsa	1912	1	107
ssh-rsa	1990	1	109
ssh-rsa	1991	1	109
ssh-rsa	2041	1	110

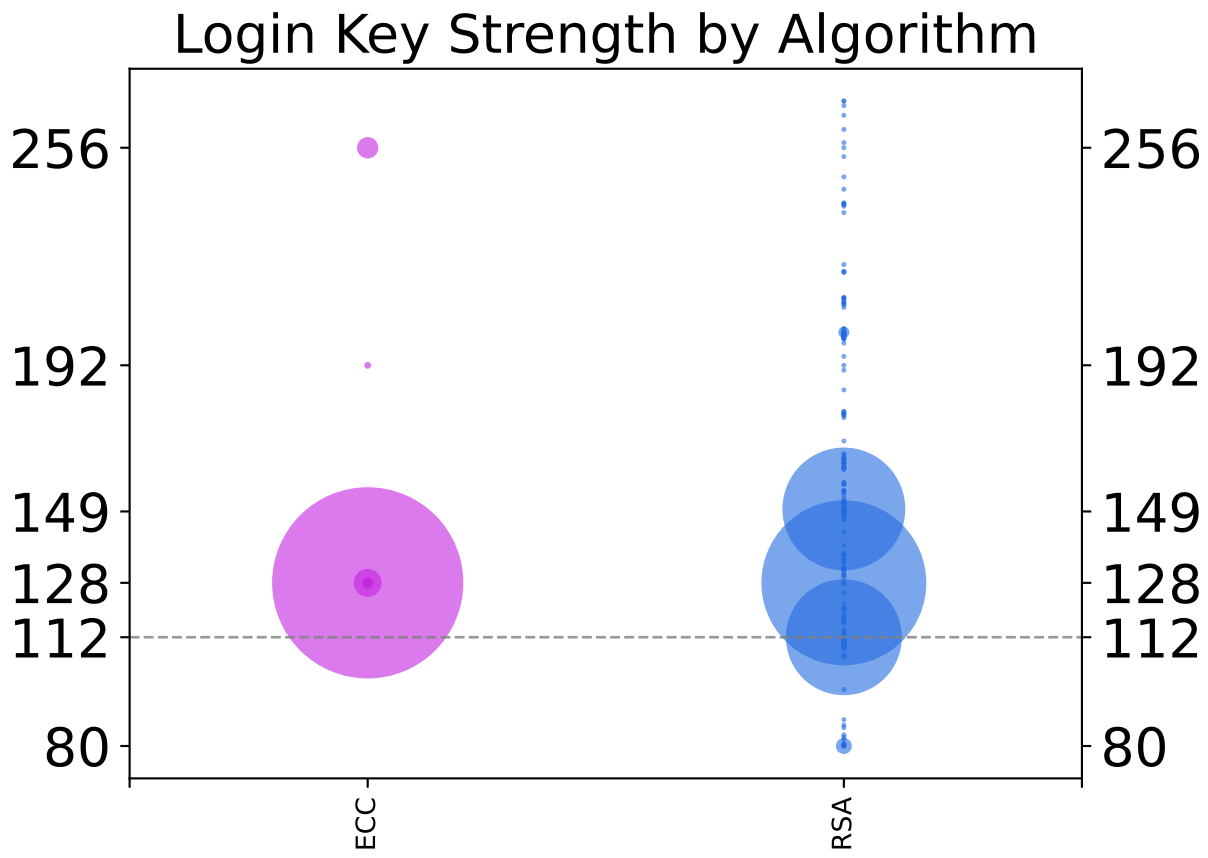

Figure 4.8: Bit strength of SSH authentication signing keys. Each circle represents a different algorithm in a class of cipher suite. The size of each circle represents how many of these keys are found. The center of each circle represent the bit strength of the key algorithm.

No weak SSH keys were found. This can likely be attributed to two factors, SSH signing keys are significantly less popular than GPG signing keys, as a consequence our SSH key sample size is significantly lower. SSH keys have also had security restrictions put on them while GPG keys have not. There are significant occurrences of GPG keys using DSA encryption algorithms. According to Microsoft, DSA is a weak algorithm[20], and is no longer supported for SSH authentication keys on Github. Around 0.6% of GPG keys are weak. The majority of these weak keys are DSA keys and RSA keys with low lengths. Tables 4.9-4.10 show the actual number of occurrences for each cipher suite. Figures 4.9-4.11 graph these key ciphers by strength and popularity. We were not able to find any correlation between key creation time and security strength in the case of GPG or SSH keys.

This does provide some support for the removal of DSA algorithms from the supported cipher suites for SSH keys on GitHub as they make up 69% of weak GPG keys. A full table of these keys and their ciphers is found in table 4.11.

4.3 Active Users

We investigated the subset of users who contributed to at least one project in the last year. We found that these users are more likely to have keys in general with a usage rate of 50% for login keys, and 13% for commit signing keys. This is up from the 1.2% usage of commit signing keys across all users and 7% usage of authentication keys. This shows a correlation between developer activity and key usage, as well as a significant amount of commit signing on the platform from users who contribute code. The behavior of users who are active on the platform and develop code is very different than that of users who do not.

The security of these keys is largely the same. These user have nearly identical usage rates of cipher suites and the appearance of weak keys is proportionally similar. Tables 4.12-4.14 show key usage rates of active users.

Table 4.9: Counts of GPG Commit Signing keys, and number of occurrences of each supported Cipher Suite

Cipher	Occurrence	Percentage
RSA (Encrypt or Sign)	175706	84.9
Ed25519	27171	13.13
RSA Sign-Only	1706	0.82
DSA	1654	0.8
ECDSA public key algorithm	712	0.34

Table 4.10: Counts of SSH Commit Signing keys, and number of occurrences of each supported Cipher Suite

Cipher	Occurrence	Percentage
ssh-ed25519	16427	67.13
ssh-rsa	6855	28.01
sk-ssh-ed25519@openssh.com	515	2.1
ecdsa-sha2-nistp256	442	1.81
sk-ecdsa-sha2-nistp256@openssh.com	134	0.55
ecdsa-sha2-nistp521	73	0.3
ecdsa-sha2-nistp384	24	0.1

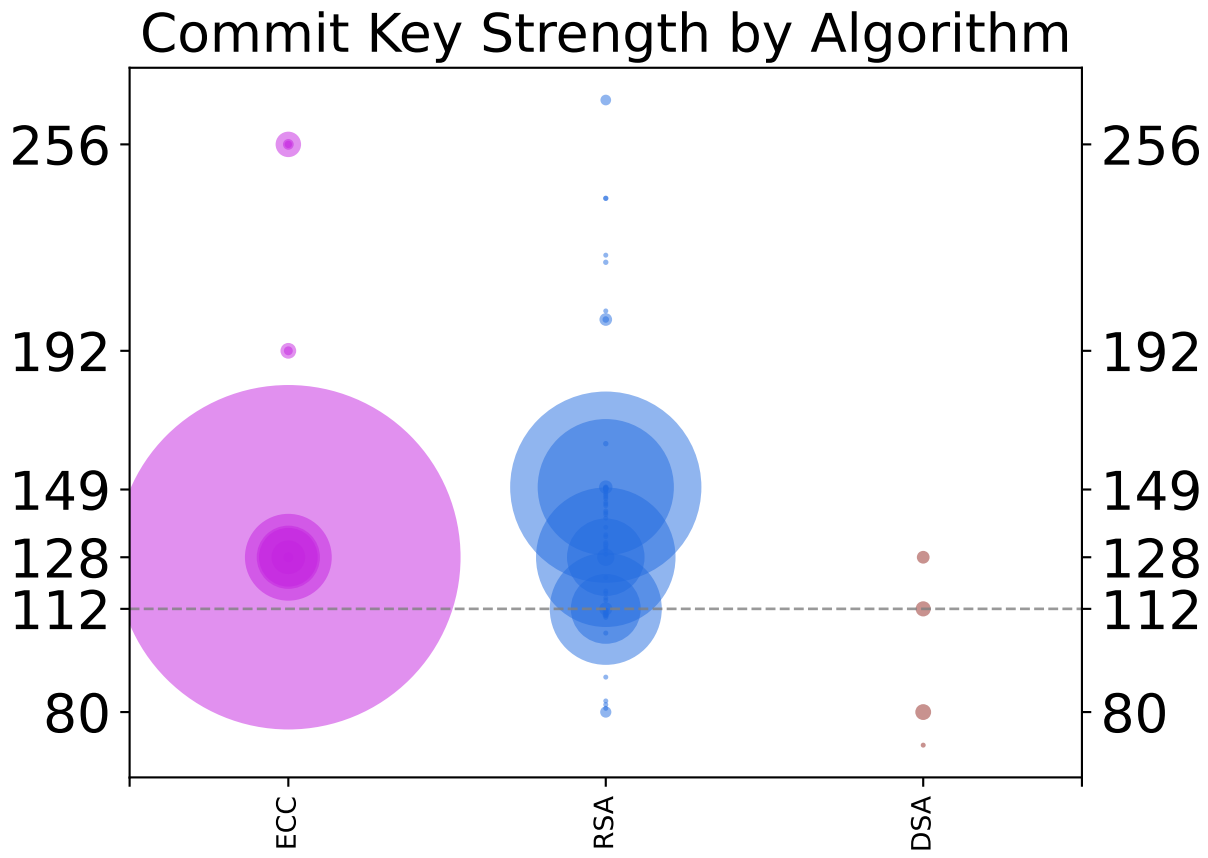

Figure 4.9: Bit strength of all commit signing keys. Each circle represents a different algorithm in a class of cipher suite. The size of each circle represents how many of these keys are found. The center of each circle represent the bit strength of the key algorithm.

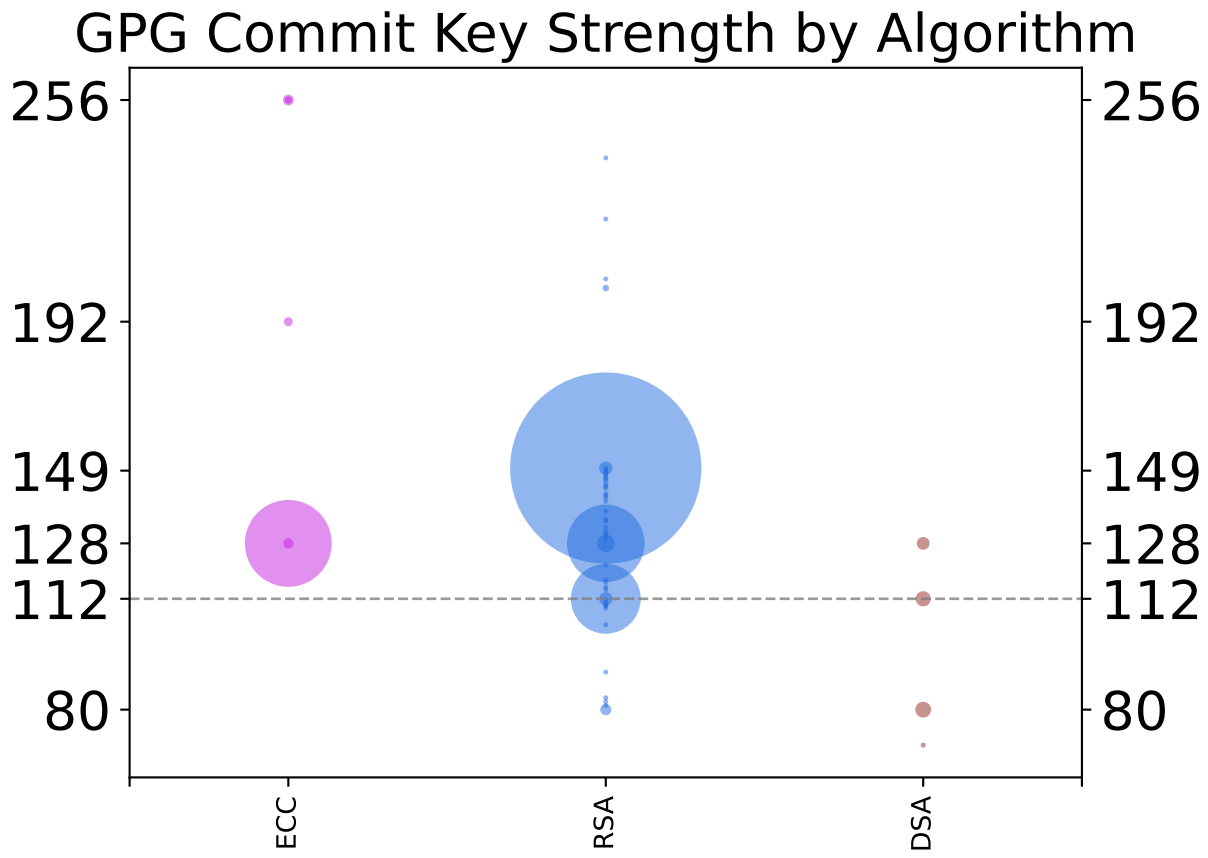

Figure 4.10: Bit strength of GPG commit signing keys. Each circle represents a different algorithm in a class of cipher suite. The size of each circle represents how many of these keys are found. The center of each circle represent the bit strength of the key algorithm.

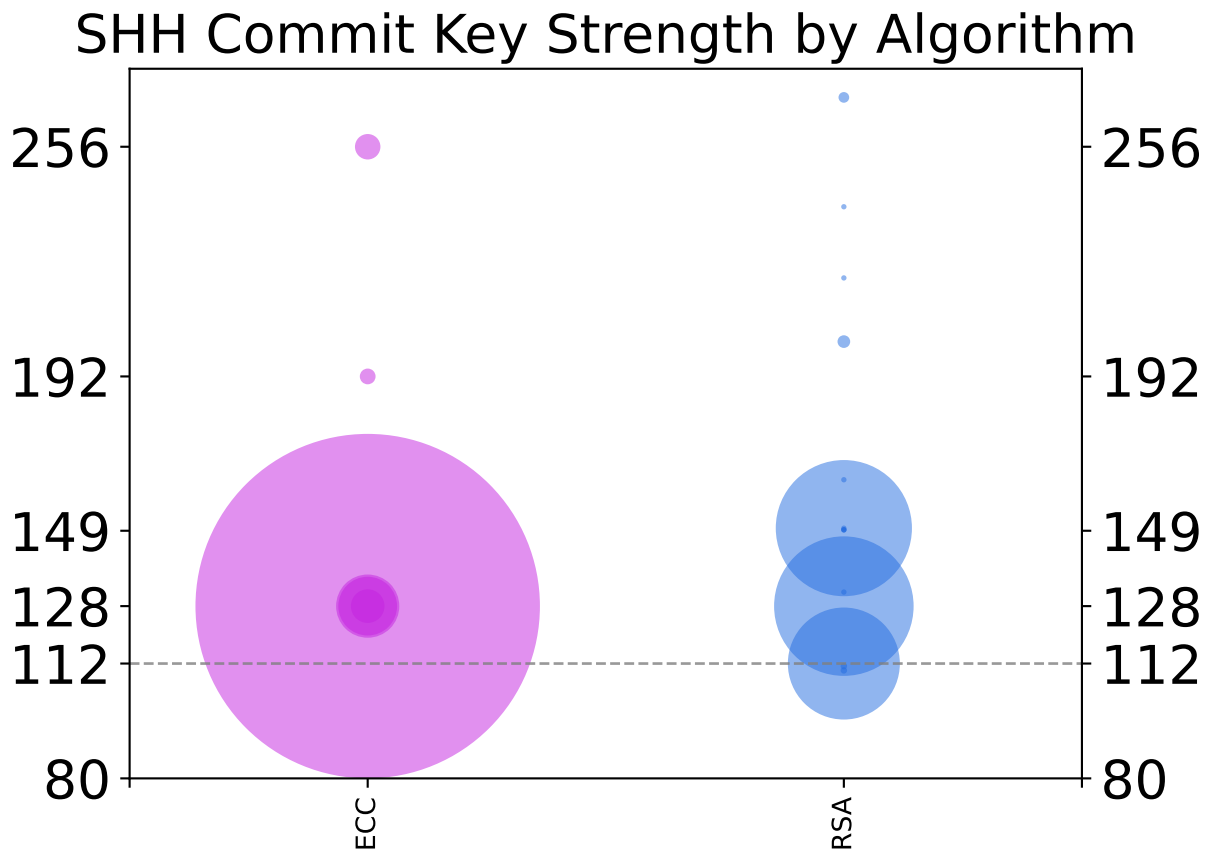

Figure 4.11: Bit strength of SSH commit signing keys. Each circle represents a different algorithm in a class of cipher suite. The size of each circle represents how many of these keys are found. The center of each circle represent the bit strength of the key algorithm.

Table 4.11: Counts of Weak GPG keys

Cipher	Key Length	Occurence	Bit Strength
DSA	1024	664	80
DSA	768	3	70
RSA (Encrypt or Sign)	1024	279	80
RSA (Encrypt or Sign)	2016	8	109
RSA (Encrypt or Sign)	1056	6	81
RSA (Encrypt or Sign)	1120	3	83
RSA (Encrypt or Sign)	1088	2	82
RSA (Encrypt or Sign)	1344	1	91
RSA (Encrypt or Sign)	1824	1	104

Table 4.12: Counts of Authentication Keys belonging to active users (Past Year)

Cipher	Occurrence	Percentage
ssh-rsa	469803	58.27
ssh-ed25519	322852	40.04
ecdsa-sha2-nistp256	7512	0.93
ecdsa-sha2-nistp521	3654	0.45
sk-ssh-ed25519@openssh.com	1548	0.19
sk-ecdsa-sha2-nistp256@openssh.com	564	0.07
ecdsa-sha2-nistp384	304	0.04

Table 4.13: Counts of GPG Commit Signing Keys belonging to active users (Past Year)

Cipher	Occurrence	Percentage
RSA (Encrypt or Sign)	83939	83.48
Ed25519	14625	14.54
DSA	887	0.88
RSA Sign-Only	718	0.71
ECDSA public key algorithm	382	0.38

Table 4.14: Counts of SSH Commit Signing Keys belonging to active users (Past Year)

Cipher	Occurrence	Percentage
ssh-ed25519	8774	68.91
ssh-rsa	3205	25.17
sk-ssh-ed25519@openssh.com	338	2.65
ecdsa-sha2-nistp256	281	2.21
sk-ecdsa-sha2-nistp256@openssh.com	87	0.68
ecdsa-sha2-nistp521	28	0.22
ecdsa-sha2-nistp384	19	0.15

4.4 World of Code

The World of Code project is a collection of data from various git platforms. The project contains information from over 200 million public GitHub repositories. We leveraged this data to answer questions about general commit signing on GitHub. Users with signing keys do not have to use them. The presence of keys is not necessarily indicative of actual commit signing.

For an overall analysis of how prevalent commit signing is on GitHub we performed a scan of 366,877,605 commits stored in the World of Code database and found that 57,967,631 of these commits are signed. This shows a commit signing rate of around 16%. This far outweighs the 1.2% of users who own signing keys. This is in line with the 13% of active developers who have commit signing keys. Considering the low contribution rate of the average GitHub user, and the fact that users with signing keys contribute more on average than any other users, their over representation in terms of actual committed data is reasonable.

It should also be noted that commits can be signed by GitHub themselves when made from the browser. There are two systems for signing commits from the browser, either via the quick edit function or using GitHub's browser based IDE. This could account for some portion of the signing commits found in World of Code.

Chapter 5

Discussion

5.1 Implications

We now have enough information to answer our original three research questions and their impact on GitHub security. Using this information we can make claims about the security of the platform and suggest possible changes to better the safety of Git systems like GitHub.

Research Question 1. What is the quality of Authentication keys on GitHub? The majority of user login keys are secure, with around 0.4% of users having weak keys. GitHub accounts are mostly inactive, over 90% of users do not contribute code, and over 60% of accounts have not been used in the past four years. This wealth of inactive accounts could pose an issue. Users with keys are significantly more likely to be active on GitHub, with higher rates of participation in all categories. This makes the integrity of these users keys critical as they are the ones pushing code to the platform. While the percentage of weak keys are low, they can be sought out by users using available GitHub resources.

Research Question 2. How many users have signing keys and how are they using them? Around 1.2% of users have any form of signing key. GPG keys are seven times more popular than SSH keys. GPG keys are also the weakest class of keys

overall, and this can be largely attributed to the fact that the security standard for GPG keys has not been updated while the security standard for SSH keys has.

Research Question 3. How many commits are signed? Around 16% of commits are signed. While few overall users have commit signing keys, the users who do commit substantially more. This is attributable to a few factors, most accounts do not commit code. The majority of code committed to GitHub comes from around 10-20% of its user base. Those users with keys are much more likely to be committing code, otherwise they would have no use for keys. This makes these users far more representative.

5.1.1 Idea: User Tags

GitHub is a broad platform that has come to serve a variety of purposes. While its primary goal is to enable collaborative code development, it has adopted many other use cases during its lifetime. GitHub is a large scale discussion forum with millions of users making issues and conversing about code in repositories. It is a distribution platform that hosts ready to use executables, websites, and public code repositories that can be pulled and built. This creates a situation where many users have accounts on GitHub for purposes other than code development. The lack of engagement and quantity of inactive users on GitHub is some cause for concern. Trusted developers who make real contributions to projects are the most critical people on GitHub.

We worry about the security of developer accounts as they are directly responsible for producing software that is used by others. If these accounts are stolen or successfully impersonated they could be used to distribute malicious code on a large scale. User accounts that produce little to no code do not have nearly as much reach and are less important to secure. Being able to tell the difference between these types of users could be beneficial to the platform. This allows for trusted users to be identified quickly and could be used for prevention or warning when non-users make

contributions. This trusted identification could be established through a variety of user data such as commit history, account age, or metrics gained from the community such as user stars.

5.1.2 Idea: Automatic Verification

One large problem with this user trust solution is it requires that commit metadata to be trusted. This is not possible in GitHub's current state as commit verification is still low. While around 16% of real commits are signed, this is not enough to construct a trusted framework. Any system that indicates account activity can be easily faked by creating a large amount of doctored commits.

There is an avenue for large scale commit signing. One of the concerns with digital signing is forcing users to maintain secret keys. GitHub already requires the use of SSH authentication keys, and allows the use of these keys for the purpose of commit signing. Developers on GitHub are already maintaining secret keys capable of commit signing. GitHub could use these SSH authentication keys to automatically sign any commits coming from that user. A system like this would require no additional user effort to implement and would use keys and systems already present on GitHub. Success could be measured quickly by looking at snapshots of commits signing rates compared to authentication key ownership over time. Higher signing rates when the proportion of key ownership stays the same would indicate meaningful progress.

There are other potential methods for user verification. Requiring the use of verified emails for commit identification would help to ensure the reliability of commit data. GitHub does already require the use of real emails when signing up for an account and scans to ensure accounts are legitimate. However, since this information is stored locally on user machines and is pushed to GitHub via Git commands, it would require additional analysis of commits. Commits would have to have an additional verification when being pushed to remote repositories or would have to be scanned after the fact. Either solution would impact the usability of the platform.

5.1.3 Idea: Security Consistency

While commit signing keys are not commonplace, the users who do have them are significantly more active. The integrity of these users is important as they have a proportionally large impact on the platform. Github does not vet the security of signing keys as frequently as authentication keys. This has led to commit signing keys being weaker overall when compared to authentication keys. SSH signing keys are significantly more secure than GPG signing keys. We have found that users are still creating weak keys in 2024. While these weak algorithms may be secure enough for the average user at the moment, many of them could become vulnerable in the near future as resources and attack methods improve. In order to increase security on GitHub their standards should be consistent across all types of key. GPG key ciphers should be curated as frequently as SSH key ciphers. GPG keys are the most weak on the platform and show no indication of improving without additional input from GitHub.

5.2 Future Work

There is still significant work to be done in this field. User key management is a valuable topic of study. Git platforms such as GitHub are responsible for some of the largest networks of personally managed keys. Additional data can be collected from GitHub as well as from other sources. We also can take a more granular approach now that we have an understanding of normal user interaction vs the most extreme user interaction. This allows us to conduct personal interviews with users of interest to garner an understanding of their usage patterns.

5.2.1 Big Data

Finishing key and user information collection is the next immediate goal for this research. The limitations of collection from the GitHub API make data aggregation

slow. Request limits heavily bottleneck data collection speed. While the GraphQl system allows us to gather some information more effectively, it has its own limitations and does not allow the collection of all required data. Increasing the bandwidth of our collection would only be beneficial to the project. This can be done by finding ways to increase GitHub API access, either through direct appeal to GitHub itself, or increasing the number of user accounts available to the collection script. Once full data collection is complete, we can draw definite conclusions about the state of security on GitHub. The collection of additional user information will also allow us to perform more contextual analysis of keys.

Alternatively, we can look towards other means of data collection. Gathering large amounts of user commits and repositories via the GitHub API is not possible in a reasonable time frame. Further integration with the World of Code project to allow for analysis of the code contained in signed commits. World of Code contains a near complete snapshot of public GitHub repositories. It can be used in conjunction with our collected data to perform a finer grain analysis of signed commits, repositories, and authors. World of Code will also allow us to see what keys exist in actual commits and find any instances of keys that are not accounted for by GitHub. The data on World of Code is hosted on a membership based server. This allows for much faster data analysis.

We are currently working with the World of Code platform and will continue to gather information from it in the coming months. Deeper investigations of the platform can also be performed after full scale data collection is completed. This includes analysis of actual commit content, repository information such as number of developers, and percentages on which developers are producing the most code. This information would allow us to investigate the scale of code contribution of developers who sign commits and those who do not.

5.2.2 Interviews

The average person does not like to maintain cryptographic systems. We now have a large list of developers who do maintain extra security keys. Many of these users provide additional contact information on GitHub. Performing a study of these users would help to provide actual user opinion on these key systems. There are a variety of methods we could use to question these developers. User emails can be attained through two methods - the emails they provide on their account summary, or the email they use to sign commits. This is not a comprehensive approach as account emails are optional, and commit emails are not verified and may not be tied to an actual inbox the user can receive. Users can also link social media accounts that can be used to make contact. Large studies could also be made using interview platforms such as Qualtrics. Any of these solutions will have disadvantages, so a mixture of methods might be more comprehensive.

Large scale email campaigns for every user who provides one or has an email linked to a real commit would be one option for contact. This is well suited to survey style interview where the targeted users simply have to fill out a short survey asking questions about their usage habits and opinions on Git platforms. This automatic method would allow for quick collection of a large amounts of data, but is restricted in the depth of effective information it can collect. Another concern with this cold call approach is having the emails be considered spam or malicious and ignored.

To supplement this, developers at the extreme end of key ownership or contribution to the platform could be contacted and interviewed directly by members of our lab. We would consider user in the top percentage of activity on GitHub for a variety of metrics such as commit activity or number of keys owned. Then interview them remotely via some online platform such as Zoom. This approach allows us to ask more complex questions about their opinions and usage on the platform. This would reveal reasoning behind user sentiment around these cryptographic systems and help us to understand what kinds of systems these users would like to engage

with. Conducting personal interviews allows us to thoroughly examine user sentiment and acquire results that are otherwise impossible with automated systems. While we would still be reaching out to these users via their linked email accounts, ideally they would be more receptive as the emails sent to them would be personalized and sent by a real person, rather than automated.

5.2.3 Contribution

One final goal of this project will be to make the data we have collected publicly available for further use by other researches. Sharing collected information allows for our work to be repeated for academic review, and built upon for further research. One concern with many of the current works looking at GitHub is that most papers are building data sets from scratch every time. Allowing our data to be used by others helps to progress our understanding of the field.

Chapter 6

Conclusion

Github is the most popular code development system in the world. Millions of developers add code to this platform everyday. Applications, websites, and other software are deployed from GitHub all the time. Security and user trust on the platform is critical. Authentication and commit signing keys are the solution to this problem. Users who utilize both are able to securely upload data to repositories, and verify they were the ones who did so. The proper management and usage of these keys is required to build secure access control on GitHub.

We analyzed a significant portion of the users on GitHub to collect information on their keys and how they interacted with the platform. We found that the majority of accounts on GitHub are not active or never contribute code. Only a small percentage of users create any form of key. The users who do create signing keys contribute more to GitHub on average.

Keys on GitHub are mostly secure, and majority of users on the platform use well known secure algorithms. Adoption of stronger and faster algorithms like Ed25519 is increasing across all categories. However, there are an amount of keys on the platform with a low security strength. A small percentage of weak keys are on the platform for both signing and authentication. Github does update the security requirements for keys and has phased out weaker key algorithms over time. However, these standards

have not been applied to all classes of key. As a result weak keys are still allowed to be generated after Github recent security updates.

Analysis of data from World of Code has shown that around 16% of commits are signed. A majority of the code on GitHub comes from a small percentage of users. Users with security are much more likely to be contributing to the platform than those without. As a result we see that their contributions make up a significant portion of actual code.

Understanding why users choose to sign commits and how they interface with critical systems like Git is necessary for the construction of usable secure systems. Analyzing key usage on a large platform like GitHub allows us to form a better image of how the average developer interfaces with security.

Bibliography

[1] Albrecht, M. R., Degabriele, J. P., Hansen, T. B., and Paterson, K. G. (2016). A surfeit of ssh cipher suites. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1480–1491, New York, NY, USA. Association for Computing Machinery. 11

[2] Ali, Y. and Smith, S. (2004). Flexible and scalable public key security for ssh. In *European Public Key Infrastructure Workshop*, pages 43–56. Springer. 11

[3] Arkhipkin, D., Betts, W., Lauret, J., and Shiryaev, A. (2008). An ssh key management system: easing the pain of managing key/user/account associations. In *Journal of Physics: Conference Series*, volume 119, page 072005. IOP Publishing. 11

[4] Benedetti, G., Verderame, L., and Merlo, A. (2022). Automatic security assessment of github actions workflows. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, SCORED'22, page 37–45, New York, NY, USA. Association for Computing Machinery. 11

[5] Cosentino, V., Luis, J., and Cabot, J. (2016). Findings from github: methods, datasets and limitations. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 137–141, New York, NY, USA. Association for Computing Machinery. 10

[6] Feng, R., Yan, Z., Peng, S., and Zhang, Y. (2022). Automated detection of password leakage from public github repositories. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 175–186, New York, NY, USA. Association for Computing Machinery. 10

[7] Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., and Yu, J. (2023). Security weaknesses of copilot generated code in github. 10

[8] Garfinkel, S. L., Margrave, D., Schiller, J. I., Nordlander, E., and Miller, R. C. (2005). How to make secure email easier to use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, page 701–710, New York, NY, USA. Association for Computing Machinery. 11

[9] Gasser, O., Holz, R., and Carle, G. (2014). A deeper understanding of ssh: Results from internet-wide scans. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. 11

[10] Ghiotto, G., Murta, L., Barros, M., and van der Hoek, A. (2020). On the nature of merge conflicts: A study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering*, 46(8):892–915. 9

[11] Horawalavithana, S., Bhattacharjee, A., Liu, R., Choudhury, N., O. Hall, L., and Iamnitchi, A. (2019). Mentions of security vulnerabilities on reddit, twitter and github. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, page 200–207, New York, NY, USA. Association for Computing Machinery. 10

[12] Johnston, C. (2010). Ps3 hacked through poor cryptography implementation. 7

[13] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 92–101, New York, NY, USA. Association for Computing Machinery. 10

[14] Kapadia, A. (2007). A case (study) for usability in secure email communication. *IEEE Security & Privacy*, 5(2):80–84. 11

[15] Lausch, J., Wiese, O., and Roth, V. (2017). What is a secure email. In *European Workshop on Usable Security (EuroUSEC)*. 11

[16] Ma, Y., Bogart, C., Amreen, S., Zaretzki, R., and Mockus, A. (2019). World of code: an infrastructure for mining the universe of open source vcs data. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 143–154. IEEE. 11

[17] Ma, Y., Dey, T., Bogart, C., Amreen, S., Valiev, M., Tutko, A., Kennard, D., Zaretzki, R., and Mockus, A. (2021). World of code: enabling a research workflow for mining and analyzing the universe of open source vcs data. *Empirical Software Engineering*, 26:1–42. 11

[18] Martinez, M. and Monperrus, M. (2019). Coming: A tool for mining change pattern instances from git commits. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 79–82. 9

[19] Meli, M., McNiece, M. R., and Reaves, B. (2019). How bad can it git? characterizing secret leakage in public github repositories. In *NDSS*. 9

[20] Microsoft (2023). Ca5384: Do not use digital signature algorithm (dsa). 37

[21] of Standards, N. I. and Technology (2020). Nist special publication 800-57 part 1 revision 5. Technical report, U.S. Department of Commerce, Washington, D.C. 18

[22] of Standards, N. I. and Technology (2023). Implementation guidance for fips 140-2 and the cryptographic module validation program. Technical report, Canadian Centre for Cyber Security, Vanier, Canada. 18

[23] Overflow, S. (2022). Stack overflow 2022 developer survey. 1

[24] Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., and Karri, R. (2022). Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. 10

[25] Pletea, D., Vasilescu, B., and Serebrenik, A. (2014). Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 348–351, New York, NY, USA. Association for Computing Machinery. 10

[26] Ruoti, S., Andersen, J., Zappala, D., and Seamons, K. E. (2015). Why johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *CoRR*, abs/1510.08555. 11

[27] Ruoti, S. and Seamons, K. (2019). Johnny's journey toward usable secure email. *IEEE Security & Privacy*, 17(6):72–76. 11

[28] Sarma, A., Redmiles, D. F., and van der Hoek, A. (2012). Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, 38(4):889–908. 9

[29] Vale, G., Hunsen, C., Figueiredo, E., and Apel, S. (2022). Challenges of resolving merge conflicts: A mining and survey study. *IEEE Transactions on Software Engineering*, 48(12):4964–4985. 9

[30] Xia, H., Pei, Q., and Xi, Y. (2016/11). The analysis and research of freak attack based on openssl. In *Proceedings of the 6th International Conference on Information Engineering for Mechanics and Materials*, pages 15–19. Atlantis Press. 6

Vita

Parker Collier was born in June of 2000, in Nashville, TN. He graduated St. Patrick's middle school in 2015, and went on to graduate from Waverly Central High School in 2019. Parker then enrolled in the University of Tennessee at Knoxville and received his Bachelors of Science in May of 2023. During his time at UTK he worked for three research labs: MABE lab under Dr. Xioapeng Xiao, MOA lab under Dr. Doowon Kim, and USER lab under Dr. Scott Routi. He will graduate with his masters in May of 2024.