applied sciences

MDPI

*Article*

# Password Guessability as a Service (PGaaS)

**Juan Bojato, Daniel Donado, Miguel Jimeno, Giovanni Moreno and Ricardo Villanueva-Polanco ***

Department of Computer Science and Engineering, Univeresidad del Norte, Barranquilla 081007, Colombia; jbojato@uninorte.edu.co (J.B.); adavendano@uninorte.edu.co (D.D.); majimeno@uninorte.edu.co (M.J.); cgmoreno@uninorte.edu.co (G.M.)
* Correspondence: rpolanco@uninorte.edu.co

**Abstract:** This paper presents an adaptable password guessability service suited for different password generators according to what a user might need when using such a service. In particular, we introduce a flexible cloud-based software architecture engineered to provide an efficient and robust password guessability service that benefits from all the features and goals expected from cloud applications. This architecture comprises several components, featuring the combination of a synthetic dataset generator realized via a generative adversarial network (GAN), which may learn the distribution of passwords from a given dictionary and generate high-quality password guesses, along with a password guessability estimator realized via a password strength estimation algorithm. In addition to detailing the architecture's components, we run a performance evaluation on the architecture's key components, obtaining promising results. Finally, the complete application is delivered and may be used by a user to estimate the strength of a password and the time taken by an average computer to enumerate it.

**Keywords:** password guessability; service; key-rank estimation problem; generative adversarial networks



**Citation:** Bojato, J.; Donado, D.; Jimeno, M.; Moreno, G.; Villanueva-Polanco, R. Password Guessability as a Service (PGaaS). *Appl. Sci.* **2022**, *12*, 1562. https://doi.org/10.3390/app12031562

Academic Editors: Luis Javier Garcia Villalba, Rafael T. de Sousa, Jr., Robson de Oliveira Albuquerque and Ana Lucila Sandoval Orozco

Received: 30 November 2021
Accepted: 24 January 2022
Published: 31 January 2022



## 1. Introduction

Online passwords security is an ongoing security problem due to constant attacks on online exposed services and the lack of safe behavior from users while connecting to those services, especially when users need to create and protect their passwords. Password guesser tools are improving their capabilities, and nowadays, attackers do not need large leaks to guess the rest of the passwords [1]. Although today, multi-factor authentication systems have emerged as a more secure alternative, password authentication remains either as the only factor used by many users or as one of the factors of multi-factor systems [2]. Any online information system must have an appropriate security level to prevent an intruder from accessing the information. Users tend to choose their passwords as combinations of characters that are easy for them to remember, such as names, dates, and phone numbers, regardless of how vulnerable passwords can be or of their knowledge about how passwords might be cracked [3,4]. As a result, attackers or hackers use techniques and tools for breaching and cracking passwords [5]. This problem represents a vulnerability to be exploited. Security companies have implemented mechanisms to improve password security. Passwords must have two fundamental properties: (1) be hard to guess, to make them secure against brute-force attackers and (2) be easy to remember, so users do not rely on third-party tools to remember them. Therefore, it is necessary to develop a tool that allows the users to know how secure their password is, i.e., a tool a user can utilize for estimating the strength of a chosen password. Additionally, such a tool should leverage the best available resources to give the appropriate feedback to the users. As evidenced by the literature discussed in Section 2, a large number of current password-strength estimators employ large but static and context-unaware password datasets and present limited usability.

The desired tool should be built on two premises: (1) learn from frequently updated datasets to reflect changes in users' predilections for password creation, and at the same time (2) give feedback about how common or secure a tested password is. There are some well-known password-guessing tools such as John the Ripper [6], and Hashcat [7]. Such tools, however, are not easily integrated as a service for companies and other services to integrate into their software. Part of a good password security validation tool is the database it uses to feed its algorithm before evaluating the given password and the algorithm used to validate it. The proposal from [8] called PassGAN uses neural networks to create valuable databases, and the algorithm in PESrank [9] ranks the given password as feedback to the user. The tools mentioned above offer excellent approaches to both problems but do not mix their solutions for an optimal approach. For this reason, this paper proposes a mechanism and architecture to strengthen passwords by measuring their degree of strength to reduce the vulnerabilities that arise from malicious attacks. The proposal creates a set of passwords with human characteristics and classifies user passwords by their strength, respectively. This paper aims to build an architecture with modular components that include models and password-generation algorithms. It will train models to generate dictionaries of passwords consistent with business systems.

The main contributions of this paper are the following:

- The first adaptable password guessability service offered as a cloud-based service to benefit from all the features and goals expected for cloud applications, e.g., scalability and availability. In particular, this paper presents a flexible cloud-based software architecture engineered to provide an efficient and robust guessability service. The service leverages one of its components to frequently generate synthetic password datasets, which are passed as inputs to the password guessability estimator component to adapt itself to the evolution of the password policies within an organization
- The presented password guessability estimator can be tailored to different languages as needed, thanks to the dynamic, synthetic password datasets. This result is an improvement concerning how password strength estimators usually work, as evidenced by the literature discussed in Section 2, since they may employ large but static and context-unaware password datasets.

This paper is structured as follows. In Section 2, we present background material on security as a service and password security. Additionally, we present the previous works on password strength estimators. In Section 3, we present our modular architecture for our proposed guessability service, detailing the design principles behind it and its components. Section 4 details the main algorithms used for implementing the critical components of our architecture. In particular, we focus on the synthetic dataset generator component and the password guessability estimator component. Section 5 presents results about the performance of our implementation of the proposed architecture. Lastly, Section 6 encloses our final comments on the paper, highlighting some future research works.

## 2. Background and Related Work

This section presents background material on security as a service and password security, as far as security issues regarding a user's interaction with edge networks, which is one of the main drivers of this research. Additionally, we present previous works on password strength estimators. In general, edge networks and Internet of Things networks interact with users frequently, demanding a secure authentication based on single or multi-tier authentication methods. The work in [10] presented a survey on remote user authentication and issues for networked communication in general. The authors discussed existing remote user authentication mechanisms approaches, including steps for registration, logins, session key agreement, and password change. Our work focuses on the issues around the login step, where the authors assumed a single-step process. The Cloud of Things is a concept recently proposed to describe the use of cloud computing by the multiple devices composing the Internet of Things (IoT). The work in [11] presented a multi-factor authentication system to be deployed in the Cloud of Things and uses secret splitting. Such

approaches might require updates in the security components of software architectures on both sides of the communication. Our single authentication approach relies on traditional and straightforward processes that would not require changes.

### 2.1. Security as a Service

There are academic and commercial solutions where security functionalities as services are the architectural design of the product. The work presented in [12] proposed a set of tools that architects could deploy in the cloud and offer as a service. This solution builds from a set of functionalities that the cloud provider could offer their clients (called cloud tenants), which deploy virtual machines with their applications where the security service is also running. The presented work used virtual machines, while our work used simple processes that do not depend on virtual machine deployments. This approach gives freedom for lightweight transition between environments. Another work presented in [13] delivers a security service that cloud tenants could use to monitor and protect their application environment deployed in the cloud virtual machine. Their service concept is a set of functionalities that fulfill the tenants' security needs while achieving the efficiency, flexibility, and low-cost features expected for a cloud service. The work of [14] proposed a series of cloud-native design patterns based on the use of micro-services to design security as a service architecture that could scale their computational capabilities depending on the demand peaks that some security events could trigger. Their service is designed to attend vulnerability assessment requests from cloud tenants. The works mentioned above all used the same idea of offering features through cloud-based services, giving modern cloud applications versatility. The same approach was vital in our proposal as it allows for readiness, vendor independence, and ease of deployment.

A specific group of academic work focuses on offering security services for resource-restrained networks. Thanks to the dissemination of 5G networks, edge computing, and massive Internet of Things (IoT), these networks are becoming popular. One such work is presented in [15], where the authors designed a set of services that a slice of virtual network could offer to their tenants. In 5G networks, there are virtual divisions created within networks, called slices. Furthermore, the authors created an architecture for offering security services within the slices. The work of [16] describes an intrusion detection system that uses machine learning algorithms to detect anomalies in IoT networks. Given the characteristics of IoT networks, the authors offered real-time detection capabilities for their service. These services are offered on top of cloud services that usually support IoT network communication protocols. This example shows how security services are being deployed for multiple purposes using similar approaches to the one proposed in this paper.

### 2.2. Password Security

The main objective of a password is to restrict the access of unauthorized persons to the system, and therefore we always want them to be sufficiently secure [17]. We have two common approaches to model password guessability starting from this definition. One uses large databases from which the strength can be estimated, while a newer approach uses neural networks. One example of the latter is the work proposed in [18]. The authors developed a Javascript-based tool that uses a small print neural network model for password guessing. As stated by the authors, such an approach generates and compresses a neural network model that users will use on the client's side. However, the model would need to be retrained and the compressed version updated to support modification to the learning dataset. Modifications might arise from how users create their passwords and might arise from the language used by users to write their passwords. The former approach uses password databases as a source for strength-guessing algorithms. One such tool is PESrank, which is in charge of estimating the password's strength through ranges that allow us to determine how likely a password is to be found. All passwords are decryptable. However, having a higher strength will cause attackers to take more resources and more time to extract them. When compared with both approaches, the proposal from this paper uses

dynamically updated databases to reflect possible changes that might arise from language or password creation styles.

We can find dictionary attacks and brute force attacks among the most common attack techniques. A dictionary attack prioritizes attack speed, although this leads to a smaller spectrum of possibilities, while a brute force attack covers many more possibilities by considerably reducing speed. A dictionary attack uses a set of common words to be tested, while brute force tests all possible character combinations to form character strings [19]. It is possible to use both techniques according to the information available about the passwords, such as a particular pattern that can be replicated. Dictionary attacks are the most commonly used because they help save time, and dictionaries are created with leaked and common passwords that different people have used. Thanks to this, researchers have created many dictionaries over the years with which these types of attacks are carried out. Recently, applications that generate password dictionaries such as PassGAN are becoming known. Side-Channel Attack (SCA) is one of the easiest and most potent methods to execute against cryptographic implementations, and their targets range from primitives, protocols, modules, and devices to uniform systems [20]. They pose a severe threat to the security of cryptographic hardware products. In this way, solutions to key enumeration problems and rank estimation problems are used to carry out these attacks.

### *2.3. Password Strength Estimators*

The work presented in [21] proposed a Password Strength Estimator that uses password ranking (PESrank), which according to authors models the behavior of powerful password crackers. It focuses on giving users feedback regarding the security ranking of the given password, which gives an idea to the person of how good or bad it is compared with others. PESrank returns the rank in fractions of a second to be helpful as an online tool. The authors showed that their tool could be modified to personalize the model used for ranking according to the application's particularities. The authors used a set of 905 million passwords to train the model. It is important to note that although it is a large set, situations might change such that the train sets no longer reflect the most current trends in password creation.

Another work is presented in [22], where the authors proposed a lightweight and fast strength estimator, and the authors tested it on different platforms. They claimed that their tool is enough to predict best guessing attacks accurately and conservatively. The work presented in [23] created an estimator that trained against a leaked list of 14.3 million passwords. Such a list is used by password-cracking tools. The authors consider the importance of including leaked passwords in the estimation process to improve the accuracy of such tools. Although this tool is trained on a large dataset, it is still considered a tool that feeds from static information. Our work deals with this issue by relying on constantly updated datasets to adjust the estimations when needed.

The authors of [24] presented a multi-modal password strength estimator where they combined unique techniques to exploit their advantages, thus overcoming the weaknesses of each. The authors stated the necessity of designing a flexible estimator that could adapt to the specification of each application environment, for example, how the language used for password creation should modify the tool's estimation. The concept of flexibility is critical in their proposal, but they tackle it from the application's perspective. On the other hand, our proposal assumes that strength estimation should change as password creation customs change.

There is a trend concerning the design of lightweight estimators that could be deployed on light clients, such as that proposed in this paper. Such an example is presented in [25]. The authors proposed a light password strength estimator with an accuracy significantly higher than popular tools such as the zxcvbn [23]. In addition, their tool is light at 33 KB and as fast as 0.18 ms, making it suitable for light clients. This tool uses a method that measures password strength by evaluating the similarity between a password and a standard strong password as selected by their methods. The work presented in [18] presented

a neural-networks-based model that runs on JavaScript and is deployed on the clients' side without substantially affecting the accuracy of their algorithm. Their approach shows how researchers might decide to sacrifice accuracy when looking for lean implementations. However, our approach does not sacrifice accuracy while maintaining a light footprint at the running device, possible thanks to the microservice architecture design.

The adaptability of estimators is put to the test when different languages are used. A particular study [26] showed the importance of adapting estimators to native languages, as the authors demonstrated in South Africa. This adaptation is one of the critical reasons behind the creation of our proposal, as language-adapted databases could improve the tool's accuracy. On the other hand, the work in [27] showed how demographics change the password creation process. It is expected that users with less Internet use proficiency might not be aware of the security issues derived from poor password creation. On the contrary, Internet-savvy users might be more aware and willing to create stronger but easy-to-remember passwords, thus creating different distributions that might change according to age ranges. Other factors than age also influence demographics, thus creating different password creation patterns. All of these factors highlight the importance of dynamically updated databases. The authors of [28] addressed the language challenge to adapt zxcvbn to Czech and Slovak languages. As a result of the adaptation, the estimation improved against a set of leaked Czech passwords and improved the execution time. The authors highlighted the importance of mixing existing English dictionaries with Czech dictionaries to improve accuracy. These are examples of the importance of using adaptable datasets. While traditional literature uses extensive datasets with more than 100 million passwords to test against, such datasets are built using traditional English passwords, which might decrease accuracy when tested against other languages such as Spanish.

## 3. PGaaS Architecture

This section presents our modular architecture for our proposed password guessability service. The architecture's objective is to offer a security service for password security assessment that uses exchangeable components that can grow or change as the companies using the architecture might need. To achieve this objective, we chose a cloud-based infrastructure, given the multiple advantages the cloud offers, and the design offers the functionality as a service, which is a well-used model nowadays on the Internet. First, we describe the design principles behind the proposed architecture, its main components, and finally, the interactions among the components.

### *3.1. Design Principles*

The design principles behind the design of our architecture are based on the microservice architecture principles [29,30]. The principles focus on achieving a set of predefined goals for the enterprise. Microservices are lightweight and focused software services deployed in the cloud that can be integrated to create new composing services [31]. The reason for choosing microservices is that they facilitate many of the expectations of cloud applications, which include elasticity, heterogeneity, and distribution. Some principles govern such applications, which help researchers design architectures that meet cloud-based applications' functional and non-functional requirements.

The principles are as follows:

- Architects should use fine-grained interfaces or, when possible, no interfaces. Fine-grained interfaces focus on using the RESTful paradigm, usually through HTTP APIs. The paradigm allows using HTTP as the communication protocol for exchanging resources between service providers and users. When possible, automation should be achieved to establish communication between services.
- Cloud-native application design principles (isolate state, distribution, elasticity, automated management, and loose coupling) [32]. These principles establish the goals for keeping the cloud services as available as possible with the highest quality standards.

All microservice architectures today rely on cloud deployments, thus demonstrating the necessity of complying with typical cloud-oriented functional requirements.

- Architects should use multiple computing and storage paradigms. One of the key features of cloud and microservices architectures is orchestrating the services to achieve an easy configuration and management of many deployed services. Orchestrating services is necessary when there are many heterogeneous paradigms available. However, all the services running such paradigms must coincide with standards to exchange information, such as using RESTful interfaces with HTTP and JSON protocols and notations.
- The architecture should guarantee lightweight deployment for easy reconfiguration. Cloud applications use lightweight infrastructure such as containers instead of virtual machines to achieve lightweight deployment. Such decisions allow easy reconfiguration and fast deployment.
- Decentralized continuous delivery. Security as a service must offer functionality with the highest levels of availability. In the particularity of our proposal, the absence of a security validation service might create insecure user passwords for as long as the service is not available. Thus, architects need to guarantee that this principle is applied to their solution.
- Lean DevOps. By definition, DevOps is a set of standards, tools, and practices that combines the software development steps with IT operations. The purpose is to decrease the development life cycle and provide continuous delivery based on high quality [33]. Every cloud-based development should focus on making DevOps operations as light as possible.

### *3.2. Architecture Components*

The architecture is based on a cloud or edge-computing architecture and is shown in Figure 1. The figure shows several layers that can be implemented in a distributed and heterogeneous fashion. Architects could combine multiple cloud platforms and service back-ends while combining them using typical service orchestration. An example of such a tool is presented in [34]. The presented tool facilitates the creation of complex and distributed microservice data flows in cloud services.

The main components of this architecture are listed below:

#### 3.2.1. Load Balancer

The load balancer is a critical component of every cloud-based software architecture that implements all the previously defined principles. The balancer attends and distributes the service requests to one of the available API instances in the layer below. Typically, balancers play the role of proxies to hide from the users the real distributed fashion of the architecture. All layers below can be multiplied to achieve elasticity.

#### 3.2.2. REST API

This component receives a request sent to the service and routes it to the handler function to be further processed. Architects use REST technology to offer a standardized interface mechanism for users to connect. The API relies on the HTTP protocol to expose the functionality of the security service. The RESTful design takes each functionality and offers it to the users as a resource available through independent endpoints built using URLs. For example, using the POST command from HTTP connected to the appropriate endpoint, users can send the created password and obtain the rank of the given password as the response.

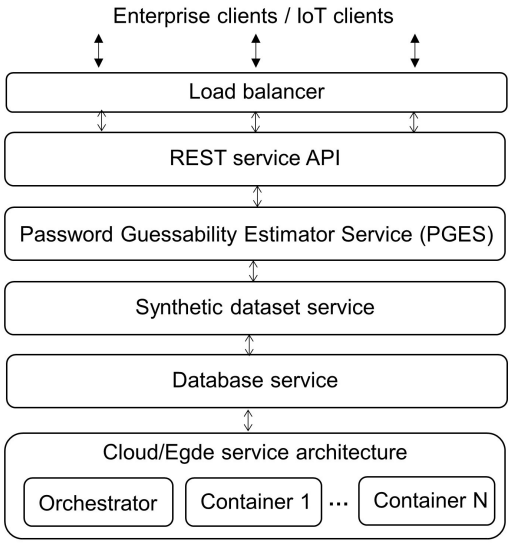


**Figure 1.** Proposed architecture.

#### 3.2.3. Password Guessability Estimator Service (PGES)

This component serves as an estimator to evaluate how guessable a given password is. In particular, this component will estimate the rank of a given password in a long list of passwords, i.e., a dictionary. In more detail, it provides a list of passwords in decreasing order from the most easily guessable to the least, and a password; the service will attempt to estimate the position of this given password within the given dictionary. This component requires its list of passwords to be updated constantly, i.e., it needs an updatable dictionary. This feature aims to make the guessability evaluations returned by this component as accurate as possible. The Password Dataset Generator Component performs these dictionary updates. In Section 4.2, we will expand on the rank estimator we implemented and its inner workings.

#### 3.2.4. Synthetic Dataset Generator Service

This component serves as a generator of lists of passwords. This component is also offered as a service. The input of this service is the parameters to train a Generative Adversary Network (GAN) to build sets of passwords that will serve as input for the PGES, as explained above.

#### 3.2.5. Database Service

The database service can be considered as a Dabatase as a Service (DBaaS), which is the most practical and service-oriented approach to use in similar architectures [35]. The DBaaS allows the programmers to think less about the low-level details of the database implementation, providing a technology-free interface to store and retrieve data. Architects could define the interface to receive only name–value pairs or elaborated queries for the storage and retrieval of information.

#### 3.2.6. Cloud/Edge Service Architecture

This component represents the infrastructure of all the architecture components. The most lightweight option for deploying services independently regardless of their dependencies is by using containers [36]. Containers are a solution against the disadvantages of using virtual machines, where the space and computational requirements in the physical servers are heavy when the interest is to scale quickly and cheaply. The use of containers allows for rapid service scalability as needed. Each container stores each of the services depicted above independently, which means one service on its container. For example, an architect might decide that the best database service is based on a previously

configured database container with the secure configurations requested by the company where the service is being deployed. Containers can quickly be turned on or off as needed, and the specific characteristics of a peak in demand might imply that only containers for some services are replicated, while others remain the same.

Orchestration tools manage the deployment of a typical container. Service orchestration enables the cloud service providers and application owners to execute a set of operations for selecting, deploying, monitoring, and dynamically controlling the configuration of the computational resources of any deployment [37]. Examples of such tools are Kubernetes [38] and Docker Swarm [39]. Kubernetes is a tool itself, as it is a container-orchestration system for automating the deployment, scaling, and management of applications running on containers. On the other side, Docker Swarm is an option on the Docker tool to allow managing and orchestrating sets of containers.

Another aspect of the architecture is that the design allows deployment in an edge computing layer, usually designed for serving IoT devices. The edge is considered a layer closer to the devices connected to the security service but not properly on the cloud. The hardware configuration of systems that build the edge are usually resource-restrained devices participating in the IoT network, but with links to the cloud where heavy resource-oriented features and full-feature services are offered.

### *3.3. Components Interaction*

The interaction of the components is guaranteed by one of the principles stated above, which are the lean DevOps operations that any software team should apply today. Those in charge of the different components should interact through tools and good practices to maintain clean architecture deployment. Implementing all the other principles, such as a lightweight deployment, is essential. The interaction is also guaranteed through a standardized implementation of common microarchitecture principles, which call for a loose connection between services and standard and light communication protocols.

The component interactions require a safe platform. Security as a service must rely on a secure architecture to guarantee the availability and trustworthiness of what is offered. From the cloud/edge architecture, it is possible to guarantee a secure deployment and administration of the containers deploying the services that constitute the architecture. The work presented in [40] for example, presents a secure container mechanism for Docker that uses a software guard extension built into Intel CPUs, called Intel SGX. This mechanism aims to protect processes running on top of the containers from outside attacks. The proposal has a small footprint on the running CPUs. There is a need for a secure communication layer to ensure no outsiders can interfere from this layer up to the others.

## 4. Architecture Implementation

This section describes the implementation of the most relevant components of our architecture. We first give details of the dataset generator component's implementation and then the password guessability estimator component.

### *4.1. Synthetic Dataset Generator Component*

In order to compute a guessability measure for a given password, a dictionary of passwords is needed, and it may vary following a password policy organized within an organization. To obtain the password dictionaries, we leveraged PassGAN (GANs) [8], which is a generative adversarial network able to generate passwords. These Generative Adversary Networks (GAN) pose the training process as an adversarial game between two networks: a generating network that generates samples and a discriminatory network that tries to classify the samples as coming from the true distribution or the model distribution. Every time the discriminator notices a difference between the two distributions, the generator slightly adjusts its parameters to make it disappear, until in the end, in theory, the generator exactly reproduces the true data distribution, and the discriminator is randomly guessing, unable to find a difference [41].

GAN networks are typically used for imaging, but PassGAN is designed to learn the distribution of password information from filtered passwords, and generates passwords without user intervention, the same as those used by people. PassGAN has an open-source implementation, and researchers can experiment with it using various pre-trained models, such as the famous RockYou dictionary, and thus generate sets of passwords. Likewise, it can also be trained with its data set to generate them. This component may therefore be configured to generate passwords similar to a password policy in an organization (i.e., context aware).

In our experiments, we use a dataset of filtered passwords. In particular, we use one from LinkedIn and experiment with PassGAN to select passwords up to 17 characters long and perform 100,000 iterations. The other parameters are left by default. Finally, once PassGAN is trained, we can generate password dictionaries with the desired amount.

### *4.2. Password Guessability Estimator Component*

This section describes how our Password Guessability Estimator Component is implemented. We first describe the key estimation problem, which is the basis for implementing this component, and then we focus on the rank estimation algorithm used in this component.

#### 4.2.1. Key Enumeration and Key Rank Estimation Problem

Let $\mathtt{K}_* = b_0 b_1 b_2 \dots b_W$ be a secret key. Assume it can be represented as a concatenation of $\mathcal{N} = W/w$ chunks, each on $w$ bits, i.e., $\mathtt{K}_* = \mathtt{K}_*^0 \parallel \mathtt{K}_*^1 \parallel \dots \parallel \mathtt{K}_*^{\mathcal{N}-1}$ with $\mathtt{K}_*^i = b_{i \cdot w} b_{i \cdot w+1} \dots b_{i \cdot w+(w-1)}$ where $0 \leq i < \mathcal{N}$. Let us assume there is an algorithm that generates full candidates $\mathtt{C}$ for $\mathtt{K}_*$, such that any full candidate $\mathtt{C}$ can also be represented as a concatenation of chunks $\mathtt{C} = \mathtt{C}^0 \parallel \mathtt{C}^1 \parallel \dots \parallel \mathtt{C}^{\mathcal{N}-1}$. The method of maximum likelihood (ML) estimation then suggests picking as $\mathtt{C}$ the value that maximizes $\Pi_{i=0}^{\mathcal{N}-1} \mathbf{P}(\mathtt{K}_*^i | \mathtt{C}^i)$. Therefore, we can construct $\mathcal{N}$ lists containing 2-tuples of the form $(\mathtt{p}, \mathtt{v})$, where $p = \mathbf{P}(\mathtt{K}_*^i | \mathtt{v})$ and $\mathtt{v} \in \{0,1\}^w$, by estimating $\mathbf{P}(\mathtt{K}_*^i | \mathtt{v})$ for each of at most $1 \leq m_i \leq 2^w$ candidate values $\mathtt{v}$ for $\mathtt{C}^i$ [42].

Let $L^i = [\mathtt{C}_0^i, \mathtt{C}_1^i, \dots, \mathtt{C}_{m_i-1}^i]$ be the list of 2-tuples for chunk $\mathtt{K}_*^i$. To recover the secret key $\mathtt{K}_*$, we require efficient algorithms that traverse the lists $L^i$ to pick 2-tuples $\mathtt{C}_j^i$ and then combine them to obtain full key candidates $\mathtt{C}$ in decreasing order based on their total probabilities (the product of its chunk candidates probabilities). This problem has been addressed previously in the side-channel analysis literature, with a variety of different algorithms able to solve this problem and the related problem known as the key-rank estimation [43–57]. We remark that each method of enumerating the full key candidates defines a new variant of the key-enumeration problem [42] and that these variants arise in many cryptographic contexts [58–63].

The key rank estimation problem, on the other hand, is defined as follows. Given $\mathcal{N}$ sub-key spaces of sizes $m_i$ with their corresponding probability distributions $\mathbf{P}(\mathtt{K}_*^i | \mathtt{v})$ sorted in non-increasing order of probabilities, a key $\mathtt{K}_* = \mathtt{K}_*^0 \parallel \mathtt{K}_*^1 \parallel \dots \parallel \mathtt{K}_*^{\mathcal{N}-1}$ and its probability $p_* = \Pi_{i=0}^{\mathcal{N}-1} \mathbf{P}(\mathtt{K}_*^i | \mathtt{C}_*^i)$, then the key rank estimation problem asks for estimating the number of full key candidates with probability higher than $p_*$, when the probability of a full key is defined as the product of its chunk key probabilities. The idea is to estimate $\mathtt{K}_*$'s rank, i.e., the position of the key $\mathtt{K}_*$ in the list of $\Pi_{i=0}^{\mathcal{N}-1} m_i$ full key candidates when this list is sorted in non-increasing probability order, from the most likely full candidate key to the least likely [9,43–57].

Our password guessability estimator component, explained in more detail in Section 4.2.2, will measure how guessable a given password is as its relative position in a long list of passwords (dictionary) used on the training of the corresponding algorithm. Note that this approach is a natural way to measure how guessable a given password is, since password-cracking tools, such as John the Ripper [6], will attempt to brute force all the passwords within a given dictionary constructed from common passwords, from the most likely to the least.

4.2.2. Guessability Problem

To measure how guessable a given password is, we first cast the guessability problem as a key-rank estimation problem as proposed in [9], and then use a key-rank estimation algorithm (ESrank) [64] to estimate the rank of the given password within a dictionary used on training by the estimator algorithm. The idea is basically to construct probability spaces as introduced in [9] and then estimate their probability distributions. Each probability space is defined from a dimension over the set of passwords.

In particular, we use five probability spaces $\mathcal{P}_i, 1 \leq i \leq 5$ with their corresponding independent probability distribution $\mathbf{P}_i, 1 \leq i \leq 5$ from a given dictionary [9]. In particular, the following are the five dimensions defined over a given password: prefix, suffix, base word, change pattern, and 133t transformation.

- A password prefix is a string that consists of all the digits and symbols preceding the first letter (can be both lowercase and uppercase) found in the password when reading it from left to right. For example, given the password 123@joseherediaxD, its prefix will be 123@ since the first letter from left to right is j. If the first character of a given password is a letter, its prefix is the empty string. For example, the prefix of josedavid2009 is the empty string.
- A password suffix is a string that consists of all the digits and symbols to the right of the first letter (can be both lowercase and uppercase) found in the password when reading it from right to left. For example, given the password 123abc456, its suffix will be 456 since the first letter found from right to left is c. If the last character of a given password is a letter, its suffix is the empty string. For example, the suffix of 23josedavid is the empty string.
- A password base word is the sub-string derived from removing both the password prefix and suffix. For example, if the password is hello12345, the base word is hello. Note that if a password only consists of a sequence of letters, its base word is the password itself.
- A password change pattern is defined as a list of positive and negative indexes indicating the position of capital letters in the password base word. On the one hand, the algorithm counts positive indices from the beginning of the base word, with a zero for the position of the leftmost character. On the other hand, it counts the negative indices from the end of the base word, with the value of $-1$ corresponding to the rightmost character position. Both negative and positive indexes do not exceed the median index (half of the length of the password base word) to avoid ambiguity. For example, if the password base word is RealMadriD123, its change pattern is $[0, 4, -4]$.
- As observed by [22], people tend to mutate passwords using the l33t transformation. A password l33t transformation replaces some characters for visually similar letters in the password base word based on the correspondence defined in Table 1. This transformation is represented as a list of integers in increasing order. More specifically, given the password base word, each occurrence of a character, as appears in Table 1, in the base word is replaced with the corresponding replacement, and so the corresponding integer is appended to the representation list. For example, given the password base word g00dPa$w0rD, it is transformed to goodPasworD, and its representation list is [1, 4].

The mutations shown in Table 1 were selected according to the previous work presented in [9]. In particular, its authors originally used these mutations to form the fifth dimension (together with the other dimensions) for PesRank and showed that PesRank performs and behaves accurately when compared with other password estimation methods (see Section 8 of [9]). Additionally, we note that two different base words may have the same representation list. Essentially, the password l33t transformation defines an equivalence relation in the set of base words. Hence, the probability distribution is computed over the resulting equivalence classes.

**Table 1.** Correspondence for a l33t transformation.

| Character | Replacement | Integer Representation |
|---|---|---|
| 0 | o | 1 |
| 1 | i | 12 |
| ! | i | 13 |
| @ | a | 2 |
| 4 | a | 3 |
| 3 | e | 6 |
| $ | s | 4 |
| 5 | s | 5 |
| 2 | z | 11 |
| % | x | 14 |
| 7 | t | 10 |
| + | t | 9 |
| 9 | g | 8 |
| 6 | g | 7 |

Training Phase

Given a list of passwords $\mathcal{D}$ generated by the dataset generator component, it is processed as follows [9].

1. From the dictionary $\mathcal{D}$, five lists $L_1, L_2, L_3, L_4, L_5$ are constructed. The list $L_1$ contains the prefixes from the passwords in $\mathcal{D}$, $L_2$ contains the corresponding lowercase, l33t-transformed base words, $L_3$ contains the corresponding suffixes, $L_4$ contains the corresponding list representations for the change patterns, and $L_5$ contains the corresponding list representations for the l33t transformations.
2. From each list $L_i, 1 \leq i \leq 5$, we find the frequency of elements in the list and compute a list $L_i^p$, having 2-tuple entries, where an entry is the form $(\mathtt{p}, \mathtt{v})$ with $\mathtt{p}$ being a probability and $\mathtt{v}$ being a unique value. This list $L_i^p$ essentially represents the probability space $\mathcal{P}_i$.
3. Each representation of $\mathcal{P}_i$ is stored persistently in an independent table of a database. This approach allows sorting these data in non-increasing order according to the first component, or querying them efficiently.
4. After computing the representations of each $\mathcal{P}_i$, these representations, along with other parameters, are used to initialize the rank estimator algorithm, ESRank, which returns two lists that are stored persistently. These lists are then internally used by the algorithm to evaluate a given password efficiently.

Since updated dictionaries may be generated by the dataset generator component once in a while, our service has to update the inner workings of the key-rank algorithm by running the training process described above. The running of this lengthy process may be configurable as its running frequency does not overload our service.

Guessability Phase

To evaluate how guessable a given password is, the corresponding function first decomposes it into the dimensions described above, which allows for the calculation of each probability $p_i, 1 \leq i \leq 5$ and so $p_* = p_1 p_2 p_3 p_4 p_5$. By calling the ESrank algorithm, it finds an estimate of the password's rank $r$ in the dictionary used for training [9].

Once the rank $r$ of a given password is estimated, we can calculate $\mathtt{nbits} = \lceil \log_2(r) \rceil$, where `nbits` represents the level of security (strength) for the given password. That is,

based on nbits we can then assign a security level to the given password, since guessing the password would require a full enumeration of $2^{\texttt{nbits}}$ password candidates in non-increasing order of probability to find it based on the dictionary used for training [9]. Furthermore, based on nbits, some explainability may be added to the responses returned to the users utilizing the service [21].

A desirable feature to add to this guessability phase is making this computation privacy preserving. Since the input password is a sensitive piece of information, it should be kept private from this component, i.e., a user may consider it desirable to compute the strength of the given password without the cloud component learning the password itself.

After a user submits their password, it will be out of their control, so the trustworthiness of the cloud servers plays a vital role in data security and privacy. Assuming the cloud components are untrustworthy means the servers may either solve the computational tasks incorrectly or learn what should not be known. Depending on the server's behavior, there are two adversarial models: the first is called the 'honest-but-curious' or 'semi-honest' model and the second is the 'malicious' model. In the honest-but-curious model, servers perform the operations complying with the required computational processes, yet are still curious about the sensitive data submitted by users. Apart from the curiosity of users' data, malicious servers may go against the requested computations and return incorrect results to save their computing power or achieve other intentions for benefits. Hence, if we want the guessability component to process the input passwords in the cloud and provide privacy guarantees to users, this component will require implementing cryptographic techniques. Such cryptographic approaches include homomorphic encryption or secure multiparty computation techniques [65,66].

Instead, for concerned users, a simple approach to guarantee their data privacy is to delegate this computation to their requesting client software application by providing the respective option within it. If such an option is selected, the client application will frequently download the lists returned by EsRank after each initialization (see Step 4) and update previously stored lists (if any). When computing the strength of a given password, the client application will run the corresponding algorithm with the recently downloaded lists and the target password as input, so the passwords never leave the client application.

## 5. Evaluation

In this section, we present the results we obtained from tests we carried out to measure the performance of some components of our service. In particular, we measured the performance of the synthetic dataset component and the password guessability estimator component. We ran our experiments on a PC with AMD Ryzen 5 2600X Six-Core Processor (12 CPUs) at 3.6 GHz, RAM Memory of 16,384 MB RAM, and a Graphics Card NVIDIA GeForce GTX 1060 6GB.

### *5.1. Synthetic Dataset Generator*

This subsection is devoted to describing the tests we carried out for measuring the performance of our synthetic dataset generator component.

#### 5.1.1. Training Performance

Regarding this component's training, we used a dataset of filtered passwords. In particular, we used one from LinkedIn [8,67], experimented with PassGAN to select passwords up to 17 characters long, and performed 100,000 iterations. The other parameters were left by default. We note that the training lasted for about five days. Finally, once PassGAN is trained, we can use it to generate password dictionaries of the desired length.

#### 5.1.2. Generation Performance

Because this component has to generate synthetic datasets as input to the password guessability estimator component, we first measure its time to generate variable-length datasets. Table 2 shows the generation time for variable-length datasets.

**Table 2.** Generation time for various sets of passwords.

| Number of Generated Passwords | Generation Time in Seconds |
|---|---|
| 10,000,000 | 277.98 |
| 20,000,000 | 538.62 |
| 30,000,000 | 810.47 |
| 40,000,000 | 1083.25 |
| 50,000,000 | 1343.49 |
| 60,000,000 | 1637.67 |
| 70,000,000 | 1911.35 |
| 80,000,000 | 2186.22 |
| 90,000,000 | 2450.27 |
| 100,000,000 | 2754.48 |

### *5.2. Password Guessability Estimator*

This subsection is devoted to describing the tests we carried out for measuring the performance of our password guessability estimator component.

### *5.3. Training Performance*

It is possible to generate variable-length dictionaries containing variable-length passwords to exploit our synthetic dataset generator. We performed an experiment that consisted of training our password guessability estimator component and PESrank's [9] with multiple datasets generated by our dataset generator, and then compared our component's training times with PESrank's.

From Figures 2 and 3, we can see that our password guessability estimator component has a better performance when compared with the PESRank implementation [9]. This performance improvement is because the PESRank algorithm performs multiple operations in memory through lists and accesses disk-to-handle text files, which causes a higher degree of complexity than our estimator component, which uses operations directly with the PostgresSQL databases, the queries of which are faster. Therefore, the execution time is shorter. Note that Figures 2 and 3 are complementary, and both show that our component is highly performant when training.

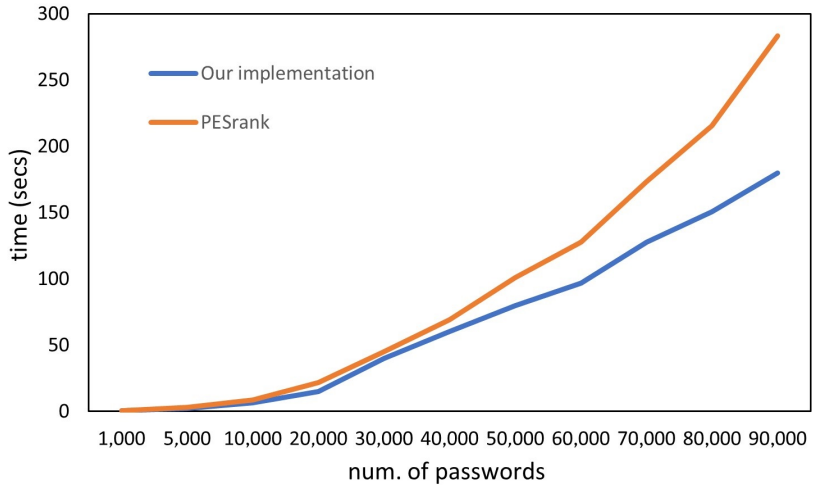


**Figure 2.** Our password guessability estimator component vs. PESRank (1 k–90 k). The x-axis represents the number of passwords in the input dataset, while the y-axis represents time in seconds.

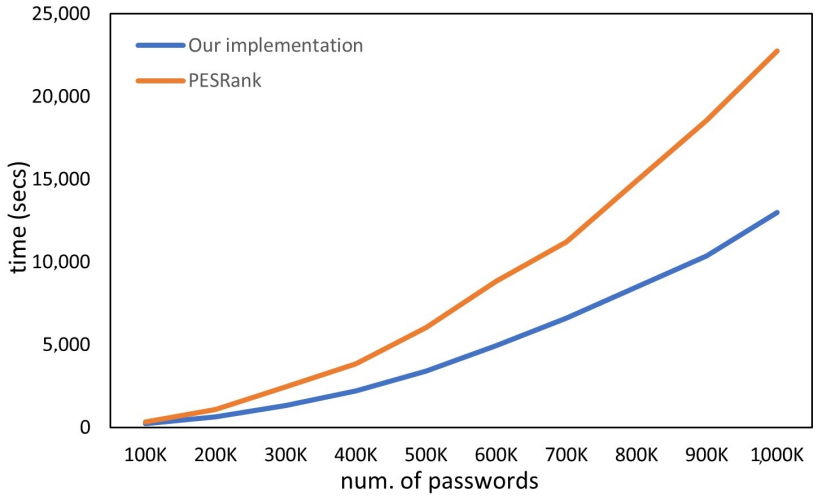


**Figure 3.** Our password guessability estimator component vs. PESRank (100 k–1000 k). The x-axis represents the number of passwords in the input dataset, while the y-axis represents time in seconds.

### *5.4. Evaluation Requests Performance*

To test the performance of requests for evaluating the strength of a password, we carried out an experiment that consisted of sending 1000 requests from a local requester (co-located in the same network as the service) and from a remote requester (located on the Internet). The results for both scenarios are shown in Figures 4 and 5, respectively. These figures show the response time variability in seconds to local and remote requests, highlighting that the service is highly performant. On the other hand, Figure 4 shows that the service serves local requests in 0.088 s on average, with a variability of 0.008 s, and that the request times are in the range [0.078, 0.22]. On the other hand, Figure 5 shows that the service serves remote requests in 6.03 s on average, with a variability of 0.82 s, and that the request times are in the range [3.72, 9.80].

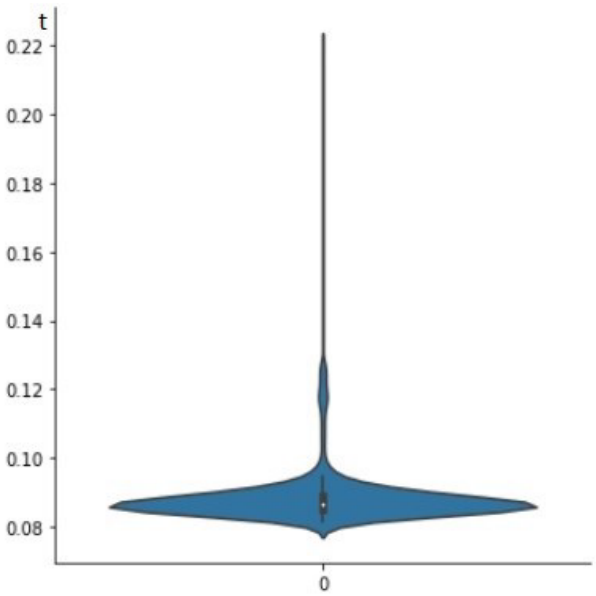


**Figure 4.** Response time variability in seconds from requests by a local requester.

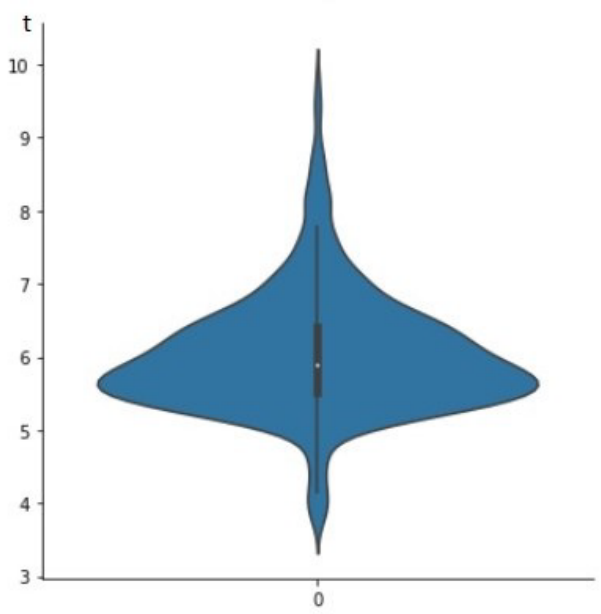


**Figure 5.** Response time variability in seconds from requests by a remote requester.

### 5.5. Password Strength

Recent studies evaluated the strength of a password as the number of attempts an attack would take to guess it. With this metric, a password with a strength $2^{\texttt{nbits}}$ can be considered as strong as a symmetric `nbits`-bit encryption key, since an attacker can guess both with the same effort [68]. Based on this observation, a series of ranges were established to determine the strength of a password. In particular, a password is considered very weak if its strength in bits is less than 30 and very strong if its strength in bits is greater than 90. In particular, Table 3 shows this classification.

**Table 3.** Classification of strength of a password.

| Strength in Bits | Classification |
|---|---|
| $\texttt{nbits} \leq 30$ | Very weak |
| $30 < \texttt{nbits} \leq 50$ | Weak |
| $50 < \texttt{nbits} \leq 70$ | Acceptable |
| $70 < \texttt{nbits} \leq 90$ | Strong |
| $\texttt{nbits} > 90$ | Very Strong |

Moreover, it is possible to express the time in years, months, days, hours, minutes, and seconds that it would take an average computer to enumerate $2^{\texttt{nbits}}$ passwords via a brute-force attack. However, this is a machine-dependent measurement.

### 5.6. Results

Based on the API created in the Python programming language, a web and mobile service was implemented that sends requests to the said API, and for which the response expresses both the degree of strength of the password entered (as previously observed in Table 3) and the time it would take an average computer to enumerate its number of bits. The source code for the web implementation is available upon request. It was possible to check the strength of a password based on X pre-trained passwords using PassGAN. In turn, it could be observed that depending on the set of passwords (stored in the Postgresql database) used as a training source for the model, a password that is vulnerable under human logic and easy for the implemented algorithm is not necessarily

so. That is why PassGAN is significant and represents a novelty compared with that implemented in PESRank, since every particular time in days, the passwords used to train the model are retrained, which improves the algorithm's ability to identify passwords that were previously undetected as easy to crack, thus ensuring better efficiency in detecting vulnerable and non-vulnerable passwords. On the other hand, the efficiency of password security detection improves as the set of passwords in the database increases. In turn, it is observed that the lower the probability of each dimension of a given password, the greater the number of bits necessary to enumerate it, and consequently, it is more secure.

Finally, the message displayed for each password is accompanied by the time it would take for an average computer to enumerate the password using a brute-force attack, thus allowing greater clarity regarding the security of the password entered.

## 6. Conclusions and Future Works

This paper presents a micro-service architecture that can offer an adaptable password strength estimator through a Security as a Service model. The interested reader may browse the source code of its implementation on Github [69]. The architecture as depicted adapts to today's requirements for cloud-based applications in terms of how scalable, heterogeneous and modular they should be. Thus, this is the first password security level validation offered as a cloud-based service to benefit from all the features and goals expected for cloud applications. The model of offering an application as a service is becoming very popular among cloud-based applications because it allows clients to use or pay for precisely the amount of service required each month, which adapts usage and payments according to what is necessary.

The proposal uses frequently updated datasets as input to the password validation tool to better reflect tendencies in password creation, contrary to what password strength estimators usually use. Training models on large datasets might not be possible on resource-constrained devices. For example, on IoT networks, devices rely on edge computing layers. These layers are composed of resource-constrained computers. Those devices running on this layer might need to train and update trained models with a particular frequency. The security-as-a-service model allows these devices to request parameters from the database service, which will run on a dynamically updated dataset. Thanks to the dynamic dataset, an adaptable password strength estimator is produced, which architects could quickly tailor to different languages as needed. The literature has shown the importance of adapting estimators to how different users from different populations create passwords, whether because of language or demographics.

In terms of future works, we plan to evaluate the architecture on different cloud computing configurations to measure how the platform could influence the tool's performance. This evaluation also includes escalating the architecture to a cluster deployment, where different orchestration systems might influence future architecture design changes. The purpose of these future evaluations is to test the architecture as the deployment escalates to a distributed fashion to address a possible exponential increase in user requests.

Another future work is to engineer the guessability component to perform its computations privately in the cloud. We plan to research how to design and build this component to be privacy preserving, and how to make it secure in the semi-honest model by employing cryptographic techniques, such as homomorphic encryption techniques or secure multiparty computation techniques.

**Author Contributions:** Conceptualization, R.V.-P. and M.J.; methodology, R.V.-P.; software, J.B., D.D. and G.M.; validation, R.V.-P.; formal analysis, R.V.-P.; investigation, R.V.-P., M.J., J.B., D.D. and G.M.; resources, G.M.; data curation, D.D.; writing—original draft preparation, J.B. and G.M.; writing—review and editing, R.V.-P. and M.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** The APC was funded by Universidad del Norte.

**Data Availability Statement:** Publicly available datasets were used and analyzed in this study. This data can be found here [67].

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. Murray, H.; Malone, D. Convergence of Password Guessing to Optimal Success Rates. *Entropy* **2020**, *22*, 378. [CrossRef] [PubMed]
2. Ibrokhimov, S.; Hui, K.L.; Abdulhakim Al-Absi, A.; Lee, H.J.; Sain, M. Multi-factor authentication in cyber physical system: A state of art survey. In Proceedings of the 2019 21st International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 17–20 February 2019; pp. 279–284. [CrossRef]
3. Dell'Amico, M.; Michiardi, P.; Roudier, Y. Password strength: An empirical analysis. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14 March 2010; pp. 1–9. [CrossRef]
4. Ur, B.; Bees, J.; Segreti, S.M.; Bauer, L.; Christin, N.; Cranor, L.F. Do users' perceptions of password security match reality? In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), San Jose, CA, USA, 7–12 May 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 3748–3760. [CrossRef]
5. Hald, S.L.; Pedersen, J.M. An updated taxonomy for characterizing hackers according to their threat properties. In Proceedings of the 2012 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 19–22 February 2012; pp. 81–86.
6. Openwall. John the Ripper Password Cracker. Available online: https://www.openwall.com/john/ (accessed on 30 September 2021).
7. Williams, R. Hashcat–Advanced Password Recovery. Available online: https://hashcat.net/hashcat/ (accessed on 30 September 2021).
8. Hitaj, B.; Gasti, P.; Ateniese, G.; Perez-Cruz, F. PassGAN: A deep learning approach for password guessing. In *Applied Cryptography and Network Security*; Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 217–237.
9. David, L.; Wool, A. Online Password Guessability via Multi-Dimensional Rank Estimation. *arXiv* **2020**, arXiv:1912.02551.
10. Mehra, R.; Meshram, A.; Chandavarkar, B.R. Remote user authentication and issues: A survey. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–6. [CrossRef]
11. Shah, R.H.; Salapurkar, D.P. A multifactor authentication system using secret splitting in the perspective of Cloud of Things. In Proceedings of the 2017 International Conference on Emerging Trends Innovation in ICT (ICEI), Pune, India, 3–5 February 2017; pp. 1–4. [CrossRef]
12. Varadharajan, V.; Tupakula, U. Security as a Service Model for Cloud Environment. *IEEE Trans. Netw. Serv. Manag.* **2014**, *11*, 60–75. [CrossRef]
13. Hawedi, M.; Talhi, C.; Boucheneb, H. Security as a Service for Public Cloud Tenants(SaaS). *Procedia Comput. Sci.* **2018**, *130*, 1025–1030. [CrossRef]
14. Torkura, K.A.; Sukmana, M.I.; Cheng, F.; Meinel, C. Leveraging cloud native design patterns for security-as-a-service applications. In Proceedings of the 2017 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 3–5 November 2017; pp. 90–97. [CrossRef]
15. Blanc, G.; Kheir, N.; Ayed, D.; Lefebvre, V.; de Oca, E.M.; Bisson, P. Towards a 5G security architecture: Articulating software-defined security and security as a service. In Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018), Hamburg, Germany, 27–30 August 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
16. Chawla, S.; Thamilarasu, G. Security as a service: Real-time intrusion detection in Internet of things. In Proceedings of the Fifth Cybersecurity Symposium (CyberSec '18), 9–10 April 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
17. Raza, M.; Iqbal, M.; Sharif, M.; Haider, W. A Survey of Password Attacks and Comparative Analysis on Methods for Secure Authentication. *World Appl. Sci. J.* **2012**, *19*, 439–444. [CrossRef]
18. Melicher, W.; Ur, B.; Segreti, S.M.; Komanduri, S.; Bauer, L.; Christin, N.; Cranor, L.F. Fast, lean, and accurate: Modeling password guessability using neural networks. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; USENIX Association: Austin, TX, USA, 2016; pp. 175–191.
19. Bošnjak, L.; Sreš, J.; Brumen, B. Brute-force and dictionary attack on hashed real-world passwords. In Proceedings of the 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 21–25 May 2018; pp. 1161–1166. [CrossRef]
20. Zhou, Y.; Feng, D. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. Cryptology ePrint Archive, Report 2005/388. 2005. Available online: https://ia.cr/2005/388 (accessed on 10 November 2021).
21. David, L.; Wool, A. An explainable online password strength estimator. In *Computer Security–ESORICS 2021*; Bertino, E., Shulman, H., Waidner, M., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 285–304.

22. Wheeler, D.L. zxcvbn: Low-budget password strength estimation. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; USENIX Association: Austin, TX, USA, 2016; pp. 157–173.
23. Schafer, C.R.; Pan, L. Password strength estimators trained on the leaked password lists. In *Applications and Techniques in Information Security*; Shankar Sriram, V.S., Subramaniyaswamy, V., Sasikaladevi, N., Zhang, L., Batten, L., Li, G., Eds.; Springer: Singapore, 2019; pp. 219–231.
24. Galbally, J.; Coisel, I.; Sanchez, I. A New Multimodal Approach for Password Strength Estimation—Part II: Experimental Evaluation. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 2845–2860. [CrossRef]
25. Guo, Y.; Zhang, Z. LPSE: Lightweight password-strength estimation for password meters. *Comput. Secur.* **2018**, *73*, 507–518. [CrossRef]
26. Maoneke, P.B.; Flowerday, S.; Isabirye, N. The influence of native language on password composition and security: A socioculture theoretical view. In *ICT Systems Security and Privacy Protection*; Janczewski, L.J., Kutyłowski, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 33–46.
27. AlSabah, M.; Oligeri, G.; Riley, R. Your culture is in your password: An analysis of a demographically-diverse password dataset. *Comput. Secur.* **2018**, *77*, 427–441. [CrossRef]
28. Doucek, P.; Pavlíček, L.; Sedláček, J.; Nedomová, L. Adaptation of password strength estimators to a non-English environment—The Czech experience. *Comput. Secur.* **2020**, *95*, 101757. [CrossRef]
29. Nadareishvili, I.; Mitra, R.; McLarty, M.; Amundsen, M. *Microservice Architecture: Aligning Principles, Practices, and Culture*, 1st ed.; O'Reilly Media, Inc.: Newton, MA, USA, 2016.
30. Zimmermann, O. Microservices tenets. *Comput. Sci.-Res. Dev.* **2017**, *32*, 301–310. [CrossRef]
31. Cerny, T.; Donahoo, M.J.; Trnka, M. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* **2018**, *17*, 29–45. [CrossRef]
32. Haberle, T.; Charissis, L.; Fehling, C.; Nahm, J.; Leymann, F. The Connected Car in the Cloud: A Platform for Prototyping Telematics Services. *IEEE Softw.* **2015**, *32*, 11–17. [CrossRef]
33. Lwakatare, L.E.; Kuvaja, P.; Oivo, M. Relationship of DevOps to Agile, lean and continuous deployment. In *Product-Focused Software Process Improvement*; Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., Mikkonen, T., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 399–415.
34. Monteiro, D.; Gadelha, R.; Maia, P.H.M.; Rocha, L.S.; Mendonça, N.C. Beethoven: An event-driven lightweight platform for microservice orchestration. In *Software Architecture*; Cuesta, C.E., Garlan, D., Pérez, J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 191–199.
35. Zheng, X. Database as a Service-Current Issues and Its Future. *arXiv* **2018**, arXiv:1804.00465.
36. Kovács, A. Comparison of different Linux containers. In Proceedings of the 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, 5–7 July 2017; pp. 47–51. [CrossRef]
37. Ranjan, R.; Benatallah, B.; Dustdar, S.; Papazoglou, M.P. Cloud Resource Orchestration Programming: Overview, Issues, and Directions. *IEEE Internet Comput.* **2015**, *19*, 46–56. [CrossRef]
38. Kubernetes. Production-Grade Container Orchestration. Available online: https://kubernetes.io/ (accessed on 11 October 2021).
39. Docker. Swarm Mode Overview. Available online: https://docs.docker.com/engine/swarm/ (accessed on 11 October 2021).
40. Arnautov, S.; Trach, B.; Gregor, F.; Knauth, T.; Martin, A.; Priebe, C.; Lind, J.; Muthukumaran, D.; O'Keeffe, D.; Stillwell, M.L.; et al. SCONE: Secure Linux containers with Intel SGX. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; USENIX Association: Savannah, GA, USA, 2016; pp. 689–703.
41. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
42. Villanueva-Polanco, R. A Comprehensive Study of the Key Enumeration Problem. *Entropy* **2019**, *21*, 972. [CrossRef]
43. Bernstein, D.J.; Lange, T.; van Vredendaal, C. Tighter, Faster, Simpler Side-Channel Security Evaluations beyond Computing Power. Cryptology ePrint Archive, Report 2015/221. 2015. Available online: http://eprint.iacr.org/2015/221 (accessed on 5 November 2021).
44. Bogdanov, A.; Kizhvatov, I.; Manzoor, K.; Tischhauser, E.; Witteman, M. Fast and memory-efficient key recovery in side-channel attacks. In *Selected Areas in Cryptography–SAC 2015*; Dunkelman, O., Keliher, L., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 310–327.
45. Choudary, M.O.; Popescu, P.G. Back to massey: Impressively fast, scalable and tight security evaluation tools. In *Cryptographic Hardware and Embedded Systems–CHES 2017*; Fischer, W., Homma, N., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 367–386.
46. Choudary, M.O.; Poussier, R.; Standaert, F.X. Score-based vs. probability-based enumeration–A cautionary note. In *Progress in Cryptology–INDOCRYPT 2016*; Dunkelman, O., Sanadhya, S.K., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 137–152.
47. David, L.; Wool, A. A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In *Topics in Cryptology–CT-RSA 2017*; Handschuh, H., Ed.; Springer International Publishing: Cham, Switzerland, 2017; pp. 311–327.
48. Glowacz, C.; Grosso, V.; Poussier, R.; Schüth, J.; Standaert, F.X. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption*; Leander, G., Ed.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 117–129.

49. Grosso, V. Scalable key rank estimation (and key enumeration) algorithm for large keys. In *Smart Card Research and Advanced Applications*; Bilgin, B., Fischer, J.B., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 80–94.
50. Longo, J.; Martin, D.P.; Mather, L.; Oswald, E.; Sach, B.; Stam, M. How Low Can You Go? Using Side-Channel Data to Enhance Brute-Force Key Recovery. Cryptology ePrint Archive, Report 2016/609. 2016. Available online: http://eprint.iacr.org/2016/609 (accessed on 15 November 2021).
51. Martin, D.P.; Mather, L.; Oswald, E.; Stam, M. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In *Advances in Cryptology–ASIACRYPT 2016*; Cheon, J.H., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 548–572.
52. Martin, D.P.; O'Connell, J.F.; Oswald, E.; Stam, M. Counting keys in parallel after a side channel attack. In *Advances in Cryptology–ASIACRYPT 2015*; Iwata, T., Cheon, J.H., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 313–337.
53. Poussier, R.; Standaert, F.X.; Grosso, V. *Simple Key Enumeration (and Rank Estimation) Using Histograms: An Integrated Approach*; CHES; Springer: Berlin/Heidelberg, Germany, 2016; pp. 61–81. [CrossRef]
54. Poussier, R.; Grosso, V.; Standaert, F.X. Comparing approaches to rank estimation for side-channel security evaluations. In *Smart Card Research and Advanced Applications*; Homma, N., Medwed, M., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 125–142.
55. Veyrat-Charvillon, N.; Gérard, B.; Renauld, M.; Standaert, F.X. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography*; Knudsen, L.R., Wu, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 390–406.
56. Veyrat-Charvillon, N.; Gérard, B.; Standaert, F.X. Security evaluations beyond computing power. In *Advances in Cryptology–EUROCRYPT 2013*; Johansson, T., Nguyen, P.Q., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 126–141.
57. Ye, X.; Eisenbarth, T.; Martin, W. Bounded, yet sufficient? How to determine whether limited side channel information enables key recovery. In *Smart Card Research and Advanced Applications*; Joye, M., Moradi, A., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 215–232.
58. Junod, P.; Vaudenay, S. Optimal key ranking procedures in a statistical cryptanalysis. In *Fast Software Encryption*; Johansson, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 235–246.
59. Paterson, K.G.; Villanueva-Polanco, R. Cold boot attacks on NTRU. In *Progress in Cryptology–INDOCRYPT 2017*; Patra, A., Smart, N.P., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 107–125.
60. Seshadri, N.; Sundberg, C.W. List Viterbi decoding algorithms with applications. *IEEE Trans. Commun.* **1994**, *42*, 313–323. [CrossRef]
61. Villanueva-Polanco, R. Cold Boot Attacks on LUOV. *Appl. Sci.* **2020**, *10*, 4106. [CrossRef]
62. Villanueva-Polanco, R.; Angulo-Madrid, E. Cold Boot Attacks on the Supersingular Isogeny Key Encapsulation (SIKE) Mechanism. *Appl. Sci.* **2021**, *11*, 193. [CrossRef]
63. Villanueva-Polanco, R. Cold boot attacks on bliss. In *Progress in Cryptology–LATINCRYPT 2019*; Schwabe, P., Thériault, N., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 40–61.
64. David, L.; Wool, A. Rank Estimation with Bounded Error via Exponential Sampling. Cryptology ePrint Archive, Report 2021/313. 2021. Available online: https://ia.cr/2021/313 (accessed on 29 November 2021).
65. Acar, A.; Aksu, H.; Uluagac, A.S.; Conti, M. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* **2018**, *51*, 1–35. [CrossRef]
66. Yang, Y.; Huang, X.; Liu, X.; Cheng, H.; Weng, J.; Luo, X.; Chang, V. A Comprehensive Survey on Secure Outsourced Computation and Its Applications. *IEEE Access* **2019**, *7*, 159426–159465. [CrossRef]
67. Gulrajani, I.; Dorsey, B. PassGAN Implementation. GitHub. 2019. Available online: https://github.com/d4ichi/PassGAN (accessed on 26 October 2021).
68. Dell'Amico, M.; Filippone, M. Monte Carlo strength evaluation: Fast and reliable password checking. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15), Denver, CO, USA, 12–16 October 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 158–169. [CrossRef]
69. Bojato, J.; Donado, D.; Jimeno, M.; Moreno, G.; Villanueva-Polanco, R. Password Guessability Service Implementation. GitHub. 2021. Available online: https://github.com/Juandavid716/API_Password_checker (accessed on 29 November 2021).